19th International Symposium on Algorithms and Data Structures

WADS 2025, August 11–15, 2025, York University, Toronto, Canada

Edited by

Pat Morin

Eunjin Oh

LIPIcs – Vol. 349 – WADS 2025 www.dagstuhl.de/lipics

Editors

Pat Morin
School of Computer Science, Carleton University, Ottawa, Canada
morin@scs.carleton.ca

Eunjin Oh
Pohang University of Science and Technology, South Korea
eunjin.oh@postech.ac.kr

ACM Classification 2012
Theory of computation

ISBN 978-3-95977-398-0

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/978-3-95977-398-0.

Publication date
September, 2025

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.WADS.2025.0

ISBN 978-3-95977-398-0 **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)
- Sławomir Lasota (University of Warsaw, PL)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN – Chair)
- Chih-Hao Luke Ong (Nanyang Technological University, SG)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Pierre Senellart (ENS, Université PSL, Paris, France)
- Alexandra Silva (Cornell University, Ithaca, US)

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Preface
Pat Morin and Eunjin Oh 0:ix

Organization
..... 0:xi

List of Authors
..... 0:xiii

Invited Talks

Constructing and Routing on Geometric Spanners
Prosenjit Bose 1:1–1:1

Unintuitive Facts About Distances on Planar Graphs
Hsien-Chih Chang 2:1–2:1

Regular Papers

Support Vector Machines in the Hilbert Geometry
Aditya Acharya, Auguste H. Gezalyan, Julian Vanecek, David M. Mount, and Sunil Arya 3:1–3:20

Evolving Distributions Under Local Motion
Aditya Acharya and David M. Mount 4:1–4:20

On Planar Straight-Line Dominance Drawings
Patrizio Angelini, Michael A. Bekos, Giuseppe Di Battista, Fabrizio Frati, Luca Grilli, and Giacomo Ortali 5:1–5:18

Vantage Point Selection Algorithms for Bottleneck Capacity Estimation
Vikrant Ashvinkumar, Rezaul Chowdhury, Jie Gao, Mayank Goswami, Joseph S. B. Mitchell, and Valentin Polishchuk 6:1–6:19

Approximation and Parameterized Algorithms for Covering with Disks of Two Types of Radii
Sayan Bandyapadhyay and Eli Mitchell 7:1–7:14

Tight Bounds on the Number of Closest Pairs in Vertical Slabs
Ahmad Biniaz, Prosenjit Bose, Chaeyoon Chung, Jean-Lou De Carufel, John Iacono, Anil Maheshwari, Saeed Odak, Michiel Smid, and Csaba D. Tóth ... 8:1–8:14

Online Routing in Directed Yao_4^∞ Graphs
Prosenjit Bose, Jean-Lou De Carufel, and John Stuart 9:1–9:22

On Geodesic Disks Enclosing Many Points
Prosenjit Bose, Guillermo Esteban, David Orden, Rodrigo I. Silveira, and Tyler Tuttle 10:1–10:20

Crossing and Independent Families Among Polygons
Anna Brötzner, Robert Ganian, Thekla Hamm, Fabian Klute, and Irene Parada .. 11:1–11:15

19th International Symposium on Algorithms and Data Structures (WADS 2025).
Editors: Pat Morin and Eunjin Oh
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Testing Whether a Subgraph Is Convex or Isometric
Sergio Cabello 12:1–12:16

Algorithms for Distance Problems in Continuous Graphs
Sergio Cabello, Delia Garijo, Antonia Kalb, Fabian Klute, Irene Parada, and Rodrigo I. Silveira 13:1–13:14

Quantum Speedups for Polynomial-Time Dynamic Programming Algorithms
Susanna Caroppo, Giordano Da Lozzo, Giuseppe Di Battista, Michael T. Goodrich, and Martin Nöllenburg 14:1–14:22

Enumerating Minimal Dominating Sets and Variants in Chordal Bipartite Graphs
Emanuel Castelo, Oscar Defrain, and Guilherme C. M. Gomes 15:1–15:15

An Improved Guillotine Cut for Squares
Parinya Chalermsook, Axel Kugelmann, Ly Orgo, Sumedha Uniyal, and Minoo Zarsav 16:1–16:19

Dynamic Streaming Algorithms for Geometric Independent Set
Timothy M. Chan and Yuancheng Yu 17:1–17:12

Parameterized Streaming Algorithms for Topological Sorting
Ho-Lin Chen, Peng-Ting Lin, and Meng-Tsung Tsai 18:1–18:20

On the Enumeration of Signatures of XOR-CNF's
Nadia Creignou, Oscar Defrain, Frédéric Olive, and Simon Vilmin 19:1–19:14

Routing Few Robots in a Crowded Network
Argyrios Deligkas, Eduard Eiben, Robert Ganian, Iyad Kanj, Dominik Leko, and M. S. Ramanujan 20:1–20:15

A WSPD, Separator and Small Tree Cover for c-Packed Graphs
Lindsey Deryckere, Joachim Gudmundsson, André van Renssen, Yuan Sha, and Sampson Wong 21:1–21:15

On Minimizing Wiggle in Stacked Area Charts
Alexander Dobler and Martin Nöllenburg 22:1–22:14

Novel Complexity Results for Temporal Separators with Deadlines
Riccardo Dondi and Manuel Lafond 23:1–23:14

Computational Geometry with Probabilistically Noisy Primitive Operations
David Eppstein, Michael T. Goodrich, and Vinesh Sridhar 24:1–24:20

An Efficient Polynomial Time Approximation Scheme for Minimizing the Total Weighted Completion Time on Uniformly Related Machines
Leah Epstein and Asaf Levin 25:1–25:21

Lower Bounds for Several Standard Bin Packing Algorithms in the Random Order Model
Leah Epstein and Asaf Levin 26:1–26:15

A QPTAS for Facility Location on Unit Disk Graphs
Zachary Friggstad, Mohsen Rezapour, Mohammad R. Salavatipour, and Hao Sun 27:1–27:18

Approximation Algorithms for the Generalized Point-To-Point Problem
Zachary Friggstad, Mohammad R. Salavatipour, and Hao Sun 28:1–28:16

Linear Layouts of Graphs with Priority Queues
Emilio Di Giacomo, Walter Didimo, Henry Förster, Torsten Ueckerdt, and Johannes Zink 29:1–29:17

Convolution and Knapsack in Higher Dimensions
Kilian Grage, Klaus Jansen, and Björn Schumacher 30:1–30:16

Repairing Schedules by Removing Waiting Times: A Parameterized Complexity Analysis
Niels Grüttemeier and Klaus Heeger 31:1–31:14

Fantastic Flips and Where to Find Them: A General Framework for Parameterized Local Search on Partitioning Problems
Niels Grüttemeier, Nils Morawietz, and Frank Sommer 32:1–32:20

Spanner for the $0/1/\infty$ Weighted Region Problem
Joachim Gudmundsson, Zijin Huang, André van Renssen, and Sampson Wong ... 33:1–33:15

Link Diameter, Radius and 2-Point Link Distance Queries in Polygonal Domains
Mart Hagedoorn and Valentin Polishchuk 34:1–34:15

Succinct Data Structures for Chordal Graph with Bounded Leafage or Vertex Leafage
Meng He and Kaiyu Wu 35:1–35:23

On the Complexity of Minimising the Moving Distance for Dispersing Objects
Nicolás Honorato-Droguett, Kazuhiro Kurita, Tesshu Hanaka, and Hirotaka Ono 36:1–36:14

A Near-Linear Time Exact Algorithm for the L_1-Geodesic Fréchet Distance Between Two Curves on the Boundary of a Simple Polygon
Thijs van der Horst, Marc van Kreveld, Tim Ophelders, and Bettina Speckmann 37:1–37:13

Clustering Point Sets Revisited
Md. Billal Hossain and Benjamin Raichel 38:1–38:16

Sweeping a Domain with Line-Of-Sight Between Covisible Agents
Kien C. Huynh, Joseph S. B. Mitchell, and Valentin Polishchuk 39:1–39:22

Grandchildren-Weight-Balanced Binary Search Trees
Vincent Jugé 40:1–40:19

Scheduling on Identical Machines with Setup Time and Unknown Execution Time
Yasushi Kawase, Kazuhisa Makino, Vinh Long Phan, and Hanna Sumita 41:1–41:16

The Parameterized Landscape of Labeled Graph Contractions
Manuel Lafond and Bertrand Marchand 42:1–42:23

On the Complexity of Finding 1-Center Spanning Trees
Pin-Hsian Lee, Meng-Tsung Tsai, and Hung-Lung Wang 43:1–43:19

Streaming Algorithms for Conflict-Free Coloring
Rogers Mathew, Fahad Panolan, and Seshikanth 44:1–44:18

Fast Kd-Trees for the Kullback–Leibler Divergence and Other Decomposable Bregman Divergences
Tuyen Pham and Hubert Wagner 45:1–45:19

Skipping Ropes: An Efficient Gray Code Algorithm for Generating Wiggly Permutations
Vincent Pilaud and Aaron Williams 46:1–46:20

B-Treaps Revised: Write Efficient Randomized Block Search Trees with High Load
Roodabeh Safavi and Martin P. Seybold 47:1–47:23

Farthest-Point Voronoi Diagrams in the Hilbert Metric
Minju Song, Mook Kwon Jung, and Hee-Kap Ahn 48:1–48:15

On the I/O Complexity of the Cocke-Younger-Kasami Algorithm and of a Family of Related Dynamic Programming Algorithms
Lorenzo De Stefani and Vedant Gupta 49:1–49:24

Deterministic $(2/3 - \varepsilon)$-Approximation of Matroid Intersection Using Nearly-Linear Independence-Oracle Queries
Tatsuya Terao 50:1–50:18

Preface

This volume contains the papers presented at WADS 2025: 19th Algorithms and Data Structures Symposium, held on August 11–August 15, 2025 in Toronto.

WADS, which alternates with the Scandinavian Symposium and Workshops on Algorithm Theory (SWAT), is a venue for researchers in the area of design and analysis of algorithms and data structures to present their work. This year, there were 94 submissions. Each submission was refereed by an average of 3 program committee members. The committee decided to accept 47 papers.

The program also includes three invited talks by Hsien-Chih Chang (Dartmouth College) and Prosenjit Bose (Carleton University). This year, WADS took place back to back with the 37th Canadian Conference on Computational Geometry (CCCG 2025) at York University, in Toronto.

The Alejandro López-Ortiz Best Paper Award for WADS 2025 was awarded to the paper *Testing whether a subgraph is convex or isometric* by Sergio Cabello.

We thank the Program Committee for their hard work and good judgement and thank all the subreviewers who contributed to the reviewing process. We thank Shahin Kamali and Eric Ruppert at York Unversity for taking on the job of local organizer for both WADS and CCCG.

July 2025

Pat Morin
Eunjin Oh

19th International Symposium on Algorithms and Data Structures (WADS 2025).
Editors: Pat Morin and Eunjin Oh
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

Program Committee

Davide Bilo	University of L'Aquila
Ahmad Biniaz	University of Windsor
Gerth Stølting Brodal	Aarhus University
Yixin Cao	The Hong Kong Polytechnic University
Jean Cardinal	Universite Libre de Bruxelles
Leah Epstein	University of Haifa
Zachary Friggstad	University of Alberta
Pawel Gawrychowski	University of Wroclaw
Sándor Kisfaludi-Bak	Aalto University
Pranabendu Misra	Chennai Mathematical Institute
Joseph Mitchell	Stony Brook University
Pat Morin	Carleton University
André Nusser	CNRS, France
Martin Nöllenburg	TU Wien
Eunjin Oh	POSTECH
Denis Pankratov	Concordia University
Eric Ruppert	York University
Kunihiko Sadakane	The University of Tokyo
Rodrigo Silveira	Politècnica de Catalunya
Michiel Smid	Carleton University
Frank Staals	Utrecht University
Leonidas Theocharous	Eindhoven University of Technology
Csaba Toth	California State University Northridge
Sebastian Wiederrecht	KAIST
Jie Xue	NYU Shanghai
Lena Yuditsky	Université libre de Bruxelles

19th International Symposium on Algorithms and Data Structures (WADS 2025).
Editors: Pat Morin and Eunjin Oh
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Aditya Acharya (3, 4)
Department of Computer Science, University of Maryland, College Park, MD, USA

Hee-Kap Ahn (48)
Graduate School of Artificial Intelligence, Department of Computer Science and Engineering, Pohang University of Science and Technology, Republic of Korea

Patrizio Angelini (5)
John Cabot University, Rome, Italy

Sunil Arya (3)
Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong

Vikrant Ashvinkumar (6)
Rutgers University, Piscataway, NJ, USA

Sayan Bandyapadhyay (7)
Portland State University, OR, USA

Michael A. Bekos (5)
University of Ioannina, Greece

Ahmad Biniaz (8)
School of Computer Science, University of Windsor, Canada

Prosenjit Bose (1, 8, 9, 10)
School of Computer Science, Carleton University, Ottawa, Canada

Anna Brötzner (11)
Malmö University, Sweden

Guilherme C. M. Gomes (15)
LIRMM, Université de Montpellier, CNRS, Montpellier, France;
Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Sergio Cabello (12, 13)
Faculty of Mathematics and Physics, University of Ljubljana, Slovenia;
Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia

Susanna Caroppo (14)
Roma Tre University, Rome, Italy

Emanuel Castelo (15)
Aix-Marseille Université, CNRS, LIS, Marseille, France

Parinya Chalermsook (16)
University of Sheffield, England

Timothy M. Chan (17)
Siebel School of Computing and Data Science, University of Illinois Urbana-Champaign, IL, USA

Hsien-Chih Chang (2)
Department of Computer Science, Dartmouth College, Hanover, NH, USA

Ho-Lin Chen (18)
National Taiwan University, Taipei, Taiwan

Rezaul Chowdhury (6)
Stony Brook University, NY, USA

Chaeyoon Chung (8)
Department of Computer Science and Engineering, Pohang University of Science and Technology, Republic of Korea

Nadia Creignou (19)
Aix-Marseille Université, CNRS, LIS, Marseille, France

Giordano Da Lozzo (14)
Roma Tre University, Rome, Italy

Jean-Lou De Carufel (8, 9)
School of Electrical Engineering and Computer Science, University of Ottawa, Canada

Lorenzo De Stefani (49)
Department of Computer Science, Brown University, Providence, RI, USA

Oscar Defrain (15, 19)
Aix-Marseille Université, CNRS, LIS, Marseille, France

Argyrios Deligkas (20)
Department of Computer Science, Royal Holloway, University of London, Egham, UK

Lindsey Deryckere (21)
The University of Sydney, Australia

Giuseppe Di Battista (5, 14)
Università degli Studi Roma Tre, Rome, Italy

Emilio Di Giacomo (29)
University of Perugia, Italy

Walter Didimo (29)
University of Perugia, Italy

19th International Symposium on Algorithms and Data Structures (WADS 2025).
Editors: Pat Morin and Eunjin Oh
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Alexander Dobler (22)
TU Wien, Austria

Riccardo Dondi (23)
Università degli Studi di Bergamo, Italy

Eduard Eiben (20)
Department of Computer Science, Royal Holloway, University of London, Egham, UK

David Eppstein (24)
University of California, Irvine, CA, USA

Leah Epstein (25, 26)
Department of Mathematics, Faculty of Natural Sciences, University of Haifa, Israel

Guillermo Esteban (10)
Departamento de Física y Matemáticas, Universidad de Alcalá, Alcalá de Henares, Spain

Fabrizio Frati (5)
Università degli Studi Roma Tre, Rome, Italy

Zachary Friggstad (27, 28)
Department of Computing Science, University of Alberta, Edmonton, Canada

Henry Förster (29)
Technical University of Munich, Heilbronn, Germany

Robert Ganian (11, 20)
Algorithms and Complexity Group, TU Wien, Austria

Jie Gao (6)
Rutgers University, Piscataway, NJ, USA

Delia Garijo (13)
University of Seville, Spain

Auguste H. Gezalyan (3)
Department of Computer Science, University of Maryland, College Park, MD, USA

Michael T. Goodrich (14, 24)
University of California, Irvine, CA, USA

Mayank Goswami (6)
Queens College, City University of New York, NY, USA

Kilian Grage (30)
Kiel University, Germany

Luca Grilli (5)
Università degli Studi di Perugia, Italy

Niels Grüttemeier (31, 32)
Fraunhofer IOSB-INA, Lemgo, Germany

Joachim Gudmundsson (21, 33)
The University of Sydney, Australia

Vedant Gupta (49)
Department of Computer Science, Brown University, Providence, RI, USA

Mart Hagedoorn (34)
Technische Universität Dortmund, Germany

Thekla Hamm (11)
TU Eindhoven, The Netherlands

Tesshu Hanaka (36)
Kyushu University, Fukuoka, Japan

Meng He (35)
Faculty of Computer Science, Dalhousie University, Halifax, Canada

Klaus Heeger (31)
Fraunhofer IOSB-INA, Lemgo, Germany

Nicolás Honorato-Droguett (36)
Nagoya University, Japan

Md. Billal Hossain (38)
Department of Computer Science, University of Texas at Dallas, TX, USA

Zijin Huang (33)
The University of Sydney, Australia

Kien C. Huynh (39)
Communications and Transport Systems, ITN, Linköping University, Sweden

John Iacono (8)
Department of Computer Science, Université libre de Bruxelles, Brussels, Belgium

Klaus Jansen (30)
Kiel University, Germany

Vincent Jugé (40)
LIGM, Univ Gustave Eiffel & CNRS, Marne-la-Vallée, France;
IRIF, Université Paris Cité & CNRS, France

Mook Kwon Jung (48)
Department of Computer Science and Engineering, Pohang University of Science and Technology, Republic of Korea

Antonia Kalb (13)
Technical University of Dortmund, Germany

Iyad Kanj (20)
School of Computing, DePaul University, Chicago, IL, USA

Yasushi Kawase (41)
The University of Tokyo, Japan

Fabian Klute (11, 13)
Universitat Politècnica de Catalunya, Barcelona, Spain

Axel Kugelmann (16)
PSL Research University, Paris, France

Kazuhiro Kurita (36)
Nagoya University, Japan

Manuel Lafond (23, 42)
Université de Sherbrooke, Canada

Pin-Hsian Lee (43)
National Taiwan Normal University, Taipei, Taiwan

Dominik Leko (20)
Algorithms and Complexity Group, TU Wien, IL, Austria

Asaf Levin (25, 26)
Faculty of Data and Decisions Science, The Technion, Haifa, Israel

Peng-Ting Lin (18)
Academia Sinica, Taipei, Taiwan

Anil Maheshwari (8)
School of Computer Science, Carleton University, Ottawa, Canada

Kazuhisa Makino (41)
Kyoto University, Japan

Bertrand Marchand (42)
Department of Computer Science, University of Sherbrooke, Canada

Rogers Mathew (44)
Department of Computer Science and Engineering, IIT Hyderabad, India

Eli Mitchell (7)
Department of Computer Science, Portland State University, OR, USA

Joseph S. B. Mitchell (6, 39)
Stony Brook University, NY, USA

Nils Morawietz (32)
Institute of Computer Science, Friedrich Schiller University Jena, Germany;
LaBRI, Université de Bordeaux, Talence, France

David M. Mount (3, 4)
Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA

Martin Nöllenburg (14, 22)
TU Wien, Austria

Saeed Odak (8)
School of Electrical Engineering and Computer Science, University of Ottawa, Canada

Frédéric Olive (19)
Aix-Marseille Université, CNRS, LIS, Marseille, France

Hirotaka Ono (36)
Nagoya University, Japan

Tim Ophelders (37)
Department of Information and Computing Sciences, Utrecht University, The Netherlands;
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

David Orden (10)
Departamento de Física y Matemáticas, Universidad de Alcalá, Alcalá de Henares, Spain

Ly Orgo (16)
Aalto University, Finland

Giacomo Ortali (5)
Università degli Studi di Perugia, Italy

Fahad Panolan (44)
School of Computer Science, University of Leeds, UK

Irene Parada (11, 13)
Universitat Politècnica de Catalunya, Barcelona, Spain

Tuyen Pham (45)
University of Florida, Gainesville, FL, USA

Vinh Long Phan (41)
Toyota Motor Corporation, Japan

Vincent Pilaud (46)
Departament de Matemàtiques i Informàtica, Universitat de Barcelona, Spain

Valentin Polishchuk (6, 34, 39)
Linköping University, Sweden

Benjamin Raichel (38)
Department of Computer Science, University of Texas at Dallas, TX, USA

M. S. Ramanujan (20)
Department of Computer Science, University of Warwick, UK

Mohsen Rezapour (27)
Department of Computing Science, University of Alberta, Edmonton, Canada

Roodabeh Safavi (47)
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Mohammad R. Salavatipour (27, 28)
Department of Computing Science, University of Alberta, Edmonton, Canada

Björn Schumacher (30)
Kiel University, Germany

Seshikanth (44)
Department of Computer Science and Engineering, IIT Hyderabad, India

Martin P. Seybold (47)
Faculty of Computer Science, Theory and Applications of Algorithms, University of Vienna, Austria

Yuan Sha (21)
The University of Sydney, Australia

Rodrigo I. Silveira (10, 13)
Departament de Matemàtiques, Universitat Politècnica de Catalunya, Barcelona, Spain

Michiel Smid (8)
School of Computer Science, Carleton University, Ottawa, Canada

Frank Sommer (32)
Institute of Logic and Computation, TU Wien, Austria

Minju Song (48)
Graduate School of Artificial Intelligence, Pohang University of Science and Technology, Republic of Korea

Bettina Speckmann (37)
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Vinesh Sridhar (24)
University of California, Irvine, CA, USA

John Stuart (9)
School of Electrical Engineering and Computer Science, University of Ottawa, Canada

Hanna Sumita (41)
Institute of Science Tokyo, Japan

Hao Sun (27, 28)
Department of Computer Science, University of Houston, TX, USA

Tatsuya Terao (50)
Research Institute for Mathematical Sciences, Kyoto University, Japan

Meng-Tsung Tsai (18, 43)
Academia Sinica, Taipei, Taiwan

Tyler Tuttle (10)
School of Computer Science, Carleton University, Ottawa, Canada

Csaba D. Tóth (8)
Department of Mathematics, California State University Northridge, Los Angeles, CA, USA; Department of Computer Science, Tufts University, Medford, MA, USA

Torsten Ueckerdt (29)
Karlsruhe Institute of Technology, Germany

Sumedha Uniyal (16)
Aalto University, Finland

Thijs van der Horst (37)
Department of Information and Computing Sciences, Utrecht University, The Netherlands; Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Marc van Kreveld (37)
Department of Information and Computing Sciences, Utrecht University, The Netherlands

André van Renssen (21, 33)
The University of Sydney, Australia

Julian Vanecek (3)
Department of Computer Science, University of Maryland, College Park, MD, USA

Simon Vilmin (19)
Aix-Marseille Université, CNRS, LIS, Marseille, France

Hubert Wagner (45)
University of Florida, Gainesville, Fl, USA

Hung-Lung Wang (43)
National Taiwan Normal University, Taipei, Taiwan

Aaron Williams (46)
Department of Computer Science, Williams College, Williamstown, MA, USA

Sampson Wong (21, 33)
The University of Copenhagen, Denmark

Kaiyu Wu (35)
Faculty of Computer Science, Dalhousie University, Halifax, Canada

Yuancheng Yu (17)
Siebel School of Computing and Data Science, University of Illinois Urbana-Champaign, IL, USA

Minoo Zarsav (16)
Aalto University, Finland

Johannes Zink (29)
Technical University of Munich, Heilbronn, Germany

Constructing and Routing on Geometric Spanners

Prosenjit Bose
School of Computer Science, Carleton University, Ottawa, Canada

Abstract

A geometric graph $G = (V, E)$ is a graph whose vertex set V is a set of points in the plane and whose edge set E is a set of segments joining vertices. Typically, the edges are weighted with the Euclidean distance between their endpoints and we refer to such graphs as *Euclidean geometric graphs*. A spanning subgraph H of a Euclidean geometric graph G is a *t-spanner* of G provided that for every edge $xy \in G$, the shortest path between x and y in H has weight that is no more than t times the weight of the edge xy. The smallest constant t for which H is a t-spanner of G is its *spanning ratio*.

An online routing algorithm $\mathcal{A}$ is an algorithm that finds a short path in a graph without having full knowledge of the graph. The routing ratio of $\mathcal{A}$ is analogous to the spanning ratio except that the ratio is with respect to the weight of the path followed by the routing algorithm as opposed to the shortest path. Thus, the routing ratio, by definition, is an upper bound on the spanning ratio.

In this talk, we will review results and present open problems on different variants of the problem of constructing and routing on geometric t-spanners.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases Geometric Spanners, Local Routing Algorithms

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.1

Category Invited Talk

Funding This research is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

© Prosenjit Bose;
licensed under Creative Commons License CC-BY 4.0

Unintuitive Facts About Distances on Planar Graphs

Hsien-Chih Chang
Department of Computer Science, Dartmouth College, Hanover, NH, USA

Abstract

Conventional wisdom told us that planar graphs are essentially edge-weighted grids, with more or less equal side-lengths. An n-node $n^{1/2}$-by-$n^{1/2}$ square grid has treewidth $\Theta(n^{1/2})$; and if we want to preserve shortest-path distances between every pair of boundary nodes, intuitively we have to keep all the $n^{1/2}$ column and row paths, which together create n "crossings" that cannot be removed. This seems to suggest that planar graphs are incompressible and not tree-like. Or does it?

In this talk we will discuss three unintuitive, and perhaps surprising, facts about planar metrics in the $(1+\varepsilon)$-approximation regime. First we demonstrate how to construct emulator for planar graphs that preserves all-pair distances between k terminals, and has size $\tilde{O}_\varepsilon(k)$. (This implies, for the grid example above, the resulting emulator has size $\tilde{O}(n^{1/2})$.)

Second, planar metrics can be covered using constantly(!) many trees, in the sense that we can construct $O(1)$ many trees independent to the size of the input graph that never shrinks distances, so that given any pair of nodes x and y, there is one tree T that contains both x and y whose distance on T is stretched by at most a $1+\varepsilon$ factor. Along the way we will introduce a novel structure on planar metrics – the gridtrees – that enables such tree covers, as well as its applications in the resolution to the Steiner point removal problem, and in constructing embeddings of planar graphs into polylog-treewidth graphs with $(1+\varepsilon)$-distortion. (Which means, if we are willing to distort the distance by a small amount, planar metrics are very much tree-like.)

Finally, we will discuss the issue of *spanning*. Both results above rely on the fact that the emulator and the tree cover use "Steiner nodes", which are nodes not presented in the original input graph. Maybe this is cheating, and the distance compression is only possible because of these nodes that appear out of nowhere? Our goal is to convince you otherwise: We can in fact construct emulators for planar graphs that are *minors*, which only uses paths and edges from the input planar graph; and in the case of tree covers, we are one or two new structures away from enforcing the trees to be spanning, that is, the edges in the trees have come from the input graph as well.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners; Theory of computation → Shortest paths

Keywords and phrases planar, emulator, tree cover, gridtree, spanning, sparsifier, tree embedding, clustering, Baker's technique, KPR decomposition, low-diameter decomposition, quadtree, shortest-path separator, portal

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.2

Category Invited Talk

Funding The research is partially supported by NSF CAREER award CCF-2443017.

Acknowledgements The results we discuss in this talk are a combination of my research in the past few years with many collaborators. The emulator result is from STOC 2022 with Zihan Tan and Robert Krauthgamer; the tree cover result comes from the papers from FOCS 2023 and SODA 2024, which is a joint work with the tree cover group consisting of Jonathan Conroy, Hung Le, Lazar Milenkovic, Shay Solomon, and Cuong Than. The polylog-treewidth embedding result is a collaboration with Vincent Cohen-Addad, Jonathan Conroy, Hung Le, Marcin Pilipczuk, and Michał Pilipczuk in SODA 2025. The distance approximating minor result is from a work recently accepted to FOCS 2025 with Jonathan Conroy, and the spanning tree cover result is again by the tree cover group, accepted to STOC 2025.

© Hsien-Chih Chang;
licensed under Creative Commons License CC-BY 4.0

Support Vector Machines in the Hilbert Geometry

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

Julian Vanecek
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

Sunil Arya
Department of Computer Science and Engineering,
Hong Kong University of Science and Technology, Hong Kong

Abstract

Support Vector Machines (SVMs) are a class of classification models in machine learning that are based on computing a maximum-margin separator between two sets of points. The SVM problem has been heavily studied for Euclidean geometry and for a number of kernels. In this paper, we consider the linear SVM problem in the Hilbert metric, a non-Euclidean geometry defined over a convex body. We present efficient algorithms for computing the SVM classifier for a set of n points in the Hilbert metric defined by convex polygons in the plane and convex polytopes in d-dimensional space. We also consider the problems in the related Funk distance.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Support vector machines, Hilbert geometry, linear classification, machine learning, LP-type problems

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.3

Funding *Sunil Arya*: Research Grants Council of Hong Kong, China project number 16214721.

1 Introduction

Support vector machines (SVMs) were introduced by Vapnik and Chervonenkis [34] as a type of classification algorithm [12]. They take two classes of data that are embedded as points in a multidimensional feature space. Once embedded, the algorithm attempts to separate them using a kernel function, which is chosen to maximize the separation distance between the two classes. Throughout, we assume linear kernels, which means that points are separated by a hyperplane.

Two sets of points $A, B \subset \mathbb{R}^d$ are said to be *linearly separable* if there exists a $(d-1)$-dimensional hyperplane h such that A and B lie on opposite sides of h, possibly on h itself (see Figure 1(a)). Define $\mathrm{Sep}(A, B)$ to be the set of separating hyperplanes. The minimum distance from either set to a separating hyperplane is called its *margin*. Given two linearly separable point sets A and B in $\mathbb{R}^d$, the (linear) *SVM problem* is that of computing the hyperplane $h \in \mathrm{Sep}(A, B)$ that maximizes the margin (see Figure 1(b)). The name arises from the fact that, assuming general position, there are $d+1$ points that achieve the margin exactly, called the *support vectors*.

SVMs are a classic tool in machine learning [5, 13, 28, 31]. They are highly regarded as a classification method due to their geometric interoperability, robustness against over-fitting, practicality with small and high dimensional datasets, and success in generalizing to unseen

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

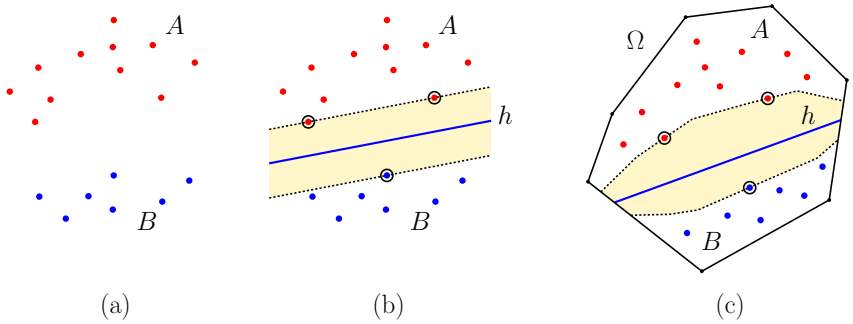


Figure 1 (a) The point sets, (b) Euclidean SVM, (c) and the Hilbert SVM.

data. SVMs are widely used in the fields of vision [23,32], natural language processing [15,22], and computational biology [4,21]. In particular, SVMs have seen a lot of use in image [8,16] and text classification problems [33].

SVMs are well studied in standard Euclidean geometry and other common geometries, such as hyperbolic geometry [10]. They have also been studied relative to a variety of different kernels [20]. In this paper, we consider the SVM problem in a different geometry, called the *Hilbert geometry*. In 1895, David Hilbert introduced the Hilbert metric [19]. It is defined on a convex body Ω. The Hilbert metric assigns a distance between two points by considering their relative positions along the chord of Ω that passes through them. (Definitions will be provided in Section 2.) The Hilbert metric is widely used in science and engineering. It has many desirable properties, including the facts that it is invariant under projective transformations, straight lines are geodesics, and it generalizes a common model of hyperbolic geometry. An excellent resource on Hilbert geometries is the handbook by Papadopoulos and Troyanov [27]. This is the first work to study SVMs in the Hilbert geometry.[1]

Given the emerging interest in the Hilbert geometry, it is natural to ask whether the classical SVM problem can be efficiently solved in this setting. The input consists of two discrete, linearly separable point sets A and B in the interior of a convex body Ω, and the objective is to compute a separating hyperplane h that maximizes the minimum Hilbert distance from every point of $A \cup B$ to h. Changing how distances are measured results in very different criteria for the optimum separating hyperplane (see Figure 1(c)). In this paper, we explore these differences and develop an efficient algorithm for the Hilbert SVM problem for convex polygons in the plane. We also generalize our results to the closely related Funk geometry (defined in Section 2.1). The following theorems summarize our main results. For our multidimensional results, Ω is a full-dimensional convex polytope of total combinatorial complexity m. By this, we mean that Ω is the convex hull of a finite set of points in general position, and m is the total number of faces of all dimensions on its boundary.

▶ **Theorem 1.**

(i) *The Hilbert SVM problem for a set of n points in an m-sided convex polygon in $\mathbb{R}^2$ is of LP-type and can be solved in $O(n \log^2 m)$ time.*

(ii) *For any fixed $d \geq 3$, the Hilbert SVM problem for a set of n points in a convex polytope in $\mathbb{R}^d$ is of LP-type and can be solved in $O(nm^3)$ time, where m is the total combinatorial complexity of Ω.*

[1] Note that SVMs have been studied in the context of reproducing kernel Hilbert spaces [37]. Despite the similarity of the names, Hilbert spaces and Hilbert geometries are entirely different.

▶ **Theorem 2.**

(i) *The Funk SVM problem for a set of n points in an m-sided convex polygon in $\mathbb{R}^2$ is of LP-type and can be solved in $O(n \log m)$ time.*

(ii) *For any fixed $d \geq 3$, the Funk SVM problem for a set of n points in a convex polytope in $\mathbb{R}^d$ is of LP-type and can be solved in $O(nm_0)$ time, where m_0 is the number of vertices of Ω.*

Note that it is possible to test whether A and B are linearly separable through a standard reduction to linear programming. Through an analysis of the geometry of separating hyperplanes in the Funk and Hilbert geometries, we show that in $\mathbb{R}^d$, these SVM problems are of LP-type [30] of combinatorial dimension $d + 1$. By standard results, in any fixed dimension d, LP-type problems can be solved with $O(n)$ violation tests and evaluations of the objective function on point sets of constant size. (See, e.g., Sharir and Welzl [30], Seidel [29], and Clarkson [11], or the deterministic algorithm of Chazelle and Matoušek [9].) Our main technical contribution is a detailed analysis of the geometric structure of the distance between points and lines and hyperplanes in the Funk and Hilbert geometries. We use this analysis to show how to compute the violation tests and objective function evaluations that are required by algorithms for solving LP-type problems.

1.1 Applications of the Hilbert Geometry

There are a number of applications of the Hilbert geometry in science and engineering. An example involves discrete probability distributions. A discrete probability distribution $\Pi = \{p_1, \ldots, p_n\}$ can be modeled as a point in the *probability simplex*, $\{x \in \mathbb{R}^n : x_i \geq 0, \sum_i x_i = 1\}$. Clearly, this body is equivalent to a simplex in $\mathbb{R}^{n-1}$, and hence the $(n-1)$-dimensional simplicial Hilbert distance provides a natural distance measure between two probability distributions. Nielsen and Sun have demonstrated the benefits of using the Hilbert metric over other standard distance functions for probability distributions [25]. They considered clustering tasks in the probability simplex and the correlation elliptope. They observed that the Hilbert metric performed competitively when compared to other distance models in the probability simplex, such as the Fischer-Hotelling-Rao metric distance and the Kullback-Liebler divergence. They also showed that the Hilbert metric on the probability simplex satisfies the important property of being *information monotone* [25].

Beyond machine learning, the Hilbert metric has found recent uses in the context of convex approximation [2]. This is due to its close relationship with Macbeath regions [1, 35], which, up to scaling factors, are equivalent to Hilbert balls. This has been used to analyze the flag approximability of convex bodies [36] and the construction of efficient convex coverings [3]. Various results from classical computational geometry have been recreated in the Hilbert metric, including algorithms for Voronoi diagrams [6, 18, 18] and Delaunay triangulations [17].

The remainder of the paper is organized as follows. In Section 2, we present preliminary definitions and facts about the Hilbert geometry and SVMs. In Section 3, we present the proof that the Hilbert SVM problem is of LP-type of combinatorial dimension three. In Section 4, we present algorithms for computing the primitives of violation testing and computing the objective function. Finally, in Section 5, we present concluding remarks.

2 Mathematical Preliminaries

In this section, we provide a number of definitions and useful facts. Throughout, for our planar results, Ω is an m-sided convex polygon. For our multidimensional results, Ω is a full-dimensional convex polytope of total combinatorial complexity m. By this, we mean that

Ω is the convex hull of a finite set of points in general position, and m is the total number of faces of all dimensions on its boundary. Let $\partial\Omega$ denote its boundary, and let $\text{int}(\Omega)$ denote its interior. Given two points $p, q \in \Omega$, 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. Given a set $S \subset \mathbb{R}^2$, let $\text{conv}(S)$ denote its convex hull. Given two points p and q, let $\overleftrightarrow{pq}$ denote the line that passes through them, that is, their affine closure.

2.1 The Funk and Hilbert Distances

In this section, we will introduce the Funk, reverse-Funk, and Hilbert geometries over Ω. For any two points $p, q \in \text{int}(\Omega)$, if $p = q$ all the distances are defined to be zero. Otherwise, let p' and q' denote the endpoints of the chord $\overline{pq}$ ordered as $\langle p', p, q, q' \rangle$ (see Figure 2(a)). These distances are defined in terms of the ratios of distances between pairs of these points.

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

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

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

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.

These distance functions are all nonnegative and satisfy the triangle inequality [27]. The Hilbert distance defines a *metric* over $\text{int}(\Omega)$. The Funk and reverse Funk are asymmetric, and so they each define a *weak metric* over $\text{int}(\Omega)$. Observe that as either point approaches the boundary of Ω, the Hilbert distance approaches ∞. (This is true for one of the two points in Funk and reverse-Funk.) The Funk and reverse-Funk distances are invariant under invertible affine transformations. The Hilbert distance has the additional property that it is invariant under invertible projective transformations.

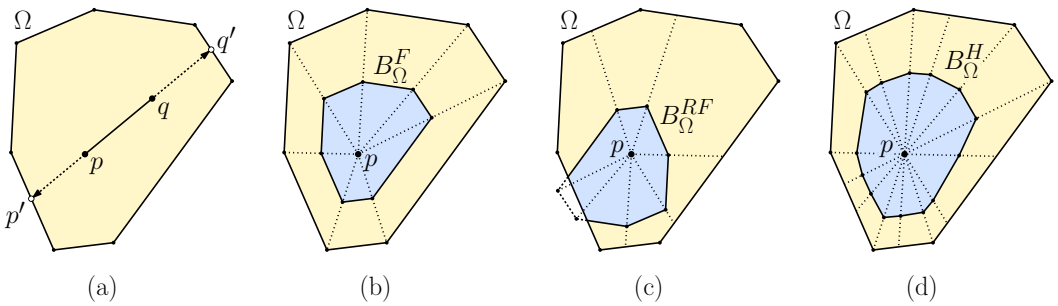


Figure 2 (a) Defining distance and balls for (b) Funk, (c) reverse Funk, and (d) Hilbert.

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

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

In particular, letting $\lambda^F(r) = 1 - e^{-r}$ and $\lambda^{rF}(r) = e^r - 1$, it is easy to verify that the Funk and reverse-Funk balls are

$$B^F_\Omega(p,r) = p + \lambda^F(r)(\Omega - p) \quad (4)$$

$$B^{rF}_\Omega(p,r) = \Omega \cap \big(p - \lambda^{rF}(r)(\Omega - p)\big)\,. \quad (5)$$

Thus, the Funk ball is a scaling of Ω by a factor of $\lambda^F(r)$ about p [26] (see Figure 2(b)), and the reverse-Funk ball is the intersection of Ω with a scaling of $-\Omega$ by a factor of $\lambda^{rF}(r)$ about p (see Figure 2(c)). Unlike the Funk ball, this scaling may generally extend outside Ω.

The Hilbert ball about a point is a bit more complicated to describe. Let us first consider the planar case. Given a point $p \in \text{int}(\Omega)$, define a *spoke* to be the chord $\overline{pv}$, for some vertex v of Ω, and define a *half-spoke* to be either of the subsegments of the spoke whose endpoint is p. The m vertices of Ω define m spokes, which induce $2m$ half-spokes joining p to the boundary of Ω (see Figure 2(d)). Nielsen and Shao showed that $B^H_\Omega(p,r)$ is a convex polygon whose vertices are the $2m$ points lying on these half-spokes at Hilbert distance r from p. Furthermore, the edges of the ball satisfy the following concurrency condition [24].

▶ **Lemma 3.** *Consider a convex polygon Ω in $\mathbb{R}^2$ and an edge e on the boundary of some Hilbert ball $B^H_\Omega(p,r)$. Let e' and e'' denote the edges of Ω that lie within the double-wedge formed by the two half-spokes that bound e (see Figure 3). Then the linear extensions of e, e', and e'' are concurrent.*

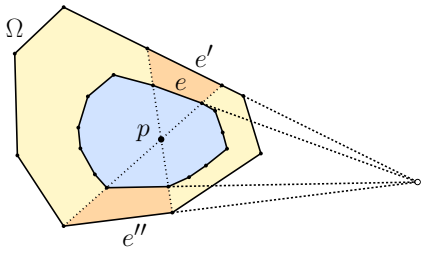


Figure 3 Concurrency of Hilbert ball edges.

This lemma generalizes to $\mathbb{R}^d$, where e, e' and e'' are facets, but the planar version suffices for our results. Since all distances will be defined with respect to Ω henceforth, we will frequently omit explicit references to Ω when discussing distances and metric balls.

The concept of spokes can be easily generalized to the case where Ω is a convex polytope in $\mathbb{R}^d$, and $p \in \text{int}(\Omega)$. In particular, for $0 \le k \le \lfloor (d-1)/2 \rfloor$, consider two faces f and f' of Ω of respective dimensions k and $d-k-1$ such that p lies in the convex hull of $f \cup f'$. It follows from straightforward conditions on dimensionality (see, e.g., [14]) that, assuming general position, for any two such faces, there is at most one line passing through p that intersects both faces. Define a *spoke* of p to be the chord of Ω contained in any such line, and again a *half-spoke* is the portion of the chord lying on one side of p. (The planar case arises when $d = 2$ and $k = 0$.)

▶ **Lemma 4.** *Consider a convex polytope Ω in $\mathbb{R}^d$ of combinatorial complexity m and a point $p \in \text{int}(\Omega)$, for any $r > 0$, the Hilbert ball $B^H_\Omega(p,r)$ is a polytope with $O(m^2)$ vertices, each of which lies on a half-spoke emanating from p. When $d = 2$, this reduces to $O(m)$.*

Proof. The planar case follows from the analysis of Nielsen and Shao [24]. For the higher-dimensional cases, we appeal to a variational argument. Consider any vertex x of $B^H_\Omega(p,r)$. Suppose to the contrary that the chord $\overline{px}$ is not a spoke. Consider the points y and y'

where this chord intersects $\partial\Omega$. It follows that there are nonempty affine neighborhoods around y and y', denoted N_y and $N_{y'}$, respectively, $N_y, N_{y'} \subset \partial\Omega$, such that lines passing through p and intersecting N_y also intersect $N_{y'}$. This implies that there is a nonempty affine neighborhood around x such that $d^H_\Omega(p, x') = r$, for all x' in this neighborhood. Hence, x is not a vertex of $B^H_\Omega(p, r)$. By the observations made above, the number of spokes passing through any point $p \in \text{int}(\Omega)$ is bounded by the square of the number of faces of Ω, that is, $O(m^2)$. ◀

2.2 On Hilbert and Funk Balls

In this section, we present a number of technical results regarding the geometry and construction of Hilbert and Funk balls for lines and hyperplanes. Due to space limitations, some details are omitted and appear in the full version of the paper.

Let us consider how to compute the reverse-Funk ball of radius r for a hyperplane h that intersects a polytope Ω. (Later, we will consider how this applies to the planar case, where h is a line.) Consider any point q that lies on the boundary of $B^{rF}(h, r)$ (see Figure 4(a)). This means that the closest point of q on h lies at Funk distance r. Letting p denote this point, we have $d^F(q, p) = r$. Consider the supporting hyperplane h' of Ω that is parallel to h and on the opposite side from q. Let q' be a vertex of Ω on h' (see Figure 4(b)). By convexity, the segment $\overline{q'q}$ intersects h at some point p within Ω. Define $\text{cp}^F(q, h)$ to be this point. By Eq. (4) (noting that q here plays the role of p in the equation), we have $\|p - q\|/\|q' - q\| = \lambda^F(r)$. It follows that the set of points q that lie on the boundary of $B^{rF}(h, r)$ define a hyperplane h'' that is parallel to h, where the relative distances between these hyperplanes is determined by $\lambda^F(r)$. By applying the same logic to the points on the opposite side of h, it follows that $B^{rF}(h, r)$ is the intersection of Ω with a slab bounded by two hyperplanes that are parallel to h.

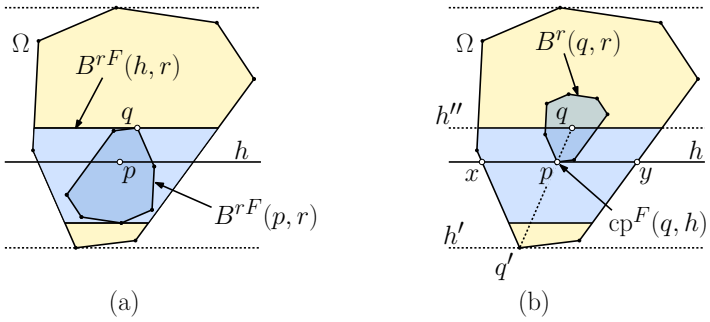


Figure 4 (a) The reverse-Funk ball around a line and (b) the closest point to q in the Funk distance.

The times to compute $\text{cp}^F(q, h)$ and $d^F(q, h)$ are dominated by the time needed to compute the supporting hyperplanes of Ω that are parallel to h. If Ω is an m-sided convex polygon in $\mathbb{R}^2$, this can be done in $O(\log m)$ time through binary search. If Ω is a convex polytope in $\mathbb{R}^d$ with m_0 vertices, this can be done in $O(m_0)$ time.

▶ **Lemma 5.** *Consider a convex polytope Ω in $\mathbb{R}^d$ and a $(d-1)$-dimensional hyperplane h that intersects the interior of Ω and $q \in \text{int}(\Omega)$. Then*

(i) *$B^{rF}(h, r)$ is the intersection of Ω with a slab that is parallel to h.*

(ii) *$\text{cp}^F(q, h)$ is the point p on $h \cap \Omega$ that minimizes the Funk distance $d^F(q, p)$.*

If Ω is an m-sided convex polygon in $\mathbb{R}^2$, $\text{cp}^F(q, h)$ and $d^F(q, h)$ can be computed in $O(\log m)$ time. If Ω is a convex polytope in $\mathbb{R}^d$ with m_0 vertices, they can be computed in $O(m_0)$ time.

Next, we consider the Hilbert ball of radius $r \geq 0$ about a line ℓ that intersects the interior of Ω, $B^H(\ell, r)$. The procedure for constructing such a ball was given by Gezalyan and Mount [18, Lemma 6], but we sketch it here for the sake of completeness.

First, extend each of the edges of Ω until the extension intersects ℓ. For each such point s, consider the triangle defined by the convex hull of the edge e that was extended to form s and the vertex v where the supporting line for Ω passing through s hits the opposite side of Ω (see Figure 5(a)). These triangles subdivide Ω. Each such triangle has two chords that cross ℓ. On each such chord, create two points at Hilbert distance r from the crossing point with ℓ. These points are in convex position, and by the projective properties of the Hilbert distance, the line segments connecting them through each triangle are part of the Hilbert ball. Together, these points constitute the vertices of the convex hull of the Hilbert ball of the line (see Figure 5(b)).

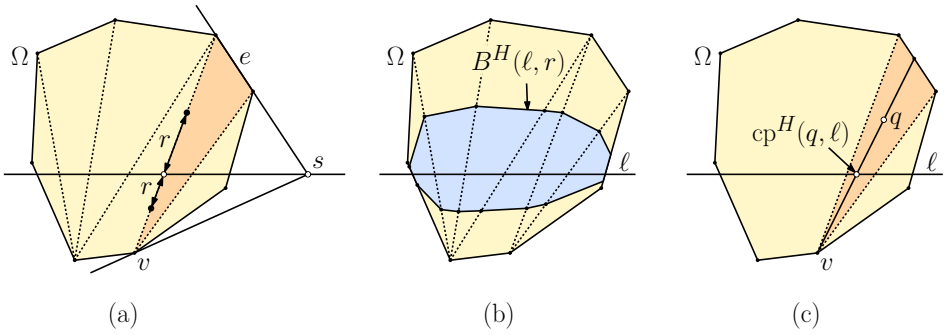


Figure 5 The Hilbert ball around a line.

Given a point $q \in \Omega$, we can compute its closest point on $\ell \cap \Omega$ as follows. Determine the triangle of the above construction that contains q (see Figure 5(c)). Consider the chord passing through q and the appropriate vertex v of the triangle. Let $\mathrm{cp}^H(q, \ell)$ denote the intersection point of this chord with ℓ. The chord that joins q and $\mathrm{cp}^H(q, \ell)$ is called the *closest-point spoke* for q with respect to ℓ.

▶ **Lemma 6.** *Consider a convex polygon Ω in $\mathbb{R}^2$ and a line ℓ that intersects the interior of Ω and $q \in \mathrm{int}(\Omega)$. Then*

(i) *$B^H(\ell, r)$ is generated by the above procedure*

(ii) *$\mathrm{cp}^H(q, \ell)$ is the point p on $\ell \cap \Omega$ that minimizes the Hilbert distance $d^H(p, q)$.*

Assuming Ω has m-sides, $\mathrm{cp}^H(q, \ell)$ and $d^H(q, \ell)$ can be computed in $O(\log^2 m)$ time.

2.3 Support Vector Machines in the Euclidean Geometry

Before discussing SVMs in the Hilbert setting, it is illustrative to explore the differences with the Euclidean case. Given two point sets $A, B \in \mathbb{R}^d$ (see Figure 6(a)) which are assumed to be linearly separable, the objective is to compute a separating hyperplane h that maximizes the *margin*, defined to be the minimum distance of either point set to h (see Figure 6(b)). The points achieving the margin are called the *support vectors*. Local minimality considerations imply that, assuming general position, the number of support vectors does not exceed $d+1$ [7].

Define a *slab* to be the region bounded between two parallel hyperplanes, the Euclidean SVM problem can be interpreted as the problem of computing the slab of maximum orthogonal width that separates A and B. The SVM hyperplane lies midway between the slab's bounding hyperplanes. The margin r is just half the slab's width (see Figure 6(b)). An alternative and

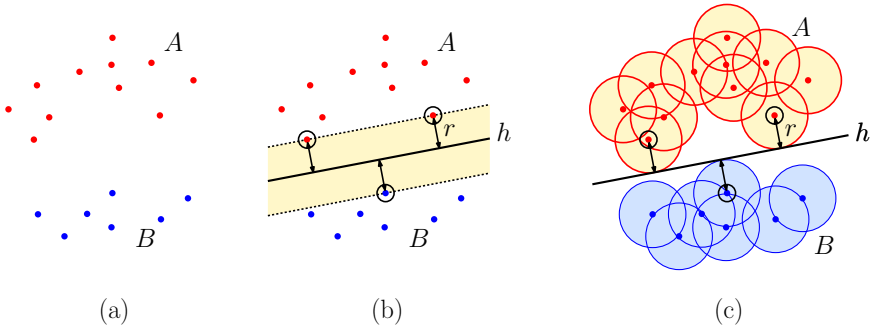


Figure 6 The Euclidean SVM problem: (a) Point sets A and B, (b) the SVM hyperplane and support vectors, (c) separating maximal balls.

equivalent interpretation is to consider the maximum radius r such that by placing balls of radius r about the points of A and B, the resulting sets of balls are linearly separable (see Figure 6(c)). By replacing Euclidean balls with metric balls, this observation can be readily generalized to any normed space.

3 SVMs in the Funk and Hilbert Geometries

In this section, we present our algorithms for solving the SVM problem in the Funk and Hilbert geometries. We consider both the planar and multidimensional cases. Recall that Ω is a full-dimensional convex polytope of total combinatorial complexity m. In both cases, we are given two linearly separable point sets $A, B \subset \text{int}(\Omega)$ (see Figure 7(a)). The objective is to compute a separating hyperplane h that maximizes the minimum Hilbert (or Funk) distance from each point of $A \cup B$ to h. Throughout, let $n = |A| + |B|$. Although we treat both n and m as asymptotic quantities, we think of m more as a large constant, since it defines a fixed geometry.

Just as in the Euclidean case (see Section 2.3), there are two equivalent formulations of the problem. In the first, we seek the separating hyperplane h of maximum radius r such that $B^H(h, r)$ contains no points of $A \cup B$ in its interior (see Figure 7(b)). The other is to determine the maximum radius r such that the union of balls $\bigcup_{a \in A} B^H(a, r)$ is linearly separable from $\bigcup_{b \in B} B^H(b, r)$ (see Figure 7(c)). As in the Euclidean SVM problem, the support vectors are associated with the points of A and B whose distance to the SVM separator equals the margin. We augment the concept of support vector to include not only the point but the half-spoke that identifies the vertex of the Hilbert or Funk ball that lies on the separating hyperplane. Note that when dealing with the Funk distance, we use the forward sense of the distance when computing balls centered at the points, and we use the reverse-Funk when computing the ball centered about the hyperplane.

3.1 Local Optimality

In $\mathbb{R}^d$, we refer to the hyperplane h that achieves the maximum Hilbert margin as the *Hilbert SVM hyperplane* for A and B, and we define the *Funk SVM hyperplane* analogously for the Funk distance. The following lemma provides a characterization of this hyperplane.

▶ **Lemma 7.** *Given a convex polytope Ω in $\mathbb{R}^d$ and two linearly separable point sets $A, B \subset \Omega$ in general position, the Hilbert (resp., Funk) SVM hyperplane h satisfies the following condition. Let r denote the margin achieved by h. There exist $d+1$ pairs $\{(p_i, s_i)\}_{i=0}^{d}$, each*

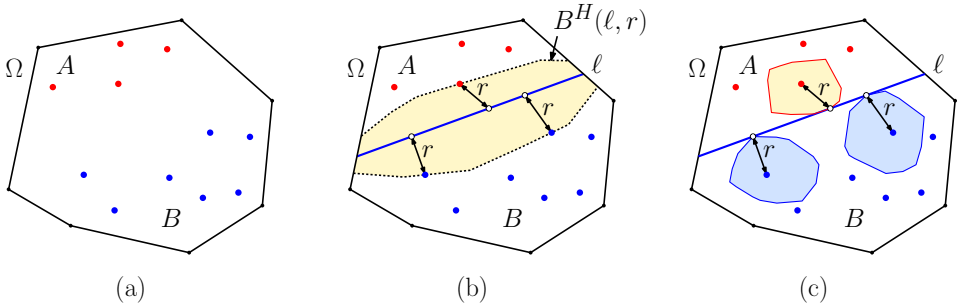


Figure 7 The Hilbert SVM problem.

consisting of a point $p_i \in A \cup B$ and a half-spoke s_i of p_i. For $0 \le i \le d$, let q_i denote the unique point at Hilbert (resp., Funk) distance r from p_i along s_i. The points $Q = \{q_i\}_{i=0}^d$ all lie on h, and furthermore, if we partition Q as $Q_a = Q \cap A$ and $Q_b = Q \cap B$, then $\text{conv}(Q_a) \cap \text{conv}(Q_b) \neq \emptyset$.

Proof. For any $r \ge 0$, let $B^H(A, r) = \bigcup_{a \in A} B^H(a, r)$ denote the set of points of Ω whose distance from any point A is at most r, and define $B^H(B, r)$ similarly for B. As mentioned above, SVM is equivalent to determining the maximum r such that $B^H(A, r)$ and $B^H(B, r)$ are linearly separable. As the value of r increases towards the optimum margin, the set of hyperplanes that separate these sets of balls becomes progressively smaller, until there is a unique separating hyperplane. Because Hilbert balls are convex polytopes, when this critical event occurs, the separating hyperplane is constrained by $d + 1$ contact points, which are generally vertices of these balls (see Figure 8(a) and (b)).

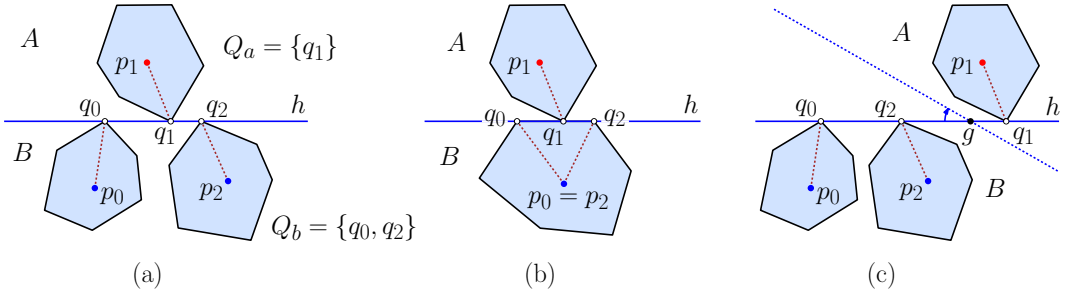


Figure 8 Criteria for local optimality for the Hilbert SVM.

By Lemma 4, these vertices lie on the half-spokes of points from A and B and each vertex is at distance r from its respective point. Let $\{(p_i, s_i\}_{i=0}^d$ denote these point-spoke combinations, and let Q_a and Q_b be as defined in the statement of the lemma. If $\text{conv}(Q_a) \cap \text{conv}(Q_b) = \emptyset$ (see Figure 8(c)), then there exists a $(d-2)$-dimensional affine subspace $g \subset h$ that separates them. By rotating h about g, we can break the contact with these metric balls, implying that it would be possible to increase the margin, a contradiction. ◀

3.2 LP-Type Problems

In this section, we show that our Hilbert and Funk SVM problems possess properties similar to that of low-dimensional linear programming, which leads to an efficient algorithmic solution. In particular, we show that these problems have a property called *LP-type* [30]. This type

of problem requires a finite set S as input and an objective function f to be maximized, which induces a total ordering over subsets of S. Generally, an LP-type problem is defined as follows.

▶ **Definition 8** (LP-type [30]). *An optimization problem over a set S and objective f is of* LP-type *if:*

- *Monotonicity: For sets $S' \subseteq S'' \subseteq S$, $f(S') \geq f(S'') \geq f(S)$.*
- *Locality: For sets $S' \subseteq S'' \subseteq S$ and every $x \in S$, if $f(S') = f(S'') = f(S' \cup \{x\})$, then $f(S') = f(S'' \cup \{x\})$.*

A *basis* of an LP-type problem is a set $S' \subseteq S$ with the property that every proper subset of S' has a larger objective-function value than S' itself. The *combinatorial dimension* of an LP-type problem is defined to be the maximum cardinality of a basis.

A natural choice for S would be $A \cup B$ and a basis would consist of $d+1$ points of this set. While this suffices for the simpler Funk distance and the Hilbert distance in the plane, we will need more information for the multidimensional Hilbert SVM problem. In particular, we will use pairs consisting of a point of $A \cup B$ and a half-spoke for this point. As seen in Lemma 7, these entities locally define the SVM hyperplane. Given A, B, and Ω, define $V = V_\Omega(A, B)$ to be the set of pairs (p, s), where $p \in A \cup B$, and s is any half-spoke emanating from p. Such a pair is called a *point-spoke*. In the multidimensional setting, the role of the set S in the definition of LP-type will be played by V.

We define the objective function for the multidimensional Hilbert SVM problem as follows. Recall that $\mathrm{Sep}(A, B)$ denotes the set of $(d-1)$-dimensional hyperplanes that separate A and B. Given a hyperplane $h \in \mathrm{Sep}(A, B)$ and a point-spoke $v = (p, s)$, define $d^H_\Omega(v, h)$ to be the Hilbert distance from p to h measured along the half-spoke s. If s does not intersect h, define the distance to be ∞. In the planar case, given any set $S' \subset A \cup B$, we define $f(S')$ to be

$$f(S') \;=\; \sup_{\ell \in \mathrm{Sep}(A,B)} \min_{p \in S'} d^H_\Omega(p, \ell).$$

In the multidimensional case, given any $h \in \mathrm{Sep}(A, B)$ and $V' \subset V$, define

$$d^H_\Omega(V', h) \;=\; \min_{v \in V'} d^H_\Omega(v, h) \qquad \text{and} \qquad f(V') \;=\; \sup_{h \in \mathrm{Sep}(A,B)} d^H_\Omega(V', h).$$

Given $p \in \mathrm{int}(\Omega)$, define $\sigma(p)$ to be the set of half-spokes of $p \in \mathrm{int}(\Omega)$. By convexity of Hilbert balls, for any $p \in A \cup B$ the half-spoke $s \in \sigma(p)$ that defines the distance $d^H_\Omega(p, h)$ to any hyperplane h is the one that minimizes the value of $d^H_\Omega((p, s), h)$. Thus, the objective function for our LP-type problem is the following.

$$f(V) \;=\; \sup_{h \in \mathrm{Sep}(A,B)} d^H_\Omega(V, h) \;=\; \sup_{h \in \mathrm{Sep}(A,B)} \min_{p \in A \cup B} \min_{s \in \sigma(p)} d^H_\Omega((p, s), h).$$

Our next result shows that Hilbert SVM satisfies these conditions.

▶ **Lemma 9.** *Given a convex polytope Ω in $\mathbb{R}^d$ and a set of linearly separable points $A, B \subset \Omega$, the Hilbert SVM problem and the Funk SVM problems are both LP-type problems of combinatorial dimension $d+1$.*

Proof. We proved in Lemma 7 that for either distance measure, any optimal solution is determined by a basis consisting of $d+1$ point-spoke pairs, which implies that the combinatorial dimension is $d+1$. Monotonicity holds trivially by the fact that adding more point-spokes to a set can never increase the minimum distance from that set to any hyperplane.

To establish locality, suppose that for some sets, $S' \subseteq S'' \subseteq S$, we have $f(S') = f(S'') = f(S' \cup \{x\})$. We assert that, the maximum-margin separating hyperplane is unique. To see why, suppose to the contrary that S' and S'' define two distinct separating hyperplanes, h' and h'', respectively, both with equal margins. These hyperplanes intersect in a $(d-2)$-affine subspace g (possibly external to Ω). Every strict convex combination h of these two hyperplanes yields a separating hyperplane that passes through g. Assuming general position (and particularly that no $d+1$ point-spokes are equidistant to g in terms of the Hilbert/Funk distance), it follows that h does not contact all $d+1$ basis point-spokes. Since $d+1$ point-spokes are needed to fix a hyperplane, there exists a perturbation of h that yields a strictly larger margin, a contradiction.

Therefore, it follows that the support vectors that form this basis are all members of S'. We may assume that $x \notin S'$, since otherwise the conclusion holds trivially. Since $x \notin S'$, the metric ball centered at x whose radius is equal to the margin does not intersect the separating hyperplane. Therefore, adding x to S'' does not change the margin value, which implies that $f(S') = f(S'' \cup \{x\})$, as desired. ◀

4 Solving the LP-Type Problems

As mentioned above, in any fixed-dimensional space, an LP-type problem over a ground set of size n can be solved in $O(n)$ time assuming free access to two primitives. The first involves evaluating the *objective function* for a given basis set. Second, the *violation test* checks whether a given point causes the objective function to decrease. In this section, we will explore the running times of these primitives.

Given point sets A and B of total size n, recall that the ground set S for the Funk SVM (in all dimensions) and the planar Hilbert SVM problems consists of $A \cup B$. Thus, $|S| = n$. For the multidimensional Hilbert SVM, the ground set consists of all point-spoke pairs V. Assuming that Ω has total combinatorial complexity m, $|V| = O(nm)$. The overall running time to solve the SVM problems will be the product of the size of the ground set and the running time of the two primitives.

Our algorithms for computing the objective function will return both the SVM hyperplane h and the margin r as a witness. This implies that the violation tests are relatively easy. In the case of the Funk distance and the two-dimensional Hilbert distance, we are given a point $p \in S$ to test. In the Funk case, by Lemma 5, we can compute the distance from p to h in $O(\log m_0)$ time in $\mathbb{R}^2$ and in $O(m_0)$ time in the multidimensional case. In the two-dimensional Hilbert case, by Lemma 6, we can compute the distance from p to h in $O(\log^2 m)$ time. In all cases, there is a violation if and only if this distance exceeds r.

The remaining case is the multidimensional Hilbert geometry. In this case, we are given a point-spoke pair (p, s) to test. In $O(m)$ time, we can determine the facets hit by the endpoints of the associated chord. We can then determine the point of intersection of this chord with the hyperplane and compute the distance from p to the hyperplane along this spoke in constant time. Again, a violation occurs if this distance exceeds r. All of these tests can be performed within the time bounds needed for the objective function evaluation, so they do not dominate the overall running time.

4.1 The Funk Objective Function

We will first discuss the objective function for the Funk SVM, since it is the easier of the two. In the planar case, Lemma 7 implies that a basis consists of three points, say $\{a, b, c\}$. From the lemma, we know that two points must come from one set and one from the other, so there

is no loss in generality in assuming that $a, c \in A$ and $b \in B$. Let ℓ be the SVM line we seek. Consider the supporting line of Ω that lies on the opposite side of ℓ as a and c. Let v denote the vertex of Ω incident to this supporting line. By Lemma 5(ii), $\mathrm{cp}^F(a, \ell)$ and $\mathrm{cp}^F(c, \ell)$ lie on the segments $\overline{av}$ and $\overline{cv}$, respectively (see Figure 9(a)). Since $d^F(a, \ell) = d^F(c, \ell)$, it follows that ℓ is parallel to $\overline{ac}$.

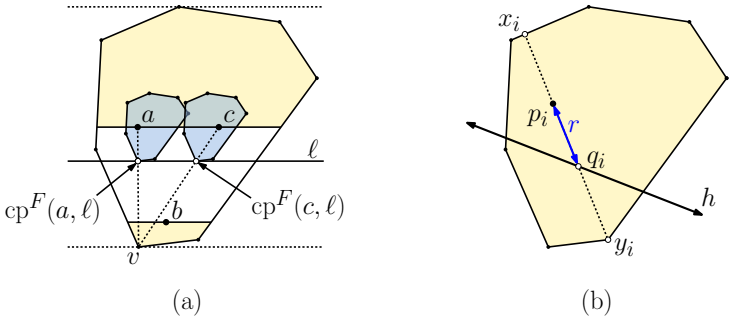


Figure 9 Computing the (a) Funk objective function and (b) Hilbert objective function.

Given the two points a and c, we can compute the vertex v in $O(\log m)$ time by binary search on the boundary of Ω. We can compute the parallel supporting line on the opposite side of Ω in $O(\log n)$ time, which is used to compute b's Funk distance. From these two lines, it is straightforward to compute the location of the parallel line ℓ that equalizes the Funk distances among all three points.

By combining this with the observations from Section 4, we can compute both the violation test and the objective function for the planar Funk SVM in time $O(\log m)$. Given that the ground set $A \cup B$ has n points, the overall running time is $O(n \log m)$, and Theorem 2(i) follows.

Next, let us consider the d-dimensional case, for any fixed $d \geq 3$. A basis consists of $d+1$ points $\{p_i\}_{i=0}^d \subseteq A \cup B$. By Lemma 5(i), the points p_i lie on two sides of a slab that is parallel to the desired SVM hyperplane. Let h denote this hyperplane. To compute h, observe that it can be represented as $\{x \in \mathbb{R}^d : \langle x, w \rangle = 1\}$, for some nonzero vector $w \in \mathbb{R}^d$. Thus, there exist reals α and β such that the points p_i above the slab satisfy $\langle p_i, w \rangle - \alpha = 0$ and the points below the slab satisfy $\langle p_i, w \rangle - \beta = 0$. This yields a system of $d+1$ homogeneous linear equations in the $d+2$ variables $\{\alpha, \beta, w_1, \ldots, w_d\}$. We can solve these equations to determine w up to a nonzero scale factor.

This fixes the orientation of h, but not its exact location. To compute its location, we first determine the two supporting hyperplanes of Ω parallel to h. This can be done in $O(m_0)$ time, where m_0 denotes the number of vertices of Ω. Given these hyperplanes, we take any basis point of A and any basis point of B, and through a straightforward calculation, in $O(1)$ time we can determine the parallel hyperplane h that equalizes the Funk distance to each.

In summary, assuming that the dimension d is fixed, both the violation test and the objective function can be computed in $O(m_0)$ time. Given that the ground set $A \cup B$ has $O(n)$ points, the overall running time is $O(nm_0)$. This establishes Theorem 2(ii).

4.2 The Multidimensional Hilbert Objective Function

Next, let us consider how to compute the objective function for the d-dimensional case, for any fixed $d \geq 3$. A basis consists of $d+1$ point-spoke pairs $\{p_i, s_i\}_{i=0}^d$. We claim that we can compute h in $O(m)$ time. To see this, observe that h can be represented as

$\{x \in \mathbb{R}^d : \langle x, w\rangle = 1\}$ for some nonzero vector $w \in \mathbb{R}^d$. For a given point-spoke pair (p_i, s_i), in $O(m)$ time we can compute the points x_i and y_i, where the extension of the half-spoke intersects $\partial\Omega$. Let us assume this is oriented so that $y_i - x_i$ is in the direction of s_i (see Figure 9(b)). For a given distance r, let $q_i = q_i(r)$ denote the point along this spoke that is at Hilbert distance r from p. To simplify the calculation, let $\widehat{r} = \exp(2r)$, and let us assume that the chord $\overline{x_i y_i}$ is affinely mapped to the interval $[0, 1]$ on the real line, which maps p_i and q_i to points in this interval (which we will continue to refer to as p_i and q_i). By Eq. (3), for $0 \le i \le d$

$$\widehat{r} \;=\; \frac{q_i}{p_i} \cdot \frac{1 - p_i}{1 - q_i} \quad \text{or equivalently} \quad \widehat{r}(p_i \cdot (1 - q_i)) \;=\; q_i \cdot (1 - p_i).$$

Together with the constraints $\widehat{r} \ge 1$ (from the fact that $r \ge 0$) and $\{0 \le q_i \le 1\}_{i=0}^{d}$ (from the requirement that the q_i's lie in the interval $[0, 1]$), this yields a system of $d + 1$ algebraic equations of degree 2 in the $d + 1$ unknowns $\{\widehat{r}, w_1, \ldots, w_d\}$. (Note that the p_i values are fixed.) Since d is a fixed constant, we can solve this system in $O(1)$ time. If this system has a feasible solution, then the objective function value is $r = \frac{1}{2} \ln \widehat{r}$, and the values of the q_i's determine the hyperplane by specifying the points where the separating hyperplane crosses each of the half-spokes.

The time to compute the objective function is dominated by the $O(m)$ time needed to compute the endpoints of each of the half-spoke extensions. By Lemma 4, each point gives rise to $O(m^2)$ half-spokes, which implies that the ground set V is of size $O(nm^2)$. Thus, the overall running time is $O(nm^3)$, and this establishes Theorem 1(ii).

4.3 The Hilbert Objective Function in the Plane

Finally, let us consider computing the objective function for the planar Hilbert SVM. From Lemma 7, a basis consists of three support vectors, which must include at least one point from each of A and B. As before, if we are presented with a subset that does not meet these conditions, we report a value of $+\infty$ for the objective function. We will focus on the more interesting case of computing the Hilbert SVM objective function for a basis consisting of three points $\{a, b, c\}$. There is no loss in generality in assuming that $a, c \in A$ and $b \in B$. As observed earlier, the value of the objective function is the maximum r such that the balls $B(a, r)$ and $B(c, r)$ are linearly separable from $B(b, r)$. The resulting separating line ℓ is equidistant to all three points (see Figure 8(a)). Note that there need not be a separating line that is equidistant to three points (see Figure 8(c)). In this section, we will present an $O(\log^2 m)$ time algorithm that either finds the desired *equidistant separating line*, or reports that no such line exists. Note that if the line does not exist, the value of the objective function reduces to a two-point case, either $\{a, b\}$ or $\{b, c\}$, whichever achieves the smaller separation distance.

4.3.1 Overview

Let us begin with an overview of the process. For the sake of illustration, we may assume that the points are given in left-to-right order (increasing x-coordinates) as $\langle a, b, c\rangle$, and b lies vertically below a point on the line segment $\overline{ac}$. This implies that there is no vertical separating line, and for any separating line a and c will be above it and b below.

By the observations made before Lemma 6, we know that for any $q \in \Omega$ and any line ℓ, the distance from q to ℓ is determined by its closest point $\mathrm{cp}^H(q, \ell)$, which lies on the closest-point spoke of q with respect to ℓ (recall Figure 5(c)). Let ℓ denote the equidistant separating line

(assuming that it exists), and let $a' = \mathrm{cp}^H(a, \ell)$, and define b' and c' analogously. It suffices to compute a', b', and c', or, failing this, to report that there is no equidistant separating line for these points.

Our approach will be based on a coordinated binary search. Let us suppose for now that an equidistant line ℓ exists, and let r^* denote the Hilbert distance of each point from this line. For any radius r, where $0 \leq r \leq d^H(a, b)/2$, let $\ell_b^a(r)$ denote the unique line that is cross-tangent to the balls $B^H(a, r)$ and $B^H(b, r)$, with a above and b below (see Figure 10(a)). Clearly, this line varies continuously as a function of r. Observe that $\{c\} = B^H(c, 0)$ lies above $\ell_b^a(0)$, and c's ball lies above the line until r reaches the value r^*, after which this condition fails. This suggests an approach based on a parametric search based on the radius r.

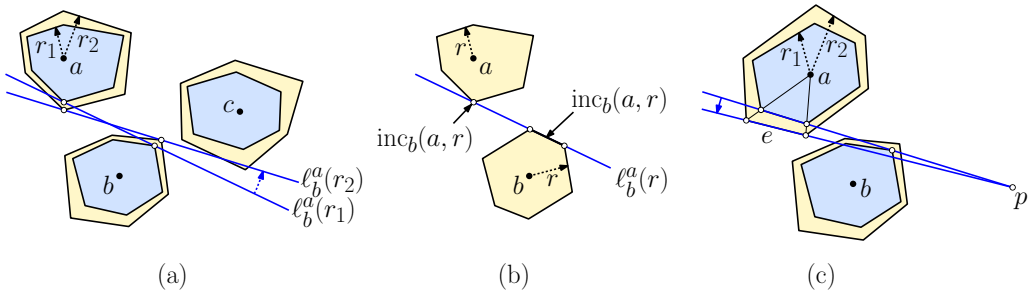


Figure 10 (a) The cross-tangents $\ell_b^a(r_1)$ and $\ell_b^a(r_2)$ for two different radii, (b) definition of inc, and (c) the proof of Lemma 10 (uniqueness).

To do this effectively, we need to establish a small set of candidate radii that contains the optimal radius r^*. We will organize the search around the contact points where the cross-tangent line intersects the boundary of the metric balls. We will show that these boundary points vary monotonically in cyclic order around the center point as the radius increases. Since we know that there is no vertical separating line, we may restrict consideration to the lower half of a's and c's metric balls (that is, the lower portion of the boundary between the leftmost and rightmost points) and the upper portion of b's metric ball.

Recall that for any radius, each vertex of a Hilbert ball lies on one of the half-spokes passing through the center point and each of the m vertices of Ω. Given a point $q \in \mathrm{int}(\Omega)$, a vertex of any Hilbert ball centered at q is uniquely identified by the half-spoke emanating from q and the ball's radius r. We refer to this half-spoke as a *virtual vertex*. The *actual vertex* is the point along this half-spoke at distance r from q. Similarly, an edge of q's metric ball, is uniquely identified by the cone with apex at q between each pair of consecutive half-spokes. This cone defines a *virtual edge* and, for a given radius r, the *actual edge* is determined by the line segment between the associated actual vertices.

For any point, there are $2m$ virtual vertices and $2m$ virtual edges. Since these entities are associated with elements on the boundary of Ω, it is possible to index these entities after $O(m)$ preprocessing time to split the spokes about the center point into half-spokes and merge them in cyclic order about the center point. Our algorithm involves a cyclic search among the virtual edges and/or the virtual vertices about each metric ball, which is performed by a coordinated binary search. The numerous technical details are presented in Section A.

5 Conclusions

In this paper, we presented an efficient algorithm for solving the SVM problem in the Hilbert and Funk geometries for a set of n points in convex polygons and polytopes of combinatorial complexity m. Our results rely on the fact that the Hilbert SVM problem has LP-type structure.

Our planar results are quite efficient, but our multidimensional results are polynomial in the total combinatorial complexity of the polytope Ω, which generally scales exponentially with the dimension. A natural question is whether the dependency can be reduced, say, to a polynomial in the combinatorial complexity of Ω, measured either as the number of facets or the number of vertices. One challenge to this is the issue that the number of vertices on the Hilbert ball grows quadratically in the number of faces of all dimensions of the polytope. Worse yet, the combinatorial structure of the Hilbert ball varies with the location of the center point.

These challenges suggest that the problem may be hard to solve in the case of general polytopes, but there are special cases of the multidimensional Hilbert geometry where the problem is still interesting. For example, Nielsen and Sun have demonstrated the usefulness of the Hilbert metric in applications of machine learning and clustering [25] for the multidimensional simplicial Hilbert geometry. Yet another direction for future work would be to consider kernel shapes other than hyperplanes.

Finally, it would be interesting to know whether our results can be extended to the reverse-Funk distance. Unlike the Funk case, the reverse-Funk ball about a hyperplane is not a slab. It seems reasonable to conjecture that the methods that were used for the more complicated Hilbert geometry could be applied to the case of the reverse-Funk SVM.

References

1 A. Abdelkader and D. M. Mount. Economical Delone sets for approximating convex bodies. In *Proc. 16th Scand. Workshop Algorithm Theory*, pages 4:1–4:12, 2018. doi:10.4230/LIPIcs.SWAT.2018.4.

2 A. Abdelkader and D. M. Mount. Convex approximation and the Hilbert geometry. In *Proc. SIAM Symp. Simplicity in Algorithms (SOSA24)*, pages 286–298, 2024. doi:10.1137/1.9781611977936.26.

3 S. Arya, G. D. da Fonseca, and D. M. Mount. Economical convex coverings and applications. In *Proc. 34th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 1834–1861, 2023. doi:10.1137/1.9781611977554.ch70.

4 A. Ben-Hur, C. S. Ong, S. Sonnenburg, B. Schölkopf, and G. Rätsch. Support vector machines and kernels for computational biology. *PLoS Comput. Biology*, 4(10):e1000173, 2008. doi:10.1371/journal.pcbi.1000173.

5 C. M. Bishop. *Pattern Recogn. and Mach. Learn.* Springer, 2016. doi:10.1007/978-0-387-45528-0.

6 M. Bumpus, C. Dai, A. H. Gezalyan, S. Munoz, R. Santhoshkumar, S. Ye, and D. M. Mount. Software and analysis for dynamic Voronoi diagrams in the Hilbert metric, 2023. arXiv:2304.02745.

7 C. 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 O. Chapelle, P. Haffner, and V. N. Vapnik. Support vector machines for histogram-based image classification. *IEEE Trans. Neural Netw.*, 10:1055–1064, 1999. doi:10.1109/72.788646.

9 B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *Journal of Algorithms*, 21:579–597, 1996. doi:10.1006/jagm.1996.0060.

10 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.

11 K. L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *J. Assoc. Comput. Mach.*, 42:488–499, 1995. doi:10.1145/201019.201036.

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

13 N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.

14 S. Das, S. R. Dev, and Sarvottamananda. A worst-case optimal algorithm to compute the Minkowski sum of convex polytopes. *Discrete Appl. Math.*, 350:44–61, 2024. doi:10.1016/j.dam.2024.02.004.

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

16 G. M. Foody and A. Mathur. A relative evaluation of multiclass image classification by support vector machines. *IEEE Trans. Geosci. Remote Sensing*, 42:1335–1343, 2004. doi:10.1109/TGRS.2004.827257.

17 A. H. Gezalyan, S. Kim, C. Lopez, D. Skora, Z. Stefankovic, and D. M. Mount. Delaunay triangulations in the Hilbert metric. In *Proc. 19th Scand. Workshop Algorithm Theory*, pages 25:1–25:17, 2024. doi:10.4230/LIPIcs.SWAT.2024.25.

18 A. H. Gezalyan and D. M. Mount. Voronoi diagrams in the Hilbert metric. In *Proc. 39th Internat. Sympos. Comput. Geom.*, pages 35:1–35:16, 2023. doi:10.4230/LIPIcs.SoCG.2023.35.

19 D. Hilbert. Ueber die gerade Linie als kürzeste Verbindung zweier Punkte. *Math. Annalen*, 46:91–96, 1895. doi:10.1007/BF02096204.

20 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.

21 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.

22 T. Joachims. *Learning to Classify Text Using Support Vector Machines*. Springer, 2002. doi:10.1007/978-1-4615-0907-3.

23 S. Krig. *Computer Vision Metrics: Survey, Taxonomy, and Analysis*. Springer nature, 2014.

24 F. Nielsen and L. Shao. On balls in a Hilbert polygonal geometry (multimedia contribution). In *Proc. 33rd Internat. Sympos. Comput. Geom.*, pages 67:1–67:4, 2017. doi:10.4230/LIPIcs.SoCG.2017.67.

25 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.

26 A. Papadopoulos and M. Troyanov. From Funk to Hilbert geometry. In *Handbook of Hilbert geometry*, volume 22 of *IRMA Lectures in Mathematics and Theoretical Physics*, pages 33–68. European Mathematical Society Publishing House, 2014. doi:10.4171/147-1/2.

27 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.

28 S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition, 2021.

29 R. Seidel. Low dimensional linear programming and convex hulls made easy. *Discrete Comput. Geom.*, 6:423–434, 1991. doi:10.1007/BF02574699.

30 M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proc. 9th Internat. Sympos. on Theoret. Aspects of Comp. Sci.*, pages 567–579, 1992. doi:10.1007/3-540-55210-3_213.

31 I. Steinwart and A. Christmann. *Support Vector Machines*. Springer, 2008. doi:10.1007/978-0-387-77242-4.

32 R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2022. doi:10.1007/978-3-030-34372-9.

33 S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66, 2001. doi:10.1162/153244302760185243.

34 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.

35 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.

36 Constantin Vernicos and Cormac Walsh. Flag-approximability of convex bodies and volume growth of Hilbert geometries, 2018. arXiv:1809.09471.

37 H. Wendland. *Scattered Data Approximation*. Cambridge University Press, 2004. doi:10.1017/CBO9780511617539.

A The Hilbert Objective Function in the Plane

In this section, we present the technical details of the evaluation of the Hilbert SVM objective function in the planar case, which were deferred from the main body of the paper.

A.1 Incidence Points and Candidate Radii

Define $\text{inc}_a(b, r)$ to be the incident virtual vertex or virtual edge of $B^H(a, r)$ with $\ell_b^a(r)$ (recall Figure 10(b)). (Assuming general position, a line will be incident to a vertex only, but we will be interested in those radii where the line is incident to an edge.) Similarly, define $\text{inc}_b(a, r)$ to be the incident vertex or edge of $B^H(b, r)$ with $\ell_b^a(r)$. Symmetrically, define $\ell_b^c(r)$ to be the unique line that is cross-tangent to the balls $B^H(c, r)$ and $B^H(b, r)$, with c lying above and b lying below, and define $\text{inc}_c(b, r)$ and $\text{inc}_b(c, r)$ analogously. Our approach is to conduct a coordinated binary search among these three sorted lists of incidence entities to determine the particular triple of entities that yield the equidistant separating line.

To limit the search to a discrete set, the candidate radius values will be chosen so that one of the lines $\ell_b^a(r)$ or $\ell_b^c(r)$ is incident to an edge of one of the three metric balls. For each virtual edge e of the metric balls centered at a, let $\text{rcand}_a(b, e)$ denote the set of radius values r such that $\text{inc}_a(b, r) = e$. Define $\text{rcand}_c(b, e)$, $\text{rcand}_b(a, e)$, and $\text{rcand}_b(c, e)$ analogously, where e comes from the appropriate metric ball (c, b, and again b, respectively). We will show below that each set is either empty or consists of a single element.

▶ **Lemma 10.** *Given the point configuration $\{a, b, c\}$ defined above and any edge e of a's Hilbert ball, if $\text{rcand}_a(b, e)$ exists, then it is unique. (The same holds for the other point combinations, $\text{rcand}_c(b, e)$, $\text{rcand}_b(a, e)$, and $\text{rcand}_b(c, e)$, where e is chosen from the appropriate ball.)*

Proof. If $\text{rcand}_a(b, r)$ exists, then by definition there is a radius r such that the cross-tangent between $B_a^H(r)$ and $B_b^H(r)$ is incident to e (see Figure 10(c)). Let r be the smallest such radius and suppose, to the contrary, that there exists a larger radius r' such that $r' \in \text{rcand}_a(b, r')$. Thus, the two cross-tangents $\ell_b^a(r)$ and $\ell_b^a(r')$ are both incident on (the virtual edge) e.

By Lemma 3, every edge of any Hilbert ball centered at a point of Ω is concurrent with the point where the linear extensions of two boundary edges of Ω meet. The point of concurrency is clearly external to Ω. Let p denote this point. Clearly, p cannot lie within the portion of either line between the contact points of the lines with either metric ball (which is internal to Ω). As the radius increases, the line rotates about p in such a way that it moves closer to b. Since the radius increases, the ball about b becomes larger, implying that $\ell_b^a(r')$ intersects $B_b^H(r)$ (see Figure 10(c)). This contradicts the fact that this line is a cross-tangent between the two balls.

Since this only uses properties of cross-tangents of Hilbert balls, the same proof applies to $\text{rcand}_c(b, e)$, $\text{rcand}_b(a, e)$, and $\text{rcand}_b(c, e)$. ◀

The second property that we will need to drive a binary search is monotonicity. As the radius increases, the virtual point of incidence moves monotonically around the boundary of the Hilbert ball (see Figure 11(a)). This would be easy if the metric balls were simple scalings of each other (as occurs, for example, with the Funk distance). In the case of Hilbert, our proof exploits the concurrency property of the edges of the metric ball (recall Lemma 3). As in the previous lemma, we will show this for $\text{inc}_a(b, r)$, but the argument holds analogously to the others.

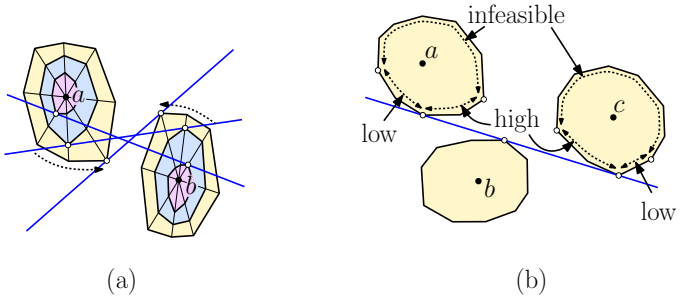


Figure 11 (a) The proof of Lemma 11 (monotonicity) and (b) boundary classification.

▶ **Lemma 11** (Monotonicity). *Given the point configuration $\{a, b, c\}$ defined above, $\text{inc}_a(b, r)$ varies monotonically with r about the boundary of a's metric ball. (The same holds for the other point combinations $\text{inc}_c(b, r)$, $\text{inc}_b(a, r)$, and $\text{inc}_b(c, r)$, where the motion is counter-clockwise for combinations involving a and clockwise for combinations involving c.)*

Proof. As the radius r increases, the cross-tangent line $\ell_b^a(r)$ changes continuously. Thus, as r increases, the object of incidence forms a sequence that alternates between the vertices and edges of the metric ball. If there is any instance where monotonicity fails, some edge e must be visited twice in this sequence. However, this would imply that there are two distinct radius values associated with this edge, which contradicts Lemma 10. ◀

A.2 Search Procedure

We now have laid the necessary foundations to describe our search. Intuitively, we will label the edges and vertices as being either "low" or "high", depending on whether a cross-tangent incident to this entity corresponds to a radius value that is smaller than or larger than the optimum value r^*, respectively. Given an edge e of a's metric ball, if $r = \text{rcand}_a(b, e)$ exists and the associated cross-tangent $\ell_b^a(r)$ passes below $B^H(c, r)$, we say that the edge is *low*. If the candidate radius exists, but the cross-tangent does not pass below $B^H(c, r)$, we say it is *high*. If the candidate radius does not exist, we say that it is *infeasible*.

Each vertex can be categorized similarly. Given a vertex v of a's metric ball, if there exists a radius value r such that $\text{inc}_a(b, r) = v$ and $\ell_b^a(r)$ passes below $B^H(c, r)$, we label this vertex as *low*. If there exists such a radius r, and $\ell_b^a(r)$ does not pass below $B^H(c, r)$, we label it as *high*. Unlike edges, a single vertex may be labeled as both low and high. Finally, if there is no such radius, we label the vertex *infeasible*. By our earlier observation that it suffices to limit consideration to the lower half of a's and c's metric balls, we can further categorize infeasible points as lying to the left or right of the low-to-high transition point.

As expected, monotonicity implies that if there is an equidistant separating line, the boundary entities of both a's and c's metric balls have a single transition point from low to high. (Assuming general position, this will take the form of a vertex that is labeled both low and high.) The following lemma summarizes the observations made in this section.

▶ **Lemma 12.** *Given the point configuration $\{a, b, c\}$ defined above, if they are not entirely infeasible, the relevant portions of the boundaries of the metric balls for a and c consist of a single contiguous low portion followed by a contiguous high portion (possibly sharing a common vertex that is both low and high).*

In light of this lemma, the first task of the search is to find the virtual vertex of a's Hilbert ball where the transition from low to high occurs, and then we do the same for c's Hilbert ball. Let us consider how this is done for a, and the algorithm for c is symmetrical. Consider any (virtual) edge e of a's Hilbert ball. We compute the radius $r = \text{rcand}_a(b, e)$ by means of a binary search along b's metric ball's boundary. Lemma 3 implies that there exists a point p (external to Ω) about which the virtual edge e rotates as the radius r changes. Let $e(r)$ denote the actual location of this edge for any given radius r (see Figure 12). Consider any virtual vertex of b's Hilbert ball. Define $v(r)$ to be the actual location of the point along this half-spoke whose distance from b is r.

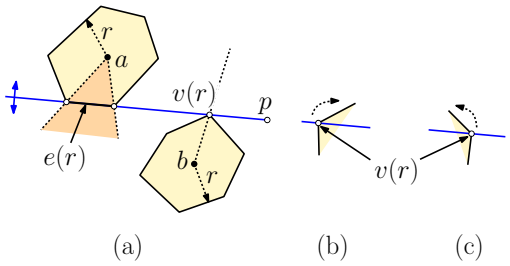


Figure 12 Searching for $\text{rcand}_a(b, e)$.

Because p is external to Ω, $v(r)$ moves continuously and monotonically along this directed spoke. Therefore, there is at most one value r where $v(r)$ lies on the linear extension of $e(r)$. If no such r exists, then this spoke is infeasible for e. Otherwise, fix r at the value where $e(r)$ and $v(r)$ are collinear. In $O(1)$ time, we can test whether this common line is a supporting line to $B^H(b, r)$ that separates the two balls (see Figure 12(a)). If so, then we have successfully found r. Otherwise, the line intersects $B^H(b, r)$ in one of two ways, either entering at $v(r)$ or exiting from $v(r)$. Depending on which case applies, monotonicity (Lemma 11) indicates whether we are "low" or "high" with respect to b's boundary (see Figure 12(b) and (c)). After $O(\log m)$ probes to b's spokes, either we succeed in finding $\text{rcand}_a(b, e)$ or we fail. In case of failure, we classify e as "infeasible". If it is infeasible, we further classify it as lying to the left or right of the transition point, based on which side of the line c lies. Since b's metric ball is bounded by $O(m)$ sides and each probe can be performed in $O(1)$ time, it follows that through binary search, we can compute the values of $r = \text{rcand}_a(b, e)$, $e(r)$, and $v(r)$ in $O(\log m)$ time.

Once we have established $r = \text{rcand}_a(b, e)$, we ask whether the ball $B^H(c, r)$ lies above or below the line through $e(r)$ and $v(r)$. This reduces to computing the closest point between c and this line, which, by the observations following Lemma 6, can be accomplished in $O(\log m)$ time. Based on this, we classify e on a's metric ball as being low or high. Since each probe of an edge of a's metric ball takes $O(\log m)$ time, in $O(\log^2 m)$ time, we can determine the transition point between low and high edges of a, whose existence is implied by Lemma 12.

In summary, in $O(\log^2 m)$ time, we can determine the low-high transition vertex for a's metric ball. By repeating the process symmetrically for c (and the cross-tangent to b), we can identify the corresponding vertex for c's metric ball.

To complete the process, we need only identify the virtual vertex associated with b's metric ball. To do this, we classify the virtual edges of b's metric ball as *left*, *right*, or *infeasible*. Let e be any virtual edge of b's metric ball. Apply the same $O(\log^2 m)$ procedure described above to determine $r_a = \text{rcand}_b(a, e)$ and $r_c = \text{rcand}_b(c, e)$. If both return "infeasible" then e is labeled as infeasible. If r_a alone returns a feasible value, we declare e to be *left* (see Figure 13). If r_c alone returns a feasible value, we declare e to be *right*. If both return a feasible value, we classify e as *left* or *right* depending on whether r_a is greater than or less than r_c, respectively. The vertex we desire is the one where the left-to-right transition occurs. The following lemma summarizes the observations made in this section.

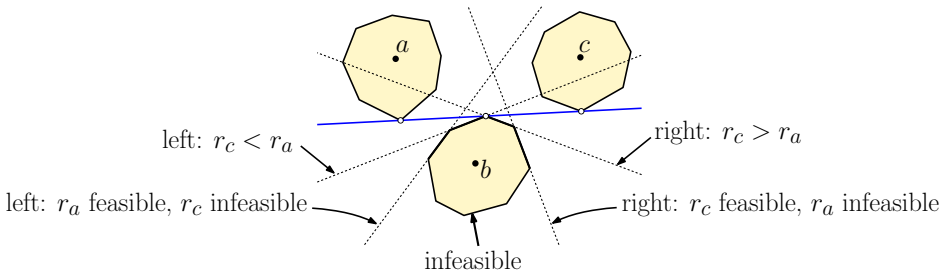


Figure 13 Searching b's metric ball.

▶ **Lemma 13.** *Given the point configuration $\{a, b, c\}$ defined above, if they are not entirely infeasible, the relevant portions of the metric ball boundary for b consist of a single contiguous left portion followed by a contiguous right portion.*

Based on this lemma, we proceed to compute the vertex that separates the left and right portions in $O(\log^2 m)$ time, analogous to how we did this for the a and c cases. When this is done, we have computed the three virtual vertices for a, b, and c that define the closest points for the Hilbert SVM line. Thus, we have eliminated the combinatorial element of the problem. All that remains is to solve the remaining numerical problem. The solution is quite technical, but the computation involves only a constant number of entities (six, to be precise), and so the running time is $O(1)$. The details are presented in the full version of the paper.

In summary, we have shown that we can compute the violation test in time $O(\log^2 m)$ and we can evaluate the objective function in time $O(\log^2 m)$. Given that both the violation test and the evaluation of the objective function can be performed in $O(\log^2 m)$ time, this completes the proof of Theorem 1.

Evolving Distributions Under Local Motion

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

David M. Mount
Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA

Abstract

Geometric data sets that arise in modern applications are often very large and change dynamically over time. A popular framework for dealing with such data sets is the evolving data framework, where a discrete structure continuously varies over time due to the unseen actions of an evolver, which makes small changes to the data. An algorithm probes the current state through an oracle, and the objective is to maintain a hypothesis of the data set's current state that is close to its actual state at all times. In this paper, we apply this framework to maintaining a set of n point objects in motion in d-dimensional Euclidean space. To model the uncertainty in the object locations, both the ground truth and hypothesis are based on spatial probability distributions, and the distance between them is measured by the Kullback-Leibler divergence (relative entropy). We introduce a simple and intuitive motion model in which, with each time step, the distance that any object can move is a fraction of the distance to its nearest neighbor. We present an algorithm that, in steady state, guarantees a distance of $O(n)$ between the true and hypothesized placements. We also show that for any algorithm in this model, there is an evolver that can generate a distance of $\Omega(n)$, implying that our algorithm is asymptotically optimal.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Evolving data, tracking, imprecise points, local-motion model, online algorithms

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.4

Related Version *Full Version*: `https://arxiv.org/pdf/2409.11779`

Acknowledgements The authors would like to thank Michael Goodrich for suggesting the question of how to model multidimensional evolving point sets.

1 Introduction

Many modern computational applications are characterized by two qualities: data sets are massive and vary over time. A fundamental question is how to maintain some combinatorial structure that is a function of such a data set. The combination of size and dynamics makes maintaining such a structure challenging. Given the large data sizes, single-shot algorithms may be too slow, and common dynamic algorithms, which support explicit requests for insertions and deletions, may not be applicable because structural changes are unseen by the algorithm.

Anagnostopoulos, Kumar, Mahdian, Upfal, and Vandin introduced a model for handling such data sets, called the *evolving data framework* [2]. In this framework, the structure varies repeatedly over time through the unseen actions of an *evolver*, which makes small, stochastic changes to the data set. The algorithm can probe the current state locally using an *oracle*. With the aid of this oracle, the algorithm maintains a *hypothesis* of the current state of the structure that is as close as possible to its actual state at all times. The similarity between the hypothesis and the current state is measured through some *distance function*. The algorithm's objective is to achieve, in the steady state, a small distance between the hypothesis and the actual state. This framework has been applied to a variety of problems [1, 6, 17, 21, 36].

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

Consider, for example, sorting. The data consists of a set of objects over some total order. The evolver repeatedly selects a random pair of adjacent objects and swaps them. The oracle is given two objects and returns their relative order. The objective of the algorithm is to maintain an order that is as close to the current state as possible, where the distance is measured in terms of the Kendall tau distance, that is, the number of pairwise order inversions [22]. It has been shown that a Kendall tau distance of $O(n)$ is achievable, and this is optimal [5, 6]. It is noteworthy that the optimal algorithm in the evolving framework is based on a simple quadratic-time sequential algorithm, and not $O(n \log n)$-time algorithms, as one might expect. The sorting problem has been generalized to tracking labels on a tree in [1], laying the foundations for a geometric framework for evolving data.

This paper focuses on the question of how to maintain a set of points whose positions evolve continuously in real d-dimensional space, $\mathbb{R}^d$. In motion tracking applications, object movement is recorded through various technologies, including GPS-enabled mobile devices [31], RFID tags [28], and camera-based sensing [35]. Examples include the movement and migration of animals on land and in oceans, traffic and transport, defense and surveillance, and analysis of human behavior (see, e.g., [13, 24]).

Imprecision and uncertainty are unavoidable problems that arise in evolving systems. Due to sensor latency, the exact location of any object cannot be known with certainty. In the best case, any algorithm can maintain only an approximation to the current state. In order to bound the degree of uncertainty, it is necessary to impose restrictions on object motion. In traditional applications of the evolving data framework, the evolver acts randomly. This is not a reasonable assumption in practice, where moving objects are subject to physical laws or may have a sense of agency [15, 32, 34]. Much work has focused on realistic models of motion, but these can be difficult to analyze theoretically. A widely used model assumes that objects have a maximum speed limit, and as time passes, an object can be inferred to lie within a ball whose radius grows linearly over time based on this speed limit [9, 12, 20]. However, in practice, the speed that any object can move depends on the congestion within its local environment.

In multidimensional space, there is no intrinsic total order, and it is less clear what it means to accurately track imprecise moving objects. Given the inherent uncertainty, we model the location of each object in terms of a spatial probability distribution. The distance between the actual state of the system and our hypothesis is naturally defined as the relative entropy (Kullback-Leibler divergence) between these two distributions. The *Kullback-Leibler* (KL) *divergence* is a fundamental measure in statistics that represents the distance between two probability distributions [23]. It has numerous applications in statistical inference, coding theory, machine learning, among others [10]. In our case, it serves two intuitive purposes. First, it measures how different the truth is from our hypothesis. Second, it can be used to quantify the additional information required to encode the actual location of each object, based on our hypothesis [10].

For the evolution of our data, we adopt a locality-sensitive stochastic motion model. We assume that with each time step, the motion of an object is constrained by its immediate environment, which we call the *local-motion model*. In this model, the distance that any object can move in a single time step is a fixed fraction of the distance to its nearest neighbor. The support of this object's probability distribution is a region of size proportional to the nearest-neighbor distance. This model has the advantage that it does not impose arbitrary speed limits on the movement of objects, it is invariant under transformations that preserve relative distances (e.g., translation, rotation, and uniform scaling), and it satisfies the observed phenomenon that objects in dense environments have less personal space [16, 19], and exhibit slower movement than those in sparse environments [4, 29].

To control the communication complexity in determining object locations, we do not require that the oracle returns the exact object positions. Instead, we assume access to an *oracle* that is given a Euclidean ball and the index i of an object. It returns a pair of bits indicating whether the i-th object lies within this ball and (if so) whether there is any other object of the set within this ball. Since every object's location is subject to uncertainty, the same query on the oracle might result in different outcomes at different times, and our algorithm is robust to such variations.

In our framework, the evolver and the algorithm operate asynchronously and in parallel. With each time step, the evolver selects an arbitrary object of the set and moves it in accordance with the local-motion model. (This need not be random and may even be adversarial.) This information is hidden from our algorithm. The algorithm selects an object and invokes the oracle on this object. Based on the oracle's response, the algorithm updates the current hypothesized distribution for this point. Thus, the evolver and algorithm are involved in a pursuit game, with the evolver incrementally changing object distributions (possibly in an adversarial manner) and the algorithm updating its hypothesized distributions. The algorithm's goal is to minimize the relative entropy between these distributions at all times. Our computational model and a formal statement of our results are presented in Section 2.

2 Problem Formulation and Results

Objects

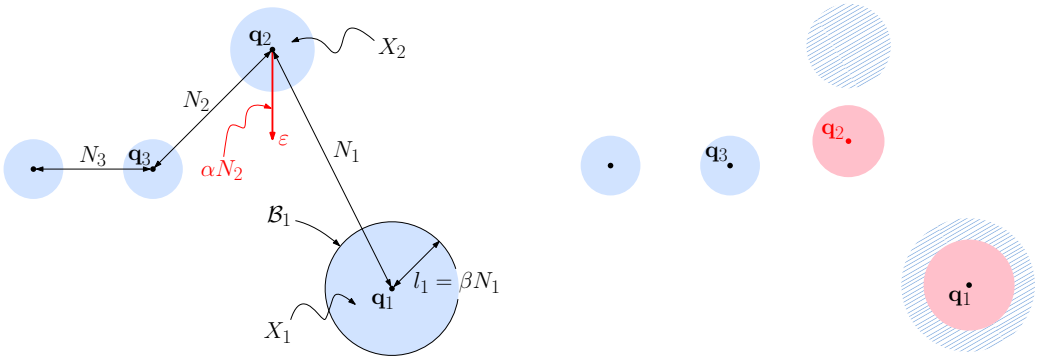
The objects that populate our system can be thought of as imprecise points [8,14,25] that are drawn independently from probability distributions that depend on the proximity to other objects. More formally, at any fixed time step, let $Q = \{\mathbf{q}_1, \ldots, \mathbf{q}_n\} \subset \mathbb{R}^d$ denote a point set of size n in d-dimensional Euclidean space, and let $[n]$ denote the index set $\{1, \ldots, n\}$. For each $\mathbf{q}_i \in Q$, let N_i denote the distance to its nearest neighbor in $Q \setminus \{\mathbf{q}_i\}$. Given $\mathbf{x} \in \mathbb{R}^d$ and nonnegative real r, let $\mathcal{B}(\mathbf{x}, r)$ denote the closed Euclidean ball of radius r centered at $\mathbf{x}$, and let $\mathrm{U}(\mathcal{B}(\mathbf{x}, r))$ denote the uniform probability distribution over this ball. Given a real β, where $0 < \beta < 1$, define the *β-local feature region* of $\mathbf{q}_i$ to be $\mathcal{B}(\mathbf{q}_i, \beta N_i)$. Let $P_i = P_i(\beta)$ denote the uniform[1] probability distribution, $\mathrm{U}\left(\mathcal{B}(\mathbf{q}_i, \beta N_i)\right)$, and let $\mathcal{P} = \{P_1, \ldots, P_n\}$ denote this set of distributions. We refer to $\mathcal{P}$ as the *truth* or *ground truth*, as it represents the true state of the system, subject to the given imprecision. Together, Q and β define a set n independent random variables $\{X_1, \ldots, X_n\}$, with X_i distributed as P_i (see Figure 1a).

The Local-Motion Model

As mentioned in the introduction, there are many ways to model the realistic motion of a collection of objects in an environment. A natural requirement is that each object's motion is affected by the presence of nearby objects. Although there are many ways to incorporate this information (see, e.g., [29]), we have chosen a very simple model, where velocities are influenced by the distance to the nearest neighbor.

We think of the objects of our system as moving continuously over time, and our algorithm queries their state at regular discrete *time steps*. From the algorithm's perspective, objects move, or "evolve," over time in the following manner. Given a parameter α, where $0 < \alpha < 1$, at each time step, an entity called *evolver* selects an object $i \in [n]$ and moves $\mathbf{q}_i$ by a distance of at most αN_i in any direction of its choosing (see Figure 1). While the value of α is known,

[1] Our choice of using a uniform probability distribution is not critical to our approach. In the full version of the paper we show that our results apply to a much broader class of distributions.



(a) Illustration of the objects, and the notations used. The evolver's action ε is shown in red.

(b) Objects that were affected by ε are shown in pink. Tiled regions show the previous state.

Figure 1 The model and an action by the evolver. Shaded regions represent objects.

the action of the evolver, including the choice of the object and the movement, is hidden from the algorithm. Throughout, we assume that the evolver is a *strong adversary*, which means that it has access to our algorithm and the input set [7].

For a nonnegative integer time step t, let $Q^{(t)}$ and $\mathcal{P}^{(t)}$ denote the underlying centers and distributions, respectively, at this time. To simplify notation, we omit the superscript when talking about the current time. We assume that there exists a *bounding region* in the form of a Euclidean ball $\mathcal{B}_0$ centered at the origin. At all times, the points $Q^{(t)}$ are restricted to lie within $\mathcal{B}_0$. The algorithm has knowledge of this ball. Define the system's *initial aspect ratio* to be $\Lambda_0 = (\text{radius}(\mathcal{B}_0))/(\min_{i\in[n]} N_i^{(0)})$. Given any positive constant c, let $c\mathcal{B}_0$ denote a factor-c expansion of this ball about the origin.

Oracle

Consider the state of the system at any fixed time t. Knowledge about the current state of the system is provided by an entity $\mathcal{O}$, called the *oracle*. It is given a Euclidean ball $\mathcal{B}(\mathbf{x}, r)$ and an object index $i \in [n]$. Recall that for each $i \in [n]$, X_i denotes a random variable distributed as P_i. The oracle is a function $\mathcal{O} : [n] \times \mathbb{R}^d \times \mathbb{R}^+ \to \{Y, N\} \times \{+, -\}$, where $\mathcal{O}(i, \mathbf{x}, r)$ returns:

- "Y" or "N" depending on whether $X_i \in \mathcal{B}(\mathbf{x}, r)$ and
- "$+$" or "$-$" depending on whether there exists $j \neq i$, such that $X_j \in \mathcal{B}(\mathbf{x}, r)$

The first element of the pair is used to estimate the location of the object i, and the second is used to estimate the distance to its nearest neighbor. Because X_i and X_j are random variables, so is $\mathcal{O}(i, \mathbf{x}, r)$. Consistent with previous applications of the evolving data framework (see, e.g., [3]), we intentionally made the oracle as weak as possible, implying that our algorithm can be used with stronger oracles.

Hypotheses and Distance

The algorithm maintains a *hypothesis* of the current object locations, which is defined to be a set $\mathcal{H} = \{H_1, \ldots, H_n\}$ of spatial probability distributions in $\mathbb{R}^d$. The *distance* between the truth $\mathcal{P}$ and the current hypothesis $\mathcal{H}$, denoted $\mathcal{D}(\mathcal{P}, \mathcal{H})$, is defined as the sum of n Kullback-Leibler divergences from the hypothesized distributions to the true ones. For $i \in [n]$, let

$$D_i \;=\; D_{\text{KL}}(P_i \parallel H_i) \;=\; \int_{\mathbf{x}\in\mathcal{B}_i} P_i(\mathbf{x}) \log \frac{P_i(\mathbf{x})}{H_i(\mathbf{x})} \mu(d\mathbf{x}),$$

where $\mu(\cdot)$ denotes the measure over $\mathcal{B}_i = \mathcal{B}(\mathbf{q}_i, \beta N_i)$ and define

$$\mathcal{D}(\mathcal{P}, \mathcal{H}) \;=\; \sum_{i=1}^{n} D_i \;=\; \sum_{i=1}^{n} D_{\mathrm{KL}}(P_i \parallel H_i). \tag{1}$$

As a baseline for comparisons, we introduce a *naïve hypothesis*, denoted $\mathcal{H}^*$, which assumes no information about the locations of the objects, other than the fact that they lie within the bounding region. Recall that $\mathcal{B}_0$ denotes this bounding region and $3\mathcal{B}_0$ denotes a factor-3 expansion about its center. Since all the points of Q lie within $\mathcal{B}_0$ and $0 < \beta < 1$, $3\mathcal{B}_0$ is guaranteed to contain all the β-local feature regions. For all $i \in [n]$, let H_i^* be the uniform distribution over $3\mathcal{B}_0$, and let $\mathcal{H}^* = \{H_1^*, \ldots, H_n^*\}$. Regardless of the initial truth, the initial distance of the ith object satisfies the following bound.

$$\begin{aligned} D_i^* \;=\; \int_{\mathbf{x}\in\mathcal{B}_i} P_i(\mathbf{x}) \log \frac{P_i(\mathbf{x})}{H_i^*(\mathbf{x})} \mu(d\mathbf{x}) \;&=\; \int_{\mathbf{x}\in\mathcal{B}_i} P_i(\mathbf{x}) \log \frac{1/(\beta N_i)^d}{1/(3\cdot \mathrm{radius}(\mathcal{B}_0))^d} \mu(d\mathbf{x}) \\ &=\; \log \left(\frac{3\cdot \mathrm{radius}(\mathcal{B}_0)}{\beta N_i} \right)^d \int_{\mathbf{x}\in\mathcal{B}_i} P_i(\mathbf{x}) \mu(d\mathbf{x}) \\ &\leq\; d \left(\log \Lambda_0 + \log \frac{3}{\beta} \right). \end{aligned}$$

Let $\mathcal{D} = \sum_{i\in[n]} D_i^*$. Under our assumption that d and β are constants, we have $\mathcal{D}^* \in \Theta(n \log \Lambda_0)$. The initial aspect ratio, Λ_0, depends on the initial configuration of the points of Q, and can be arbitrarily high. In the best case, when the points are uniformly distributed in $\mathcal{B}_0$, we have $\Lambda_0 = \Omega(n^{1/d})$ and hence $\mathcal{D} \in \Omega(n \log n)$. The objective of our algorithm is to maintain a significantly smaller bound on this distance.

The combination of evolver, oracle, and distance function constitute a model of evolving motion, which we henceforth call the (α, β)-*local-motion model*.

Class of Algorithms

We assume that the algorithm that maintains the hypothesis runs in discrete steps over time. With each step, it may query the oracle a constant number of times, perform a constant amount of work, and then update the current set of hypotheses. The number of oracle queries is independent of n but can depend on dimension d, local feature scale factor β, and motion factor α. In our case, this work takes the form of updating the hypothesis for the object that was queried. In the purest form of the evolving data framework, the evolver and algorithm alternate [2]. Instead, similar to the generalized framework proposed in [1], we assume that there is a fixed *speed-up factor*, denoted σ. When amortized over the entire run, the ratio of the number of steps taken by the algorithm and the evolver does not exceed σ.

Objective and Results

The objective of the algorithm is as follows. Given any starting ground-truth configuration, after an initial "burn-in" period, the algorithm guarantees that the hypothesis is within a bounded distance of the truth subject to model assumptions and the given speed-up factor. Our main result is that there exists an algorithm with constant speed-up factor σ that maintains a distance of $O(n)$ in steady state.

▶ **Theorem 1.** *Consider a set of n evolving objects in $\mathbb{R}^d$ under the (α, β)-local-motion model, for constants α and β, where $0 < \alpha, \beta < \frac{1}{3}$. There exists an algorithm of constant speed-up σ, and burn-in time $t_0 \in O(n \log \Lambda_0 \log\log \Lambda_0)$ such that for all $t \geq t_0$, this algorithm maintains a distance of $O(n)$.*

The algorithm and its analysis will be proved in Section 4. Given that no algorithm can guarantee an exact match between hypothesis and truth, it is natural to wonder how close this is to optimal. In Section 3, we will show that it is asymptotically optimal by showing that for any algorithm and any constant speed-up factor, there exists an evolver that can force a distance of $\Omega(n)$ in steady state.

Why the KL Divergence?

The principal challenge in generalizing the evolving framework from simple 1-dimensional applications like sorting to a multidimensional setting is the lack of an obvious distance measure that captures how close the hypothesis is to the current state. Our approach is motivated by an information-theoretic perspective. The KL divergence $D_i = D_{\mathrm{KL}}(P_i \parallel H_i)$ serves as a measure of how different the actual object distribution P_i is from our hypothesis H_i. (Note that the KL divergence is asymmetric, which is to be expected, given the asymmetric roles of the truth and our hypothesis.)

As an example of this information-theoretic approach, consider the following application in coding theory and space quantization [10]. The objective is to communicate the location of an imprecise point X_i with any arbitrary resolution δ. From Shannon's source coding theorem [26], the theoretical lower bound for the expected number of bits required for communication is given by the *Shannon entropy* [30]

$$b_{P_i} = \sum_{C_\delta \in B_i} P_i(C_\delta) \log \frac{1}{P_i(C_\delta)},$$

where C_δ is a cell (a d-dimensional box) of size δ in $\mathbb{R}^d$, and $P(C)$, by a slight abuse of notation, is the probability associated with a cell C using the underlying probability distribution function P. This is roughly achieved by a Huffman coding [18] of the space around q_i, which intuitively assigns a number of bits proportional to the log of the inverse of the probability measure associated with an event, which in our case is X_i lying within a cell C_δ of size δ.

Since the P_i's are not available to us, we use a similar strategy of encoding the space using H_i. Therefore, in expectation, we use roughly

$$b_{H_i} = \sum_{C_\delta \in B_i} P_i(C_\delta) \log \frac{1}{H_i(C_\delta)}$$

bits. When δ is arbitrarily small, the difference $(b_{H_i} - b_{P_i}) \sim D_i$. Therefore, $\mathcal{D}(\mathcal{P}, \mathcal{H}) = \sum_{i \in [n]} D_i$ roughly represents the expected number of additional bits needed to represent the location of X_i's individually using just H_i's, compared to the absolute theoretical limit.

Paper Organization

The remainder of the paper is organized as follows. In the next section, we present Theorem 2, which provides a lower bound on the distance achievable by any algorithm in our model. In Section 4, we present the algorithm and analyze its steady-state performance. In Section 5, we discuss possible relaxations to the assumptions made in our model.

3 Lower Bound

In this section, we show that maintaining $\mathcal{D}(\mathcal{P}, \mathcal{H}) \in O(n)$ is the best we can hope to achieve. This is established in the following theorem. Due to space limitations, the proof has been deferred to Section A.1.

▶ **Theorem 2.** *For any algorithm $\mathcal{A}$, there exists a starting configuration $Q^{(0)}$ and an evolver (with knowledge of $\mathcal{A}$) in the local-motion model such that, for any positive integer t_0, there exists $t \geq t_0$, such that $\mathcal{D}^{(t)} = \mathcal{D}(\mathcal{P}^{(t)}, \mathcal{H}^{(t)}) \in \Omega(n)$.*

The proof follows a similar structure to other lower-bound proofs in the evolving data framework [1, 3, 6]. Intuitively, over a period of time of length cn, for $c < 1$, the algorithm can inspect the locations of only a constant fraction of points. During this time, the evolver can move a sufficient number of uninspected points so that the overall distance increases by $\Omega(n)$.

4 Algorithm

In this section, we present our algorithm and analyze its performance. We begin by defining our hypothesis. For every index i, we define two parameters: $h_i \in \mathbb{R}$, and $\mathbf{k}_i = (k_{i,1}, \ldots k_{i,d}) \in \mathbb{R}^d$. For $\mathbf{x} = (x_1, \ldots x_d) \in \mathbb{R}^d$ we set the hypothesis probability density for the ith point to be the d-fold product of independent Cauchy distributions, one per coordinate, with center at $\mathbf{k}_i$ and scaling parameter h_i [33]. That is,

$$H_i(\mathbf{x}) \;=\; f_{h_i,\mathbf{k}_i}(\mathbf{x}) \;=\; \frac{1}{\pi^d h_i^d} \prod_{j=1}^{d} \left(1 + \frac{(x_j - k_{i,j})^2}{h_i^2}\right)^{-1} \qquad \text{(2: Hypothesis Def.)}$$

(Note that this differs from the standard multivariate Cauchy [33].) If we let $f_{1,\mathbf{0}}(\mathbf{x})$ denote the probability density function for the standard d-fold product of Cauchy distributions (centered at the origin unit scale), we can express this equivalently as

$$f_{h_i,\mathbf{k}_i}(\mathbf{x}) \;=\; \frac{1}{h_i^d} f_{1,\mathbf{0}}\left(\frac{\mathbf{x} - \mathbf{k}_i}{h_i}\right), \quad \text{where} \quad f_{1,\mathbf{0}}(\mathbf{x}) \;=\; \frac{1}{\pi^d} \prod_{j=1}^{d} \left(1 + x_j^2\right)^{-1}.$$

It will be convenient to define the *hypothesis ball* $\mathcal{B}_i^H = \mathcal{B}(\mathbf{k}_i, h_i)$, which we use to illustrate H_i. Note that, unlike our truth probabilities P_i, our hypothesis distributions have unbounded support. Our algorithm modifies the parameters $\mathbf{k}_i$ and h_i in response to information received from the oracle.

4.1 Potential Function

Due to the subtleties of tracking the Kullback-Leibler divergence, we introduce a potential function Φ to aid in the analysis. For each object index $i \in [n]$, we define an individual potential function Φ_i, which bounds the distance D_i to within a constant. The total potential Φ will be the sum of these functions.

Let $s_i = \|\mathbf{q}_i - \mathbf{k}_i\|$ denote the Euclidean distance between the center of the actual distribution and the center of the hypothesis ball. Let $l_i = \beta N_i$ denote the radius of the local feature of $\mathbf{q}_i$. Recall that h_i is the radius of the hypothesis ball (see Figure 2). We define the individual and system *potentials* to be

$$\Phi_i \;=\; \Phi(P_i, H_i) \;=\; \log\left(\frac{\max(s_i, l_i, h_i)}{\sqrt{l_i h_i}}\right) \quad \text{and} \quad \Phi \;=\; \Phi(\mathcal{P}, \mathcal{H}) \;=\; \sum_{i \in [n]} \Phi_i \qquad (3)$$

The following lemma shows that, up to additive and scalar constants that depend on the dimension, distances can be bounded above by this potential.

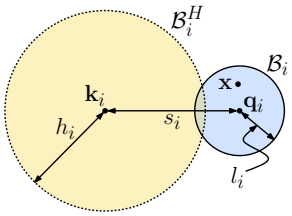


Figure 2 The definition of the individual potential Φ_i.

▶ **Lemma 3.** *There exists a constant c_d (depending on dimension) such that for any $i \in [n]$, $D_i \le c_d(1+\Phi_i)$, and hence $\mathcal{D} \le c_d(n+\Phi)$.*

Proof. Recall that $P_i(\mathbf{x})$ is the probability density function of a uniform distribution over a ball of radius l_i. Thus, for $\mathbf{x} \in \mathcal{B}_i$, $P_i(\mathbf{x}) = \omega_d / l_i^d$, where ω_d denotes the volume of the unit Euclidean ball in $\mathbb{R}^d$, and $\mathcal{B}_i$. Therefore,

$$\begin{aligned} D_i &= \int_{\mathcal{B}_i} P_i(\mathbf{x}) \log \frac{P_i(\mathbf{x})}{H_i(\mathbf{x})} \mu(d\mathbf{x}) \\ &= \int_{\mathcal{B}_i} P_i(\mathbf{x}) \log \frac{\omega_d / l_i^d}{(\pi^d h_i^d)^{-1} \prod_{j=1}^d \left(1 + \frac{(x_j - k_{i,j})^2}{h_i^2}\right)^{-1}} \mu(d\mathbf{x}) \\ &= \int_{\mathcal{B}_i} P_i(\mathbf{x}) \log \left(\omega_d \pi^d\right) \mu(d\mathbf{x}) \;+\; \int_{\mathcal{B}_i} P_i(\mathbf{x}) \log \frac{h_i^d \prod_{j=1}^d \left(1 + \frac{(x_j - k_{i,j})^2}{h_i^2}\right)}{l_i^d} \mu(d\mathbf{x}). \end{aligned} \tag{4}$$

For $\mathbf{x} \in \mathcal{B}_i$ and $j \in [d]$, by the triangle inequality, we have

$$|x_j - k_{i,j}| \;\le\; \|\mathbf{x} - \mathbf{k}_i\| \;\le\; \|\mathbf{x} - \mathbf{q}_i\| + \|\mathbf{q}_i - \mathbf{k}_i\| \;\le\; l_i + s_i.$$

Therefore,

$$\begin{aligned} D_i &\le \log\left(\omega_d \pi^d\right) \;+\; \int_{\mathbf{x} \in \mathcal{B}_i} P_i(\mathbf{x}) \log \left[\frac{h_i^d}{l_i^d} \prod_{j=1}^d \left(1 + \frac{(s_i + l_i)^2}{h_i^2}\right)\right] \mu(d\mathbf{x}) \\ &\le \log\left(\omega_d \pi^d\right) \;+\; \log \left[\frac{h_i^d}{l_i^d} \left(1 + \frac{(s_i + l_i)^2}{h_i^2}\right)^d\right] \int_{\mathbf{x} \in \mathcal{B}_i} P_i(\mathbf{x}) \mu(d\mathbf{x}) \\ &= \log\left(\omega_d \, \pi^d\right) \;+\; d \log \left(\frac{h_i^2 + (s_i + l_i)^2}{l_i \; h_i}\right) \;\le\; c_d \left(1 + \log \frac{\max(s_i, l_i, h_i)}{\sqrt{l_i h_i}}\right), \end{aligned}$$

for a suitably chosen c_d that depends only on the dimension. ◀

▶ Remark (Potential function and approximating pairwise distances). The potential function Φ_i we define here is symmetric. It is remarkable that a symmetric function bounds an asymmetric function D_i under the choice of Cauchy distributions as hypotheses.

To demonstrate the value of our potential function, suppose that for $\forall i, \Phi_i < \log c$, for some constant c. This implies that $h_i < c\sqrt{l_i h_i}$, which further implies $h_i < c' l_i$ for some constant c'. Similarly, $s_i < c\sqrt{l_i h_i} < c'' l_i$ for some constant c''. By the triangle inequality, the distance between the centers of two hypothesis balls satisfies

$$\|\mathbf{k}_i - \mathbf{k}_j\| \;\le\; s_i + s_j + \|\mathbf{q}_i - \mathbf{q}_j\| \;\le\; c'' l_i + c'' l_j + \|\mathbf{q}_i - \mathbf{q}_j\|.$$

Since $l_i = N_i/\beta \le \|\mathbf{q}_i - \mathbf{q}_j\|/\beta$, we have $\|\mathbf{k}_i - \mathbf{k}_j\| \le c^* \|\mathbf{q}_i - \mathbf{q}_j\|$, for some constant c^*.

Since $\|\mathbf{k}_i - \mathbf{k}_j\|$ is a constant approximation for $\|\mathbf{q}_i - \mathbf{q}_j\|$, it immediately follows that we can maintain a constant-weight approximation of structures like the Euclidean minimum spanning tree and the Euclidean traveling salesman tour on Q by constructing the same on $\mathbf{k}_i$'s, the centers of the hypotheses H_i's. ⌟

An important feature of our choice of a potential function is that the evolver cannot change its value by more than a constant with each step. Recall that the evolver selects an object index i and moves q_i by a distance of most αN_i in any direction. This results in at most an α fraction change in the values of l_i and s_i, and hence the resulting change in Φ_i is bounded by a constant. However, note that the movement of q_i could potentially affect the local feature sizes of a number of other objects in the system. We can show by a packing argument that there is only a constant number of indices j, whose Φ_j value is affected. Furthermore, these values are only changed by a constant. This is encapsulated in the following lemma. Its proof has been deferred to Section A.1.

▶ **Lemma 4.** *Each step of the evolver increases the potential Φ by at most a constant.*

4.2 The Algorithm

Next, we present our algorithm. The algorithm runs in parallel to the evolver, running faster by a speed-up factor of σ (whose value will be derived in our analysis). Each *step* of the algorithm involves a probe of an object by the oracle, and following that, a possible modification of a hypothesis, H_i. The total running time of the algorithm is proportional to the number of steps.

Let us begin with a high-level explanation. The details are provided in Algorithm TrackByZoom. Recall that we are given a set of n objects, each represented by X_i, a uniform probability distribution over a ball $\mathcal{B}_i = \mathcal{B}(\mathbf{q}_i, \beta N_i)$, where N_i is the distance to its nearest neighbor and β is the local feature scale. The points $\mathbf{q}_i$ are restricted to lie within a bounding ball $\mathcal{B}_0$ centered at the origin. The algorithm maintains a hypothesis in the form of n Cauchy distributions, each represented by a hypothesis ball $\mathcal{B}_i^H = \mathcal{B}(\mathbf{k}_i, h_i)$, where $\mathbf{k}_i$ is the center of the distribution and h_i is the distribution's scaling parameter.

The algorithm begins with an initial hypothesis, where each hypothesis ball is just $\mathcal{B}_0$. This reflects the fact that we effectively assume nothing about the location of $\mathbf{q}_i$, except that it lies within the bounding ball. The algorithm then proceeds in a series of *iterations*, where each iteration handles all the n indices in order. The handling of each index involves two processes, called *zoom-out* and *zoom-in*.

In the zoom-out process, we query the oracle on index i to check whether (1) the sampled point X_i lies within its hypothesis ball and (2) at least one other X_j lies within a concentric ball whose radius is larger by a factor of $\frac{1}{\beta}$. If so, we expand the hypothesis ball by a factor of $3/(1-2\beta)$. (We show in Lemma 5 that this guarantees that the hypothesis ball $\mathcal{B}_i^H$ now contains the local feature ball $\mathcal{B}_i$.) We then proceed to the zoom-in process. If not, we double the hypothesis ball and repeat (see Figure 3).

In the zoom-in process, we first check whether X_i lies within the hypothesis ball. (The evolver could have moved q_i.) If not, we return to the zoom-out process. If so, we next check the $\frac{1}{\beta}$-expansion of this ball. If there is no other X_j in this expanded ball, we accept the current hypothesis for i, and go on to the next point in the set. (We show in Lemma 10 that Φ_i at this point in time is bounded by a constant). If there is such an X_j however, we may need to shrink the hypothesis ball for the current index. We cover the hypothesis ball with a collection of $O(1)$ balls whose radii are smaller by a constant factor, and we test whether X_i

Algorithm TrackByZoom Tracking Evolving Distributions.

procedure TrackByZoom($\mathcal{O}, n, \beta, \mathcal{B}_0$)

▷ Maintain hypothesis $\mathcal{H}$ by running the procedure at a speed-up factor σ, given the membership Oracle:$\mathcal{O}$, the input size:n, the local feature parameter:β, and the the maximum bounding ball centered at origin:$\mathcal{B}_0$

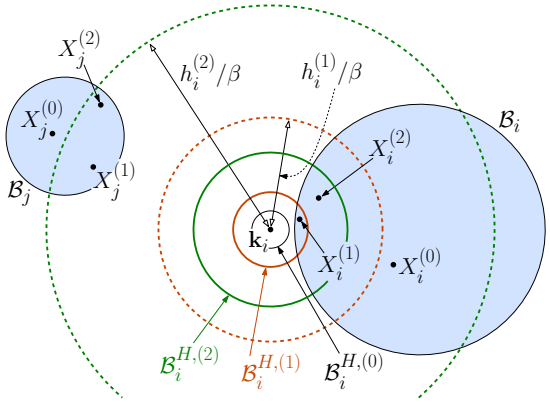
  for $i \leftarrow 1$ to n **do**
    ▷ Initially set the hypotheses according to the given bounding ball
    $h_i \leftarrow \text{radius}(\mathcal{B}_0)$, $\mathbf{k}_i \leftarrow \mathbf{0}$ ▷ Set $H_i(\mathbf{x}) \leftarrow f_{\mathbf{0},R_0}(\mathbf{x})$, as in Eq.(2: Hypothesis Def.)
  end for

  $i \leftarrow 1$ ▷ Start with the first point

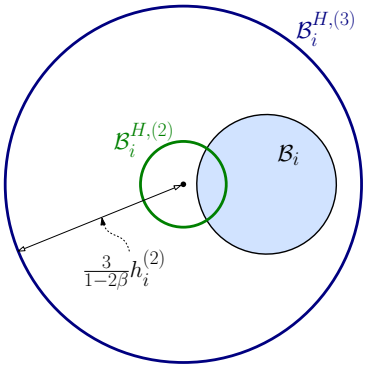
  ZOOM_OUT:
  $\mathcal{O}_i \leftarrow \mathcal{O}(i, \mathbf{k}_i, h_i)$, $\mathcal{O}_i^* \leftarrow \mathcal{O}(i, \mathbf{k}_i, h_i/\beta)$ ▷ Store values to use later
  ▷ If $\mathcal{B}_i^H$ (hypothesis ball) contains X_i, and its $1/\beta$-expansion contains some X_j
  if $\mathcal{O}_i = (Y, \cdot)$ **and** $\mathcal{O}_i^* = (Y, +)$ **then**
    $h_i \leftarrow \frac{3}{1-2\beta} h_i$ ▷ Update $\mathcal{B}_i^H$ so that it contains $\mathcal{B}_i$ (local feature of $\mathbf{q}_i$)
    go to ZOOM_IN
  end if
  $h_i \leftarrow 2h_i$ ▷ Zoom out by expanding $\mathcal{B}_i^H$
  ▷ Keep zooming out until $X_i \in \mathcal{B}_i^H$, and its $1/\beta$-expansion contains another X_j
  go to ZOOM_OUT

  ZOOM_IN:
  $\mathcal{O}_i \leftarrow \mathcal{O}(i, \mathbf{k}_i, h_i)$, $\mathcal{O}_i^* \leftarrow \mathcal{O}(i, \mathbf{k}_i, h_i/\beta)$ ▷ Store values to use later
  if $\mathcal{O}_i = (N, \cdot)$ **then** ▷ X_i no longer in $\mathcal{B}_i^H$
    go to ZOOM_OUT
  end if
  ▷ $X_i \in \mathcal{B}_i^H$, and its $1/\beta$-expansion doesn't contain another X_j
  if $\mathcal{O}_i = (Y, \cdot)$ **and** $\mathcal{O}_i^* = (Y, -)$ **then**
    $i \leftarrow 1 + i \mod n$ ▷ Φ_i is a constant, ready to move to the next point
    go to ZOOM_OUT
  end if
  ▷ If $\mathcal{B}_i^H$ (hypothesis ball) contains X_i, and its $1/\beta$-expansion contains some X_j
  if $\mathcal{O}_i = (Y, \cdot)$ **and** $\mathcal{O}_i^* = (Y, +)$ **then**
    ▷ Ψ is the set of balls of radius $h_i / \left(2\lceil \frac{3}{1-2\beta} \rceil\right)$ which cover $\mathcal{B}_i^H$
    for $\mathcal{B}(\mathbf{x}, r) \in \Psi(\mathcal{B}_i^H, 2\lceil \frac{3}{1-2\beta} \rceil)$ **do**
      if $\mathcal{O}(i, \mathbf{x}, r) = (Y, \cdot)$ **then** ▷ Found the nested ball that contains X_i
        $\mathbf{k}_i \leftarrow \mathbf{x}$, $h_i \leftarrow \frac{3}{1-2\beta} r$ ▷ Expand the nested ball so that it contains B_i
      end if
    end for
  end if
  ▷ Keep zooming in until $X_i \in \mathcal{B}_i^H$, and its $1/\beta$-expansion contains no other X_j
  go to ZOOM_IN

end procedure



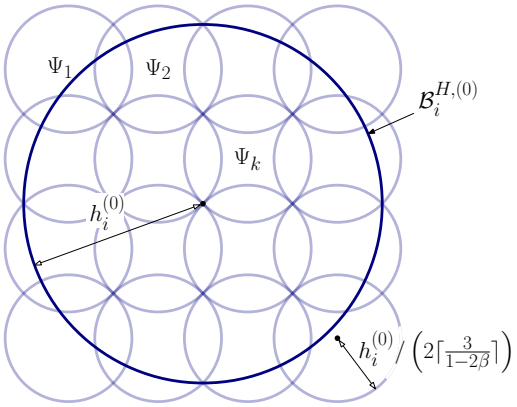
(a) Multiple stages of the zoom-out process starting at $t = 0$. Stage $t = 2$ is the first time $\mathcal{B}_i^{H,(t)}$ contains X_i, and its $1/\beta$-expansion contains X_j.



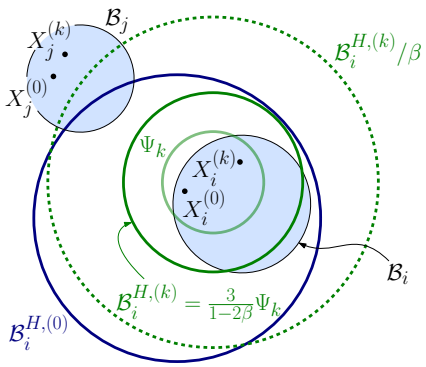
(b) $\mathcal{B}_i^{H,(3)}$ is the hypothesis ball at the end of the zoom-out process. Note that $\mathcal{B}_i \subset \mathcal{B}_i^{H,(3)}$.

Figure 3 The zoom-out process of TrackByZoom (Line 6).

lies within any of them (see Figure 4). As soon as we find one, we shrink the hypothesis ball about this ball. We repeat this process until one of the earlier conditions applies (causing us to return to the zoom-out process or move on to the next point).



(a) Set of nested balls covering $\mathcal{B}_i^{H,(0)}$, which we call Ψ. radius(Ψ_i) = radius$(\mathcal{B}_i)/\left(2\lceil\frac{3}{1-2\beta}\rceil\right)$, and $|\Psi| \in O(1)$.



(b) Ψ_k is the first nested ball to contain X_i. $\mathcal{B}_i^{H,(k)}$ contains X_i, and its $1/\beta$-expansion does not contain X_j. TrackByZoom moves on to index $i + 1$.

Figure 4 Illustration of the zoom-in process in TrackByZoom (Line 14).

4.3 Analysis

In this section, we analyze the distance that the algorithm maintains with the ground truth. Recall that the initial hypotheses are based on the bounding ball $\mathcal{B}_0$, which is centered at the origin. Letting $R_0 = \text{radius}(\mathcal{B}_0)$, we have $H_i^{(0)}(\mathbf{x}) = f_{\mathbf{0},R_0}(\mathbf{x})$, as defined in Eq. (2: Hypothesis Def.). Using Eq. (3), the initial potential function satisfies:

$$\Phi_i^{(0)} = \log\sqrt{R_0/l_i^{(0)}}, \quad \text{and} \quad \Phi^{(0)} \in O(n \log \Lambda_0) \qquad (\Lambda_0\text{:Aspect Ratio}) \tag{5}$$

Suppose the condition in line 8, and 23 are satisfied. We prove the following lemma to justify the expansion factor $\frac{3}{1-2\beta}$. The proof is rather technical and has been deferred to the full version of the paper.

▶ **Lemma 5** (Expansion Factor). *If $\mathcal{O}(i, \mathbf{k}_i, h_i) = (Y, \cdot)$ and $\mathcal{O}(i, \mathbf{k}_i, h_i/\beta) = (Y, +)$, then $\mathcal{B}_i \subseteq \mathcal{B}\left(\mathbf{k}_i, \frac{3}{1-2\beta} h_i\right)$.*

Let us consider how Algorithm TrackByZoom affects the potential. For a particular index i, during the *zoom-out* process, we double the radius h_i of the hypothesis ball. If the Euclidean distance between the centers of the hypothesis, and the local-feature ball is much larger than h_i, then $\Phi = \log(\max(s_i, l_i, h_i)/\sqrt{l_i h_i}) = \log(s_i/\sqrt{l_i h_i})$ decreases by an additive constant. However, once h_i is greater than s_i, the algorithm risks increasing Φ. We show that the number of steps where Φ increases is a constant for every index i. However, during the zoom-in process, we maintain the invariant that the hypothesis ball contains the local-feature ball (Lemma 5), which implies that $\max(s_i, l_i, h_i) = h_i$. Therefore, Φ decreases by a constant amount whenever we shrink the hypothesis ball by a constant factor. However, if the evolver moves q_i while index i is being processed, our algorithm may transition to the zoom-out process again. Hence, if the evolver executes some $\varepsilon^{\langle z \rangle}$ steps in the zth iteration, our algorithm may increase Φ by only $O\left(n + \varepsilon^{\langle z \rangle}\right)$. We obtain the following lemma. The proof is deferred to Section A.1

▶ **Lemma 6** (Algorithm's contribution towards Φ). *For any iteration z, let $\varepsilon^{\langle z \rangle}$ denote the number of steps executed by the evolver. Then TrackByZoom increases Φ in $O\left(n + \varepsilon^{\langle z \rangle}\right)$ steps, each time by $O(1)$. In each of the remaining steps of the zth iteration, it decreases Φ by $\Theta(1)$ in an amortized sense.*

Let $\Phi^{\langle z \rangle}$ be the potential function at the start of the zth iteration. And let $\Delta^{\langle z \rangle}$ be the time taken by the zth iteration of the algorithm. At the conclusion of the *zoom-in* process for a particular index i, the algorithm fixes a hypothesis H_i which is at a reasonable distance from the truth P_i. In fact, we show in Lemma 10 that at this point in time, Φ_i is ϕ_0, a constant. Therefore, by Lemma 6, TrackByZoom running at a constant speed-up σ roughly spends $\sim \Phi_i^{\langle z \rangle}$ time (within constant factors), in reducing $\Phi_i^{\langle z \rangle}$ to ϕ_0. Summing over all indices, we see that the time taken by zth iteration, $\Delta^{\langle z \rangle}$, is $\sim \Phi^{\langle z \rangle}$. Intuitively, for a sufficiently large speed-up factor, the actions of the evolver can only affect $\Delta^{\langle z \rangle}$ by a small fraction of this amount. We summarize these observations in the following lemma. The proof is deferred to Section A.1

▶ **Lemma 7** (Iteration time proportional to Potential). *There exists a constant speed-up factor σ for TrackByZoom such that $\Delta^{\langle z \rangle} \in \Theta_\sigma\left(n + \Phi^{\langle z \rangle}\right)$.*

We are now ready to derive how $\Phi^{\langle z \rangle}$ changes over the course of multiple iterations. We show that for a sufficiently large (constant) speed-up factor σ, after $\sim \log\log \Lambda_0$ iterations Φ converges to $O(n)$. (Recall that Λ_0 is the initial aspect ratio.)

▶ **Lemma 8.** *There exists a constant speed-up factor σ for Algorithm TrackByZoom and $z_0 \in \mathbb{Z}$, such that for all $z > z_0$, $\Phi^{\langle z \rangle} \in O(n)$*

Proof. By Lemma 4, in the zth iteration the evolver takes at most $\varepsilon^{\langle z \rangle} = O(\Delta^{\langle z \rangle})$ steps and increases Φ by at most $O(\Delta^{\langle z \rangle})$. Therefore, by Lemma 6, TrackByZoom increases Φ by $O(n + \Delta^{\langle z \rangle})$. Since the algorithm runs at a speed-up factor σ it takes $\sigma\Delta^{\langle z \rangle}$ steps, and by Lemma 6, it decreases Φ by at least $\lambda(\sigma\ \Delta^{\langle z \rangle} - O(n + \Delta^{\langle z \rangle}))$, for some $\lambda \in O(1)$. Accounting for all the increases and decreases we have:

$$\begin{aligned}\Phi^{\langle z+1\rangle} &\leq \Phi^{\langle z\rangle} + O(\Delta^{\langle z\rangle}) + O(n + \Delta^{\langle z\rangle}) - \lambda(\sigma\ \Delta^{\langle z\rangle} - O(n + \Delta^{\langle z\rangle}))\\ &\leq \Phi^{\langle z\rangle} + O(\Delta^{\langle z\rangle}) - \sigma\lambda\Delta^{\langle z\rangle} + O(n).\end{aligned}$$

There exists a sufficiently large $\sigma \in O(1)$ such that $\sigma\lambda\Delta^{\langle z\rangle} - O(\Delta^{\langle z\rangle}) \geq a\Delta^{\langle z\rangle}$, for $0 < a \in O(1)$. So,

$$\Phi^{\langle z+1\rangle} \leq \Phi^{\langle z\rangle} - a\Delta^{\langle z\rangle} + O(n).$$

By Lemma 7, for a sufficiently large (constant) σ, there exists $c_\sigma > 0$ such that $\Delta^{\langle z\rangle} \geq c_\sigma \left(n + \Phi^{\langle z\rangle}\right)$. Therefore,

$$\Phi^{\langle z+1\rangle} \leq (1 - ac_\sigma)\Phi^{\langle z\rangle} + (1 - ac_\sigma)O(n) \leq (1 - ac_\sigma)^{z+1}\Phi^{\langle 0\rangle} + \left(\sum_{y=1}^{z+1}(1 - ac_\sigma)^y\right) O(n).$$

By Eq. (5), $\Phi^{\langle 0\rangle} = \Phi^{(0)} \in O(n \log \Lambda_0)$, and since there exists $b > 1$ such that $(1 - ac_\sigma) < 1/b < 1$, we have

$$\Phi^{\langle z+1\rangle} \leq \frac{O(n \log \Lambda_0)}{b^{z+1}} + O(n). \tag{6}$$

Observe that this is $O(n)$ whenever $z > \log\log \Lambda_0$. ◀

If $\Phi^{\langle z\rangle} \in O(n)$, then by Lemma 7, $\Delta^{\langle z\rangle} \in \Theta(n)$. Thus, Φ increases by at most $O(n)$ throughout iteration z. Therefore, for any time t during iteration z, $\Phi^{(t)} \in O(n)$ as well.

Using Eq. (6) we observe that for all $z > 1$, $\Phi^{\langle z\rangle} \leq O(n \log \Lambda_0)$. By Lemma 7, this implies that $\Delta^{\langle z\rangle} \in \Theta(n \log \Lambda_0)$. Hence, for some $t_0 \in O(n \log \Lambda_0 \cdot \log\log \Lambda_0)$, we have $\Phi^{(t)} \in O(n)$, for all $t > t_0$.

By combining Lemmas 8 and 3, we complete the proof of Theorem 1. It is noteworthy that there is a trade-off between the speed-up factor σ and the motion parameter α. Recall that with each step, the evolver can move q_i by a distance of up to αN_i. If we reduce this parameter to $\alpha' = (1 + \alpha)^{\frac{1}{c}} - 1$, then it takes the evolver at least c steps to effect the same change as before. Thus, by reducing the local-motion factor, it is possible to achieve a speed-up factor of $\sigma = 1$. Formally we have,

▶ **Corollary 9.** *There exists a motion parameter $\alpha' < 1$ such that for a set of n evolving objects in $\mathbb{R}^d$ under the (α', β)-local-motion model, where $\beta < \frac{1}{3}$, Algorithm* TrackByZoom *(with speed-up factor 1) maintains a distance of $O(n)$ for all $t \geq t_0$, $t_0 \in O(n \log \Lambda_0 \log\log \Lambda_0)$*

5 Extensions and Conclusion

The local-motion model introduced in Section 2 assumes a uniform probability distribution for each object. However, the algorithm TrackByZoom with an appropriate speed-up factor can be applied to a wider class of probability distributions. A distribution P_i in this class has $\mathcal{B}_i$, the volume of the ith object, as its support, and is scaled according to l_i, the local feature size. Many natural truncated distributions [11], such as the uniform distribution, the truncated normal distribution [27], the truncated Cauchy distribution belong to this family. Details are presented in the full version of the paper.

An additional refinement to the motion model we suggested might involve redefining the local feature region and the speed of a point based on the distance to its k-nearest neighbors for a fixed k, instead of just using the nearest neighbor distance. We conjecture that our principal results extend to this generalization as well, given a suitable range counting

oracle. Yet another variation of the model could possibly involve letting the uncertainty associated with a point have unbounded support (for e.g. a normal rather than a truncated normal distribution). This is a significantly harder problem, as a point may lie outside its local feature with a constant probability, requiring a maintaining algorithm to have a super-constant speed-up. We aim to study these in the future.

References

1 A. Acharya and D. M. Mount. Optimally tracking labels on an evolving tree. In *Proc. 34th Canad. Conf. Comput. Geom.*, pages 1–8, 2022.

2 A. Anagnostopoulos, R. Kumar, M. Mahdian, and E. Upfal. Sorting and selection on dynamic data. *Theoretical Computer Science*, 412(24):2564–2576, 2011. doi:10.1016/j.tcs.2010.10.003.

3 A. Anagnostopoulos, R. Kumar, M. Mahdian, E. Upfal, and F. Vandin. Algorithms on evolving graphs. In *Proc. 3rd Innov. Theor. Comput. Sci. Conf.*, pages 149–160, 2012. doi:10.1145/2090236.2090249.

4 M. Beermann and A. Sieben. The connection between stress, density, and speed in crowds. *Scientific Reports*, 13:13626, 2023. doi:10.1038/s41598-023-39006-8.

5 J. J. Besa, W. E. Devanny, D. Eppstein, M. T. Goodrich, and T. Johnson. Optimally sorting evolving data. In *45th Internat. Colloq. Autom., Lang., and Prog.*, pages 81:1–81:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.81.

6 J. J. Besa, W. E. Devanny, D. Eppstein, M. T. Goodrich, and T. Johnson. Quadratic time algorithms appear to be optimal for sorting evolving data. In *Proc. 20th Workshop Algorithm Eng. and Exp.*, pages 87–96, 2018.

7 A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 2005.

8 K. Buchin, M. Löffler, P. Morin, and W. Mulzer. Delaunay triangulation of imprecise points simplified and extended. *Algorithmica*, 61:674–693, 2011. doi:10.1007/s00453-010-9430-0.

9 D. Busto, W. Evans, and D. Kirkpatrick. Minimizing interference potential among moving entities. In *Proc. 30th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 2400–2418, 2019. doi:10.5555/3310435.3310582.

10 T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 2012. doi:10.1002/047174882X.

11 Y. Dodge. *The Oxford Dictionary of Statistical Terms*. Oxford Univ. Press, 2003. doi:10.1002/sim.1812.

12 W. Evans and D. Kirkpatrick. Minimizing query frequency to bound congestion potential for moving entities at a fixed target time. *Algorithms*, 17:246, 2024. doi:10.3390/a17060246.

13 J. Gudmundsson, P. Laube, and T. Wolle. Computational movement analysis. In W. Kresse and D. Danko, editors, *Handbook of Geogr. Inf.*, pages 725–741. Springer, 2012. doi:10.1007/978-3-540-72680-7_22.

14 L. J. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry: Building robust algorithms from imprecise computations. In *Proc. Fifth Annu. Sympos. Comput. Geom.*, pages 208–217, 1989. doi:10.1145/73833.73857.

15 D. Helbing, L. Buzna, A. Johansson, and T. Werner. Self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions. *Transportation Science*, 39:1–24, 2005. doi:10.1287/trsc.1040.0108.

16 O. Hesham and G. Wainer. Context-sensitive personal space for dense crowd simulation. In *Proc. Symp. Simulation for Architecture and Urban Design*, pages 1–8, 2017. URL: https://dl.acm.org/doi/10.5555/3289787.3289806.

17 Q. Huang, X. Liu, X. Sun, and J. Zhang. Partial sorting problem on evolving data. *Algorithmica*, 79:960–983, 2017. doi:10.1007/s00453-017-0295-3.

18 D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40:1098–1101, 1952. doi:10.1109/JRPROC.1952.273898.

19 U. Jain. Effects of population density and resources on the feeling of crowding and personal space. *J. Social Psych.*, 127:331–338, 1987. doi:10.1080/00224545.1987.9713699.

20 S. Kahan. A model for data in motion. In *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pages 265–277, 1991. doi:10.1145/103418.103449.

21 V. Kanade, N. Leonardos, and F. Magniez. Stable matching with evolving preferences. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2016.

22 M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30:81–93, 1938. doi:10.2307/2332226.

23 S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22:79–86, 1951. doi:10.1214/aoms/1177729694.

24 P. Laube. *Computational Movement Analysis*. Springer Cham, 2014. doi:10.1007/978-3-319-10268-9.

25 M. Löffler and M. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56:235–269, 2010. doi:10.1007/s00453-008-9174-2.

26 D. J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003. URL: https://books.google.com/books?id=AKuMj4PN_EMC.

27 F. Nielsen. Statistical divergences between densities of truncated exponential families with nested supports: Duo Bregman and duo Jensen divergences. *Entropy*, 24:421, 2022. doi:10.3390/e24030421.

28 R. Pan, Z. Han, T. Liu, H. Wang, J. Huang, and W. Wang. An RFID tag movement trajectory tracking method based on multiple RF characteristics for electronic vehicle identification ITS applications. *Sensors*, 23:7001, 2023. doi:10.3390/s23157001.

29 A. Seyfried, B. Steffen, W. Klingsch, and M. Boltes. The fundamental diagram of pedestrian movement revisited. *J. Stat. Mech. Theory Exp.*, 2005:P10002, 2005. doi:10.1088/1742-5468/2005/10/P10002.

30 C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:623–656, 1948. doi:10.1002/j.1538-7305.1948.tb00917.x.

31 S. M. Tomkiewicz, M. R. Fuller, J. G. Kie, and K. K. Bates. Global positioning system and associated technologies in animal behaviour and ecological research. *Phil. Trans. R. Soc. B*, 365:2163–2176, 2010. doi:10.1098/rstb.2010.0090.

32 M. van Kreveld, M. Löffler, F. Staals, and L. Wiratma. A refined definition for groups of moving entities and its computation. *Internat. J. Comput. Geom. Appl.*, 28(2):181–196, 2018. doi:10.4230/LIPIcs.ISAAC.2016.130.

33 S. Verdú. The Cauchy distribution in information theory. *Entropy*, 25:346, 2023. doi:10.3390/e25020346.

34 L. Wiratma, M. Löffler, and F. Staals. An experimental comparison of two definitions for groups of moving entities. In *Proc. 10th Internat. Conf. Geogr. Inf. Sci.*, pages 64:1–64:6, 2018. doi:10.4230/LIPIcs.GIScience.2018.64.

35 S. Yi, H. Li, and X. Wang. Understanding pedestrian behaviors from stationary crowd groups. In *Proc. IEEE Conf. Comput. Vis. Pattern Recogn.*, pages 3488–3496, 2015. doi:10.1109/CVPR.2015.7298971.

36 Y. Zou, G. Zeng, Y. Wang, X. Liu, X. Sun, J. Zhang, and Q. Li. Shortest paths on evolving graphs. In H. T. Nguyen and V. Snasel, editors, *Computational Social Networks*, pages 1–13. Springer, 2016. doi:10.1007/978-3-319-42345-6_1.

A Appendix

A.1 Deferred Proofs

Note that some of the restatements of the lemmas given here provide additional details, which were not present in the original statements.

The next lemma applies whenever Algorithm TrackByZoom has completed its processing of an object on Line 20. It shows that the potential value for this object does not exceed a constant.

▶ **Lemma 10** (Done tracking X_i). *There exists a constant ϕ_0, such that whenever Algorithm TrackByZoom completes its processing of any object i, $\Phi_i \leq \phi_0 \in O(1)$.*

Proof. The algorithm moves on to the next point when the condition in line 19 is satisfied ($\mathcal{O}(i, \mathbf{k}_i, h_i) = (Y, \cdot)$ and $\mathcal{O}(i, \mathbf{k}_i, h_i/\beta) = (Y, -)$). Since $\mathcal{O}(i, \mathbf{k}_i, h_i) = (Y, \cdot)$, $\mathcal{B}_i^H = \mathcal{B}(\mathbf{k}_i, h_i)$ intersect. Moreover, if $\mathbf{q}_j$ is the current nearest neighbor of $\mathbf{q}_i$, $\mathcal{O}(i, \mathbf{k}_i, h_i/\beta) = (Y, -)$ implies X_j lies outside the $1/\beta$ expansion of $\mathcal{B}_i^H$ (see Figure 5).

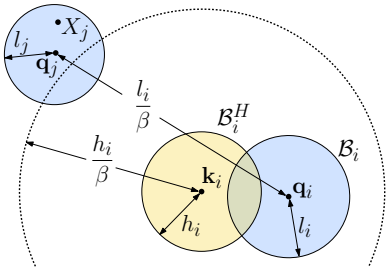


Figure 5 Proof of Lemma 10.

Now because $\mathbf{q}_j$ is the nearest neighbor of $\mathbf{q}_i$, $N_j \leq N_i$, hence $l_j \leq l_i$. By triangle inequality, the following series of inequality holds:

$$\frac{h_i}{\beta} \;\leq\; \|X_j - \mathbf{k}_i\| \;\leq\; \|X_j - \mathbf{q}_i\| + \|\mathbf{q}_i - \mathbf{k}_i\| \;\leq\; \left(\frac{l_i}{\beta} + l_j\right) + (h_i + l_i) \;\leq\; \left(\frac{1}{\beta} + 2\right) l_i + h_i,$$

implying that $\frac{1-\beta}{1+2\beta} h_i \leq l_i$.

Now consider the hypothesis ball, $\mathcal{B}_i^{H^-}$, set by the algorithm in the previous step (via Line 26). From Lemma 5, we have $\mathcal{B}_i \subseteq \mathcal{B}_i^{H^-}$. Let $h_i^- = \text{radius}\left(\mathcal{B}_i^{H^-}\right)$. Since the algorithm shrinks the hypothesis ball by at least a factor of 2 during the zoom-in process, h_i^- is at least $2h_i$. And therefore $2h_i \geq l_i$. We also have $s_i \leq h_i + l_i$ (see Figure 5). Therefore,

$$\Phi_i = \log\left(\frac{\max(s_i, l_i, h_i)}{\sqrt{l_i h_i}}\right) \leq \log\left(\frac{l_i + h_i}{\sqrt{l_i h_i}}\right)$$
$$\leq \log\left(\frac{2h_i + h_i}{\sqrt{\frac{1-\beta}{1+2\beta} h_i \cdot h_i}}\right) \leq \phi_0, \quad \text{for } \phi_0 = \log\left(3 \cdot \sqrt{\frac{1+2\beta}{1-\beta}}\right). \qquad \blacktriangleleft$$

▶ **Theorem 2** (Lower Bound on the Distance). *For any algorithm $\mathcal{A}$, there exists a starting configuration $Q^{(0)}$ and an evolver (with knowledge of $\mathcal{A}$) in the local-motion model such that, for any positive integer t_0, there exists $t \geq t_0$, such that $\mathcal{D}^{(t)} = \mathcal{D}(\mathcal{P}^{(t)}, \mathcal{H}^{(t)}) \in \Omega(n)$.*

Proof. We construct a point set on the real line ($\mathbb{R}$), and we will mention at the end how to adapt the proof to any dimension d. Given the small constant β, the local feature constant, we define $Q^{(0)}$ to be the following set of $n = 2m$ points in $\mathbb{R}$.

$$Q^{(0)} \;=\; \bigcup_{i\in[m]} \left\{a_i^{(0)} = 100i\right\} \cup \bigcup_{i\in[m]} \left\{b_i^{(0)} = 100i + 1\right\}.$$

Let $\mathcal{D}_{a,i} = \mathcal{D}_{2(k-1)+1},\ k \in [m]$, the distance corresponding to the first point in the tuple. We similarly define $P_{a,i}$ and $H_{a,i}$. Let N_i denote the current distance to a_i's nearest neighbor. The local feature interval for a_i, denoted I_i, is $100i + \beta N_i[-1, 1]$. Recall that $P_{a,i}$ is the uniform probability over I_i. Therefore, $P_{a,i}(\mathbf{x}) = 1/|\mathcal{I}_i|$, where $|\mathcal{I}_i| = 1/\beta N_i$ is the diameter of $\mathcal{I}_i$. Now

$$\begin{aligned}\mathcal{D}_{a,i}^{(t_0)} \;&=\; \int_{\mathbf{x}\in\mathcal{I}_i} P_{a,i}^{(t_0)}(\mathbf{x}) \log \frac{P_{a,i}^{(t_0)}(\mathbf{x})}{H_{a,i}^{(t_0)}(\mathbf{x})} d\mathbf{x} \\ &=\; \int_{\mathbf{x}\in\mathcal{I}_i} P_{a,i}^{(t_0)}(\mathbf{x}) \log P_{a,i}^{(t_0)}(\mathbf{x}) d\mathbf{x} \;-\; \int_{\mathbf{x}\in\mathcal{I}_i} P_{a,i}^{(t_0)}(\mathbf{x}) \log H_{a,i}^{(t_0)}(\mathbf{x}) d\mathbf{x}.\end{aligned}$$

By the concavity of the log function and Jensen's inequality, we have

$$\begin{aligned}\mathcal{D}_{a,i}^{(t_0)} \;&\geq\; -\log|\mathcal{I}_i| \int_{\mathbf{x}\in\mathcal{I}_i} P_{a,i}^{(t_0)}(\mathbf{x}) d\mathbf{x} \;-\; \log \int_{\mathbf{x}\in\mathcal{I}_i} H_{a,i}^{(t_0)}(\mathbf{x}) P_{a,i}^{(t_0)}(\mathbf{x}) d\mathbf{x} \\ &=\; -\log|\mathcal{I}_i| - \log \frac{\int_{\mathbf{x}\in\mathcal{I}_i} H_{a,i}^{(t_0)}(\mathbf{x}) d\mathbf{x}}{|\mathcal{I}_i|} \;=\; -\log \int_{\mathbf{x}\in\mathcal{I}_i} H_{a,i}^{(t_0)}(\mathbf{x}) d\mathbf{x}. \qquad (7)\end{aligned}$$

We may assume that $\mathcal{D}^{(t_0)} \in o(n)$ (for otherwise we can set $t = t_0$ and are done). This implies there are only $o(n)$ many a_i's such that $\mathcal{D}_{a,i}^{(t_0)} \in \Omega(1)$. Call the remaining $n/2 - o(n)$, a_i's the proximal set, denoted T_a.

We are now ready to describe the evolver's actions. It chooses to stay dormant until time t_0. For a large enough constant M, consider the time interval $[t_0, t_0 + n/M]$. The algorithm $\mathcal{A}$, running at a speed-up factor of σ, can modify at most $\sigma n/M$ hypotheses in the time interval. The evolver chooses not to move any of those points. Therefore $\mathcal{A}$ can only reduce $\mathcal{D}^{(t_0)}$ by $o(n)$ over this time interval. Now there are at least $n/2 - \sigma n/M - o(n)$ members in the proximal set T_a, whose hypotheses were not altered by $\mathcal{A}$. Call this set the stable set, denoted S_a.

For $\kappa = \lceil \log(2/(1+\alpha)) \rceil$, the evolver selects some $n/(\kappa M)$ members from S_a. Call that set S'_a. To be specific, for all $a_i \in S'_a$, the evolver chooses to move a_i away from b_i some κ times, by a distance of exactly $\alpha N_{a,i}$, where $N_{a,i}$ is the distance of a_i from its current nearest neighbor. (Note that the nearest neighbor will remain b_i throughout these operations.) Given this value of κ, after the conclusion of these operations, the distance between a_i and b_i is at least 2. For $\beta < 1/3$, the local feature of a_i changes to an interval $\mathcal{I}'_i$, where $\mathcal{I}'_i \cap \mathcal{I}_i = \emptyset$. Now, for $t = t_0 + n/M$, using a similar analysis as Eq. (7), we have

$$\mathcal{D}_{a,i}^{(t)} \;\geq\; -\log \int_{\mathbf{x}\in\mathcal{I}'_i} H_{a,i}^{(t)}(\mathbf{x}) d\mathbf{x}. \qquad (8)$$

Thus, for all $a_i \in S'_a$, we have $H_{a,i}^{(t)} = H_{a,i}^{(t_0)}$ and $D_{a,i}^{(t_0)} = o(1)$. Therefore,

$$\int_{\mathbf{x}\in\mathcal{I}_i} H_{a,i}^{(t_0)}(\mathbf{x}) d\mathbf{x} \;\geq\; e^{-o(1)}.$$

If $\mathcal{D}_{a,i}^{(t)} \in o(1)$ as well, then similarly from Eq. (8), we have

$$\int_{\mathbf{x}\in\mathcal{I}_i'} H_{a,i}^{(t)}(\mathbf{x})d\mathbf{x} \;\geq\; e^{-o(1)}.$$

But, this yields a contradiction since $\mathcal{I}_i' \cap \mathcal{I}_i = \emptyset$ and $H_{a,i}^{(t)} = H_{a,i}^{(t_0)}$ implies that

$$\int_{\mathbf{x}\in\mathcal{I}_i'\cup\mathcal{I}_i} H_{a,i}^{(t)}(\mathbf{x})d\mathbf{x} \;\geq\; 2e^{-o(1)} \;>\; 1.$$

Therefore, for $a_i \in S_a'$, $\mathcal{D}_{a,i}^{(t)} \in \Omega(1)$, and since $|S_a'| = \Omega(n)$, we have $\mathcal{D}^{(t)} \in \Omega(n)$.

For a general dimension d, we define Q, in the same way except all the points lie on a single axis. The evolver also moves these points along that particular axis. The rest of the analysis involves integration over a region of space rather an interval, but is straightforward and carries over to $\mathbb{R}^d$. ◀

▶ **Lemma 4** (Evolver's contribution to Φ)**.** *Every step of the evolver increases the potential function Φ by at most a constant.*

Proof. Let's say the evolver chooses to move $\mathbf{q}_i$ at time 0. Now $l_i^{(0)}/\beta$ is the nearest neighbor distance of $\mathbf{q}_i$ at time 0. Therefore $\mathbf{q}_i$ moves by at most $(\alpha/\beta)l_i^{(0)}$ distance, implying $s_i^{(1)} \leq s_i^{(0)} + (\alpha/\beta)l_i^{(0)} \leq (1+\alpha/\beta)\max\left(s_i^{(0)}, l_i^{(0)}, h_i^{(0)}\right)$. Similarly $(1-\alpha)l_i^{(0)} \leq l_i^{(1)} \leq (1+\alpha)l_i^{(0)}$. Therefore,

$$\begin{aligned}
\Phi_i^{(1)} - \Phi_i^{(0)} \;&\leq\; \log\frac{\max\left(s_i^{(1)}, l_i^{(1)}, h_i^{(1)}\right)}{\sqrt{l_i^{(1)}h_i^{(1)}}} - \log\frac{\max\left(s_i^{(0)}, l_i^{(0)}, h_i^{(0)}\right)}{\sqrt{l_i^{(0)}h_i^{(0)}}} \\
&\leq\; \log\frac{\max\left(s_i^{(0)} + (\alpha/\beta)l_i^{(0)}, (1+\alpha)l_i^{(0)}, h_i^{(0)}\right)}{\sqrt{(1-\alpha)l_i^{(0)}h_i^{(0)}}} \;-\; \log\frac{\max\left(s_i^{(0)}, l_i^{(0)}, h_i^{(0)}\right)}{\sqrt{l_i^{(0)}h_i^{(0)}}} \\
&\leq\; \log\left(\frac{1+\alpha/\beta}{\sqrt{1-\alpha}}\frac{\max\left(s_i^{(0)}, l_i^{(0)}, h_i^{(0)}\right)}{\sqrt{l_i^{(0)}h_i^{(0)}}}\right) \;-\; \log\frac{\max\left(s_i^{(0)}, l_i^{(0)}, h_i^{(0)}\right)}{\sqrt{l_i^{(0)}h_i^{(0)}}} \\
&=\; \log\frac{1+\alpha/\beta}{\sqrt{1-\alpha}} \in O(1)
\end{aligned} \tag{9}$$

Let $\mathcal{N}_i$ be the set of indices of points whose nearest neighbor at time 0 was $\mathbf{q}_i^{(0)}$, or whose nearest neighbor at time 1 was $\mathbf{q}_i^{(1)}$. Only points with indices in this set have their potential function changed. If $j \in \mathcal{N}_i$, $s_j^{(0)}$, and $h_j^{(0)}$ do not change. Since $\mathbf{q}_i$ moves by at most $(\alpha/\beta)l_i^{(0)}$ distance, the nearest neighbor distance $N_j = l_j/\beta$ changes by at most $(\alpha/\beta)l_i^{(0)}$ as well. Therefore,

$$-(\alpha/\beta)l_i^{(0)} \;\leq\; N_j^{(1)} - N_j^{(0)} \;\leq\; (\alpha/\beta)l_i^{(0)} \;\implies\; -\alpha l_i^{(0)} \;\leq\; l_j^{(1)} - l_j^{(0)} \;\leq\; \alpha l_i^{(0)} \tag{10}$$

Since $(1-\alpha)l_i^{(0)} \;\leq\; l_i^{(1)}$, we also have

$$-\frac{\alpha}{1-\alpha}l_i^{(1)} \;\leq\; l_j^{(1)} - l_j^{(0)} \;\leq\; \frac{\alpha}{1-\alpha}l_i^{(1)} \tag{11}$$

If $\mathbf{q}_i^{(0)}$ was the nearest neighbor of $\mathbf{q}_j^{(0)}$, then $N_j^{(0)} \leq N_i^{(0)}$, and hence $l_j^{(0)} \leq l_i^{(0)}$. Using the last fact, and dividing Eq. (10) by $l_j^{(0)}$, we have $(1-\alpha) \leq l_j^{(1)}/l_j^{(0)} \leq (1+\alpha)$. Similarly, If $\mathbf{q}_i^{(1)}$ was the nearest neighbor of $\mathbf{q}_j^{(1)}$, then using Eq. (11) we have $(1-\alpha/(1-\alpha)) \leq l_j^{(1)}/l_j^{(0)} \leq (1+\alpha/(1-\alpha))$. Therefore $l_j^{(1)}/l_j^{(0)} \in \Theta(1)$ for $j \in \mathcal{N}_i$. And hence,

$$\begin{aligned}
\Phi_j^{(1)} - \Phi_j^{(0)} &= \log \frac{\max\left(s_i^{(1)}, l_i^{(1)}, h_i^{(1)}\right)}{\sqrt{l_i^{(1)} h_i^{(1)}}} - \log \frac{\max\left(s_i^{(0)}, l_i^{(0)}, h_i^{(0)}\right)}{\sqrt{l_i^{(0)} h_i^{(0)}}} \\
&= \log \frac{\max\left(s_i^{(0)}, l_i^{(1)}, h_i^{(0)}\right)}{\max\left(s_i^{(0)}, l_i^{(0)}, h_i^{(0)}\right)} - \log \frac{\sqrt{l_i^{(1)} h_i^{(0)}}}{\sqrt{l_i^{(0)} h_i^{(0)}}} \\
&\leq \left|\log \frac{l_i^{(1)}}{l_i^{(0)}}\right| + \left|\log \frac{l_i^{(0)}}{l_i^{(1)}}\right| \in O(1), \quad \forall\, j \in \mathcal{N}_i
\end{aligned} \tag{12}$$

Finally, we show that $|\mathcal{N}_i| \in O(1)$. To that effect we solve a general problem: What is the maximum number of points in Q, whose nearest neighbor is $\mathbf{q}_1$? Without loss of generality, let $\mathbf{q}_2$ be the nearest neighbor of $\mathbf{q}_1$, such that $\|\mathbf{q}_2 - \mathbf{q}_1\| = 1$. Let $\mathbf{q}_3$'s nearest neighbor be $\mathbf{q}_1$. Consider the line segment $\overline{\mathbf{q}_1\mathbf{q}_3}$, and its intersection with $\mathcal{B}(\mathbf{q}_1, 1)$ the ball centered at $\mathbf{q}_1$, and passing through $\mathbf{q}_2$. Call the intersection point $\mathbf{q}_3'$. For any general point $\mathbf{q}_x \in Q$, we similarly define $\mathbf{q}_x'$. Note $\mathbf{q}_2' = \mathbf{q}_2$. By triangle inequality, we have $\|\mathbf{q}_3' - \mathbf{q}_2'\| \geq \|\mathbf{q}_3 - \mathbf{q}_2\| - \|\mathbf{q}_3 - \mathbf{q}_3'\| = \|\mathbf{q}_3 - \mathbf{q}_2\| - \|\mathbf{q}_3 - \mathbf{q}_1\| + \|\mathbf{q}_1 - \mathbf{q}_3'\|$. Since $\mathbf{q}_1$ is the nearest neighbor of $\mathbf{q}_3$, we have $\|\mathbf{q}_3 - \mathbf{q}_2\| \geq \|\mathbf{q}_3 - \mathbf{q}_1\|$. Therefore $\|\mathbf{q}_3' - \mathbf{q}_2'\| \geq \|\mathbf{q}_1 - \mathbf{q}_3'\| = 1$. Therefore, for any $x, y \in [n], x \neq y$, such that $\mathbf{q}_1$ is the nearest neighbor of $\mathbf{q}_x$ and $\mathbf{q}_y$, we have $\|\mathbf{q}_x' - \mathbf{q}_y'\| \geq 1$. Let $\mathcal{M}_1 = \{\mathbf{q}_x' \mid \mathbf{q}_1 \text{is the nearest neighbor of } \mathbf{q}_x\}$. Draw a ball of radius $1/2$ around $\mathbf{q}_1$, and every member of $\mathcal{M}_1$. Each of these balls are non-overlapping. However all of them are contained within a ball around $\mathbf{q}_1$ of radius $3/2$. Therefore $|\mathcal{M}_1| < (3/2)^d/(1/2)^d = 3^d \in O(1)$. Observing that $\mathcal{N}_i \leq 2\mathcal{M}_1$, and using Eq. (9), and 12 we have the result. ◀

▶ **Lemma 6** (Algorithm's contribution towards Φ). *For any iteration z, let $\varepsilon^{\langle z \rangle}$ denote the number of steps executed by the evolver. Then* TrackByZoom *increases Φ in $O\left(n + \varepsilon^{\langle z \rangle}\right)$ steps, each time by $O(1)$. In each of the remaining steps of the zth iteration, it decreases Φ by $\Theta(1)$ in an amortized sense.*

Proof. For a particular index i, we first concentrate on the zoom-out routine of our Algorithm TrackByZoom (see Line 6). Assume that the evolver leaves q_i untouched for the time being. We show later how to handle the case where the evolver moves q_i, while the algorithm is processing the index i. The algorithm increases h_i by a constant factor in Line 12. Therefore, $\Phi = \log(\max(s_i, l_i, h_i)/\sqrt{l_i h_i})$ decreases by a constant amount whenever $\max(s_i, l_i, h_i) \neq h_i$. Now suppose $\max(s_i, l_i, h_i) = h_i$ at time t, for the first time during the zoom-out routine. That implies a 2-expansion of $\mathcal{B}_i^H$ in line 12 contains the entire $\mathcal{B}_i$. Therefore, a further $1/\beta$-expansion definitely contains another X_j, $j \neq i$, satisfying the condition in line 8. We conclude that once $\max(s_i, l_i, h_i) = h_i$, the algorithm only increases Φ by a constant amount before moving on to the zoom-in routine of the algorithm (line 14).

Now, we look at the zoom-in routine of the algorithm (line 14). The algorithm only zooms in as long as the condition in line 23 is satisfied. Since we expand $\mathcal{B}_i^H$ with an expansion factor $\frac{3}{1-2\beta}$ in line 26, using Lemma 5 we conclude that $\mathcal{B}_i \subset \mathcal{B}_i^H$ in every step during the zoom-in routine. That implies $\max(s_i, l_i, h_i) = h_i$ throughout the zoom-in process, further implying $\Phi_i = \log\sqrt{h_i/l_i}$.

In line 24 we subdivide the hypothesis ball into $|\Psi| = \left(2\lceil\frac{3}{1-2\beta}\rceil\sqrt{d}\right)^d \in O(1)$ balls and make constant number of queries to the oracle. Therefore, h_i reduces by a factor of $\frac{3}{1-2\beta}/\left(2\lceil\frac{3}{1-2\beta}\rceil\right) \geq 2$. And hence, Φ reduces by at least a constant amount. We amortize the reduction over the $|\Psi|$ calls to the Oracle, to denote a constant reduction in Φ in every zoom-in step.

If the evolver moved q_i while the algorithm was processing the index i, there is a possibility that X_i might move outside of $\mathcal{B}_i^H$ during the zoom-in routine. We take care of this condition in line 16 to initiate the zoom-out process. Since the algorithm possibly increases Φ by a constant amount when switching from zoom-out to zoom-in, we conclude: For every evolver's step, our algorithm might increase Φ by at most a constant amount as well. ◀

▶ **Lemma 7** (Iteration time proportional to Potential). *There exists a constant speed-up factor σ for* TrackByZoom *such that $\Delta^{\langle z\rangle} \in \Theta_\sigma\left(n + \Phi^{\langle z\rangle}\right)$.*

Proof. By Lemma 4, the evolver can only increase the potential by $\Phi_\varepsilon^{\langle z\rangle} = O(\Delta^{\langle z\rangle})$ during the iteration. By Lemma 6 we observe that the algorithm increases Φ by some amount that is $O(n + \Delta^{\langle z\rangle})$. Since by Lemma 10, the algorithm reduces each Φ_i to at most ϕ_0, it reduces the overall potential by at most $\Phi^{\langle z\rangle} - n\phi_0 + \Phi_\varepsilon^{\langle z\rangle} + O(n + \Delta^{\langle z\rangle}) = O(n + \Phi^{\langle z\rangle} + \Delta^{\langle z\rangle})$. For a speed-up factor σ, the algorithm TrackByZoom spends $O(n + \Phi^{\langle z\rangle} + \Delta^{\langle z\rangle})/\sigma$ time to do so, implying $\Delta^{\langle z\rangle} \leq O(n + \Phi^{\langle z\rangle} + \Delta^{\langle z\rangle})/\sigma$, further implying $\Delta^{\langle z\rangle} \in O_\sigma(n + \Phi^{\langle z\rangle})$, for a sufficiently large constant σ.

To see why $\Delta^{\langle z\rangle} \in \Omega(n + \Phi^{\langle z\rangle})$, we follow a similar logic, except to observe that the evolver can decrease the potential by at most $\Phi_\varepsilon^{\langle z\rangle} = O(\Delta^{\langle z\rangle})$ as well. By Lemma 10, the algorithm reduces the overall potential Φ by at least $\Phi^{\langle z\rangle} - n\phi_0 - \Phi_\varepsilon^{\langle z\rangle} = \Omega(n + \Phi^{\langle z\rangle} - \Delta^{\langle z\rangle})$, which takes it at least $\Omega(n + \Phi^{\langle z\rangle} - \Delta^{\langle z\rangle})/\sigma$ time, implying that $\Delta^{\langle z\rangle} \geq \Omega(n + \Phi^{\langle z\rangle} - \Delta^{\langle z\rangle})/\sigma$, resulting in $\Delta^{\langle z\rangle} \in \Omega_\sigma(n + \Phi^{\langle z\rangle})$. ◀

On Planar Straight-Line Dominance Drawings

Patrizio Angelini
John Cabot University, Rome, Italy

Michael A. Bekos
University of Ioannina, Greece

Giuseppe Di Battista
Università degli Studi Roma Tre, Rome, Italy

Fabrizio Frati
Università degli Studi Roma Tre, Rome, Italy

Luca Grilli
Università degli Studi di Perugia, Italy

Giacomo Ortali
Università degli Studi di Perugia, Italy

Abstract

We study the following question, which has been considered since the 90's: Does every *st*-planar graph admit a planar straight-line dominance drawing? We show concrete evidence for the difficulty of this question, by proving that, unlike upward planar straight-line drawings, planar straight-line dominance drawings with prescribed y-coordinates do not always exist and planar straight-line dominance drawings cannot always be constructed via a contract-draw-expand inductive approach. We also show several classes of *st*-planar graphs that always admit a planar straight-line dominance drawing. These include *st*-planar 3-trees in which every stacking operation introduces two edges incoming into the new vertex, *st*-planar graphs in which every vertex is adjacent to the sink, and *st*-planar graphs in which no face has the left boundary that is a single edge.

2012 ACM Subject Classification Theory of computation → Computational geometry; Mathematics of computing → Graph algorithms

Keywords and phrases *st*-graphs, dominance drawings, planar straight-line drawings, upward planarity

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.5

Funding *Giuseppe Di Battista*: Supported by the European Union, Next Generation EU, Mission 4, Component 1, CUP C53D23003680006 PRIN project no. 2022TS4Y3N "EXPAND: scalable algorithms for EXPloratory Analyses of heterogeneous and dynamic Networked Data".
Fabrizio Frati: Supported by the European Union, Next Generation EU, Mission 4, Component 1, CUP J53D23007130006 PRIN proj. 2022ME9Z78 "NextGRAAL: Next-generation algorithms for constrained GRAph visuALization".
Luca Grilli: Supported by MUR PNRR project SERICS (COVERT: CUP_J93C23002310006), funded by the European Union – Next Generation EU.

Acknowledgements This research initiated at the First Summer Workshop on Graph Drawing (SWGD 2021). Thanks to the organizers and the participants for an inspiring atmosphere.

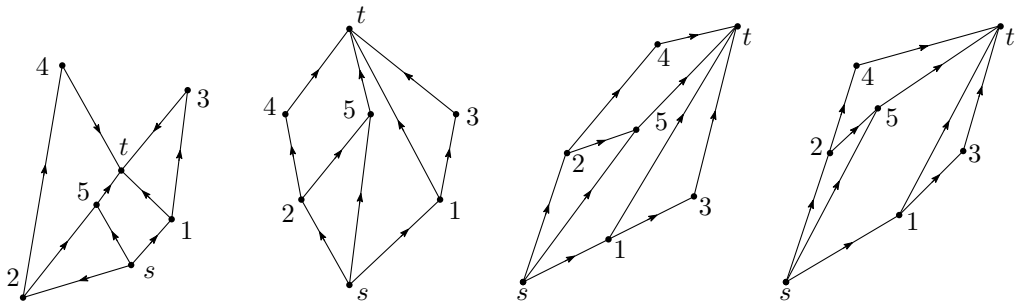
1 Introduction

Drawings of directed graphs are an evergreen research topic in the graph drawing literature. Early papers on the subject go back to the 80's [13, 14, 34] and the number of papers on the topic published since 2023 is in double digits [1, 2, 3, 4, 6, 7, 9, 10, 11, 12, 16, 20, 22, 23, 24, 25, 28, 30]. From an applicative perspective, many domains require techniques for visualizing

© Patrizio Angelini, Michael A. Bekos, Giuseppe Di Battista, Fabrizio Frati, Luca Grilli, and Giacomo Ortali;
licensed under Creative Commons License CC-BY 4.0

directed graphs, such as visualization tools for biological networks and SIEM systems for cyber threat intelligence. Many standards for drawing directed graphs have been defined, and in most of them the drawing is *upward*, i.e., each edge is represented by a Jordan arc whose y-coordinates monotonically increase from the tail to the head of the edge. Di Battista and Tamassia [14] proved that every *upward planar graph* (that is, a directed graph that admits an upward planar drawing) admits an upward planar *straight-line* drawing, a result analogous to Fary's celebrated result about the geometric realizability of planar graphs [19]. In order to prove the geometric realizability of upward planar graphs, it suffices to look at upward planar graphs whose faces are delimited by 3-cycles. Indeed, every upward planar graph is a subgraph of an *st-planar graph* [14] (that is, an upward planar graph with a single source s and a single sink t), which in turn is a subgraph of a *maximal st-planar graph* [14] (that is, an st-planar graph to which no edge can be added without losing simplicity or upward planarity).

One of the easiest algorithms, if not *the* easiest algorithm, for constructing upward planar straight-line drawings is due to Di Battista, Tamassia and Tollis [15]. This algorithm assigns x- and y-coordinates to the vertices simply by performing two pre-order traversals of the input st-planar graph. Moreover, the algorithm constructs upward planar straight-line drawings that are actually *dominance drawings*. These are *xy-monotone drawings* (that is, each edge is represented by a Jordan arc whose x- and y-coordinates monotonically increase from the tail to the head of the edge) such that, for any pair of vertices u, v, there exists a directed path from u to v in the graph if and only if $x(u) \leq x(v)$ and $y(u) \leq y(v)$ hold in the drawing. Dominance drawings constitute an interesting graph drawing style because they express the reachability between vertices by their dominance relationship, i.e., by the coordinates assigned to them; this allows one to answer reachability queries in constant time, see, e.g, [27, 35]. For more about dominance drawings, see, e.g., [5, 8, 18, 26, 31, 32, 34]. Figure 1 shows planar straight-line drawings of an st-planar graph that are non-upward, upward (and not xy-monotone), xy-monotone (and not dominance), and dominance.



(a) Non-upward. **(b)** Upward. **(c)** xy-monotone. **(d)** Dominance.

Figure 1 Four planar straight-line drawings of an st-planar graph G. (a) A non-upward drawing. (b) An upward drawing. (c) An xy-monotone drawing. (d) A dominance drawing.

Di Battista, Tamassia and Tollis's algorithm [15] does not actually construct an upward planar straight-line drawing of every st-planar graph. Indeed, it may construct a non-planar drawing if the input st-planar graph contains *transitive* edges, where an edge is transitive if the graph contains a directed path from the tail to the head of the edge. By subdividing each transitive edge with a new vertex, their algorithm constructs a planar dominance drawing of any st-planar graph in which each edge is either a straight-line segment (if it is non-transitive) or a 1-bend polyline (if it is transitive). Whether this bend per edge can be eliminated by designing an algorithm different from the one in [15] is the question we study in this paper.

Formally, we ask: Does every *st*-planar graph admit a planar straight-line dominance drawing? Apart from the *st*-planar graphs without transitive edges, the question is known to have a positive answer for *series-parallel digraphs* [8]. We prove the following results.

- In Section 3, we prove a remarkable difference between dominance and upward drawings. We revisit the two main approaches for the construction of upward planar straight-line drawings of *st*-planar graphs and prove that they cannot be successfully applied to construct planar straight-line dominance drawings. The first approach [14] contracts an internal edge of the graph, constructs a drawing inductively, and then expands the previously contracted edge to be a "short" segment. We show that there exist *st*-planar graphs in which no edge can be used in the contract-draw-expand approach so to get a planar straight-line dominance drawing. The second method [17, 21, 33] prescribes the y-coordinates of the vertices, so that the tail of any edge is assigned a smaller y-coordinate than its head. This additional constraint on the drawing allows for easier recursive schemes for its construction. We prove that planar straight-line dominance drawings with prescribed y-coordinates do not always exist. We believe that these results provide solid evidence for the difficulty of constructing planar straight-line dominance drawings.
- In Section 4, we study *st*-planar graphs whose underlying graph is a planar 3-tree. Planar 3-trees, also known as stacked triangulations and Apollonian networks, constitute a common benchmark for planar graph drawing problems, as they allow for easy inductive constructions; for example, every planar 3-tree with at least four vertices can be constructed by "stacking" a vertex inside a face of a smaller planar 3-tree. For our question, the study of *st*-planar 3-trees turns out to be complicated, as inductive drawing constructions do not cope well with the dominance relationship that needs to be ensured between vertices that are "far away" in the graph. We show how to construct planar straight-line dominance drawings for two classes of *st*-planar 3-trees, the first one with a constraint on the orientation (every stacking operation introduces two edges incoming into the new vertex) and the second one with a constraint on the graph structure (every stacking operation happens in a face incident to the sink). The latter graph class coincides with the maximal *st*-planar graphs in which the sink is adjacent to every vertex.
- In Section 5, we improve the mentioned result by Di Battista, Tamassia and Tollis [15], by proving that a planar straight-line dominance drawing always exists for any *st*-planar graph in which every transitive edge is to the right of every directed path from the tail to the head of the edge. This result is obtained via an ear decomposition of the graph. This shows that the problem of constructing planar straight-line dominance drawings is made difficult by the interaction between "left transitive edges" and "right transitive edges".

All our algorithms construct drawings whose resolution is exponentially small (or worse). This drawback is sometimes necessary for upward planar straight-line drawings [15], and hence also for planar straight-line dominance drawings. However, for the graph classes we considered, we do not know whether an exponentially-small resolution is actually required in order to construct planar straight-line dominance drawings.

2 Preliminaries

A *drawing* of a graph maps each vertex to a distinct point in the plane and each edge to a Jordan arc between its endpoints. A drawing is *straight-line* if each edge is represented by a straight-line segment and *planar* if no two edges intersect, except at common endpoints. Two planar drawings of a connected graph are *plane-equivalent* if they define the same clockwise order of the edges incident to each vertex and the same clockwise order of the vertices and

edges along the boundary of the outer face. A *plane embedding* is an equivalence class of planar drawings and a *plane graph* is a graph with a plane embedding. Whenever we talk about planar drawings of a plane graph, we always assume that they are in the equivalence class associated with the plane graph. An *st-plane graph* is an *st*-planar graph with a plane embedding (for its underlying graph) in which s and t are incident to the outer face. An *st*-plane graph is *maximal* if no edge can be added to it while maintaining it an *st*-plane graph. Since every *st*-planar graph can be augmented (by adding vertices and edges) to maximal without altering the reachability between vertices [14], the existence of a planar straight-line dominance drawing for all *st*-planar graphs can be decided by only looking at maximal *st*-planar graphs. Note that, in any planar straight-line dominance drawing of an *st*-planar graph, the vertex placements can be perturbed so that no two vertices share the same x- or the same y-coordinate and so that the drawing remains planar, straight-line and dominance. Hence, throughout the paper, every considered dominance drawing has distinct x- and distinct y-coordinates for its vertices. Two vertices in a directed graph are *incomparable* if no directed path goes from any of the vertices to the other one.

As a warm-up result, we prove that every Hamiltonian *st*-planar graph has a planar straight-line dominance drawing. A directed graph is *Hamiltonian* if it contains a directed path $(v_1 = s, v_2, \ldots, v_n = t)$, where $\{v_1, v_2, \ldots, v_n\}$ is the vertex set of the graph.

▶ **Theorem 1.** *Hamiltonian st-planar graphs admit planar straight-line dominance drawings.*

Proof. Consider a Hamiltonian *st*-planar graph G. Construct an upward planar straight-line drawing Γ of G; this always exists [14]. Stretch Γ vertically, so that the slope of every edge is in the range $(45°, 135°)$. Now rotate Γ in clockwise direction by 45°. Since the slope of every edge is now in the range $(0°, 90°)$, we have that Γ is xy-monotone. Since vertical stretch and rotation are affine transformations, Γ is planar, as well. Finally, since G contains a Hamiltonian path $(v_1, \ldots, v_n)$, vertex v_j is reachable from vertex v_i, for every $1 \leq i < j \leq n$. Since the slope of the edge (v_k, v_{k+1}) is in the range $(0°, 90°)$, for $k = i, \ldots, j-1$, vertex v_j is in the first quadrant of vertex v_i, hence Γ is a dominance drawing. ◀

3 Planar Straight-line Dominance Drawings are Difficult to Get

In this section, we revisit the two main approaches for the construction of upward planar straight-line drawings of *st*-planar graphs and prove that they cannot be enhanced (or at least not in a direct way) to construct planar straight-line dominance drawings.

3.1 Constructing Drawings via Contractions and Expansions

Di Battista and Tamassia [14] first proved that every *st*-plane graph admits an upward planar straight-line drawing. Their proof extends to directed graphs a well-known proof by Fáry [19], showing that every (undirected) plane graph admits a planar straight-line drawing. We briefly describe the algorithm by Di Battista and Tamassia [14].

An internal edge (u, v) of a maximal *st*-plane graph G is *contractible* if it satisfies the following conditions: (1) The vertices u and v have exactly two common neighbors, denoted by z_1 and z_2; note that the cycles (u, v, z_1) and (u, v, z_2) delimit internal faces of G. (2) For $i = 1, 2$, the edges connecting u and v with z_i are both incoming or both outgoing at z_i.

The *contraction* of a contractible edge (u, v) constructs a graph G' by identifying u and v into a vertex w with the following adjacencies (see Fig 2a). For every neighbor $z \notin \{u, v, z_1, z_2\}$ of u (of v), we have that G' contains an edge between w and z, which is outgoing at z if and only if the edge between u and z (resp. between v and z) is outgoing

at z. Also, for $i = 1, 2$, we have that G' contains an edge between w and z_i, which is outgoing at z_i if and only if the edges connecting u and v with z_i are both outgoing at z_i. It is easy to see that G' is a maximal st-plane graph.

The core of Di Battista and Tamassia's algorithm lies in the following two statements[1]: (i) every maximal st-plane graph G has a contractible edge (u, v), whose contraction results in a maximal st-plane graph G'; and (ii) an upward planar straight-line drawing Γ of G can be obtained from an upward planar straight-line drawing Γ' of G' by *expanding* w, that is, by replacing w with a sufficiently small segment (with a suitable slope) representing (u, v).

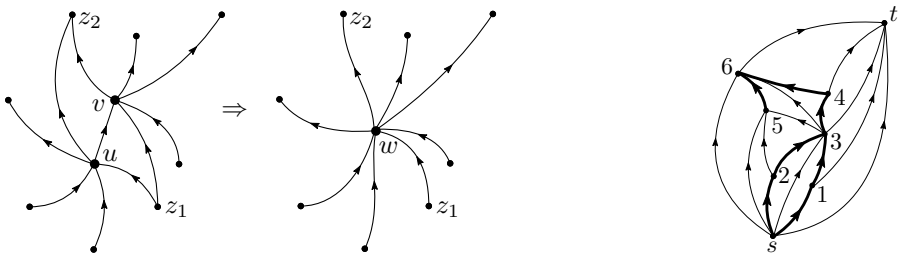


Figure 2 (a) The contraction of an edge (u, v) in a maximal st-plane graph. (b) A maximal st-plane graph with no dominance-expandable edge. Thin edges are not contractible, while fat edges are contractible but not dominance-expandable; for example, $(1, 3)$ is not dominance-expandable, because vertex 2 is a predecessor of vertex 3 but not a predecessor of vertex 1.

Since, depending on the geometric placement of the neighbors of w in Γ', the edge (u, v) might need to be an arbitrarily small segment in Γ, in order for Γ to be a dominance drawing we need u and v to have the same successors and predecessors. That is, let $\mathcal{S}(z)$ be the set of successors of a vertex z, that is, the set of all vertices z' such that there exists a directed path from z to z'. Analogously, let $\mathcal{P}(z)$ be the sets of predecessors of a vertex z. A contractible edge (u, v) is *dominance-expandable* if $\mathcal{S}(u) = \mathcal{S}(v) \cup \{v\}$ and $\mathcal{P}(v) = \mathcal{P}(u) \cup \{u\}$. Di Battista and Tamassia's approach could be enhanced to construct planar straight-line dominance drawings if every maximal st-plane graph contained a dominance-expandable edge. However, we can prove that there exist maximal st-plane graphs with no dominance-expandable edge, as the one in Fig 2b, which constitutes a barrier for this approach we cannot overcome.

We remark that, for every graph class for which we can prove the existence of planar straight-line dominance drawings in the upcoming sections, there exist graphs in the class that do not have a dominance-expandable edge or such that the contraction of any dominance-expandable edge would result in a graph not in the same class.

3.2 Constructing Drawings by Prescribing the y-Coordinates

Eades, Feg, Lin, and Nagamochi [17] and, independently, Pach and Tóth [33] proved that every upward planar drawing can be straightened while preserving the y-coordinates of the vertices. This implies that every st-plane graph admits an upward planar straight-line drawing with prescribed y-coordinates (as long as these respect the reachability between

[1] Di Battista and Tamassia's proof actually distinguishes the case in which G contains a *separating triangle* (a 3-cycle with vertices in its interior) from the case in which it does not, performing different constructions in the two cases. However, the first case is unnecessary, as a contractible edge in an st-planar graph can always be found, similarly to what was noted by Wood [36] for undirected graphs.

vertices). This result was strengthened by Hong and Nagamochi [21], who proved that every internally-triconnected st-plane graph admits an upward planar straight-line *convex* drawing with prescribed y-coordinates and prescribed outer face. It is interesting that, while more constrained, drawings with prescribed y-coordinates (and a prescribed outer face) allow for an easier recursive construction.

We now show that, unlike upward planar straight-line drawings, planar straight-line dominance drawings with given y-coordinates do not always exist.

▶ **Theorem 2.** *For every $n \geq 7$, there exists an st-planar graph G_n with vertex set $\{v_1, v_2, \ldots, v_n\}$ such that:*

- *there exists a planar dominance drawing of G_n such that $y(v_i) = i$, for $i = 1, \ldots, n$; and*
- *there exists no planar straight-line dominance drawing of G_n such that $y(v_i) = i$, for $i = 1, \ldots, n$.*

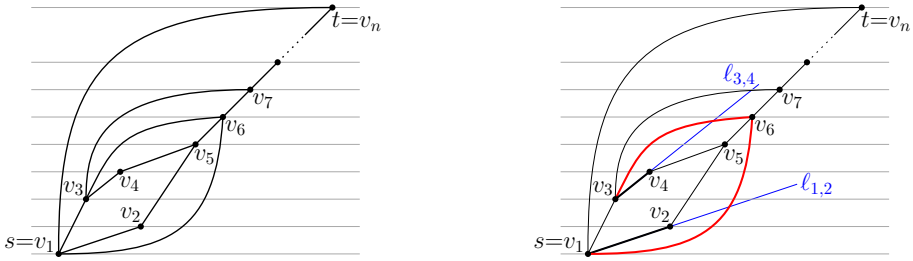


Figure 3 (a) The graph for the proof of Theorem 2. (b) The rays $\ell_{1,2}$ and $\ell_{3,4}$ diverge.

Proof. The st-planar graph G_n consists of the directed paths $(s = v_1, v_2, v_5)$, (v_1, v_3, v_4, v_5), $(v_5, v_6, \ldots, v_n = t)$, and of the edges (v_1, v_6), (v_3, v_6), (v_3, v_7), and (v_1, v_n). Fig 3a shows a planar dominance drawing of G_n with $y(v_i) = i$, for $i = 1, \ldots, n$. Suppose, for a contradiction, that a planar straight-line dominance drawing Γ of G_n exists with $y(v_i) = i$, for $i = 1, \ldots, n$. We prove that the plane embedding in Γ of the underlying graph of G_n is the one in Fig 3a, except, possibly, for the position of the edge (s, t). Obviously, the path $(v_1, v_2, v_5, v_6, \ldots, v_n)$ has a unique plane embedding. Since v_2 and v_4 are incomparable and $y(v_2) < y(v_4)$, we have $x(v_4) \leq x(v_2)$, hence the clockwise order of the vertices along the cycle $\mathcal{C} := (v_1, v_3, v_4, v_5, v_2)$ is v_1, v_3, v_4, v_5, v_2. From that, we get that the edges (v_3, v_6) and (v_3, v_7) lie above the path $(v_3, v_4, v_5, v_6, v_7)$, and finally that the edge (v_1, v_6) lies below the path (v_1, v_2, v_5, v_6). For any distinct $i, j \in \{1, \ldots, n\}$, let $\ell_{i,j}$ be the ray starting at v_i and passing through v_j. Since $x(v_1) < x(v_3) < x(v_4) \leq x(v_2)$, we have $x(v_2) - x(v_1) > x(v_4) - x(v_3)$. Also, we have $y(v_2) - y(v_1) = y(v_4) - y(v_3) = 1$. Hence, the ray $\ell_{1,2}$ has smaller slope than the ray $\ell_{3,4}$; that is, such rays diverge, see Fig 3b. Since the ray $\ell_{1,6}$ has smaller slope than $\ell_{1,2}$, and since the ray $\ell_{3,6}$ has larger slope than $\ell_{3,4}$, it follows that $\ell_{1,6}$ and $\ell_{3,6}$ also diverge, while they meet at v_6, a contradiction which proves the theorem. Note that vertices $v_8, \ldots, v_n$ only serve the purpose of creating an infinite graph family. ◀

We can similarly show that one cannot, in general, prescribe the x-coordinates of a planar straight-line dominance drawing.

Also, we can strengthen Theorem 2 by proving that, for every $n \geq 10$ and for every sequence $y_1 < \cdots < y_n$ of y-coordinates, there exists an st-planar graph G'_n with vertex set $\{v_1, \ldots, v_n\}$ such that there exists a planar dominance drawing of G'_n with $y(v_i) = y_i$,

for $i = 1, \ldots, n$, and there exists no planar straight-line dominance drawing of G'_n with $y(v_i) = y_i$, for $i = 1, \ldots, n$. That is, the y-coordinates prescribed by Theorem 2 do not need to be uniformly distributed.

The key point for this is the observation that the proof of Theorem 2 works as long as $y(v_2) - y(v_1) \leq y(v_4) - y(v_3)$. Hence, we can consider the four lines $y = y_i$, with $i = 4, 5, 6, 7$, and then distinguish two cases. If $y_5 - y_4 \leq y_7 - y_6$, we let our st-planar graph G'_n contain the graph G_7 from the proof of Theorem 2 and we set $y(v_i) = y_{i+3}$, for $i = 1, \ldots, 7$, where $v_1, \ldots, v_7$ is the vertex set of G_7. Otherwise, that is, if $y_7 - y_6 < y_5 - y_4$, we let our st-planar graph G'_n contain the graph obtained by reversing the edge directions of the graph G_7 and we set $y(v_i) = y_{8-i}$, for $i = 1, \ldots, 7$, where $v_1, \ldots, v_7$ is the vertex set of G_7.

4 *st*-plane 3-trees

A *plane 3-tree* is a plane graph recursively defined as follows. A 3-cycle embedded in the plane is a plane 3-tree. Any plane 3-tree with $n \geq 4$ vertices can be obtained from a plane 3-tree with $n-1$ vertices by *stacking* a new vertex into an internal face, that is, by connecting the new vertex to the three vertices incident to the face. An *st-plane 3-tree* is an st-plane graph whose underlying graph is a plane 3-tree. In our opinion, st-plane 3-trees constitute a very challenging class of st-plane graphs for our problem. Indeed, the "natural" strategies for drawing the graphs in this class are to either recursively construct and then combine the drawings of three smaller st-plane 3-trees, or to iteratively add a single vertex to a previously constructed drawing of a smaller st-plane 3-tree; both these strategies do not cope well with the geometric relationship that has to be ensured for incomparable vertices. Nevertheless, in this section we show how to obtain planar straight-line dominance drawings of two classes of st-plane 3-trees.

4.1 Upper *st*-plane 3-trees

Consider the construction of an st-plane 3-tree G via repeated stacking operations. If a vertex u is stacked into a face delimited by a cycle (a, b, c), where a and c are the source and the sink of the cycle, respectively, then the edge (a, u) is directed from a to u, the edge (u, c) is directed from u to c, while the edge (b, u) might be directed either way. We say that G is an *upper st-plane 3-tree* if, at every stacking operation, the edge that can be directed either way is always directed towards the newly inserted vertex. We have the following.

▶ **Theorem 3.** *Upper st-plane 3-trees admit planar straight-line dominance drawings.*

Proof. Let G be an n-vertex upper st-plane 3-tree whose outer face is delimited by the cycle (s, m, t). Let Δ be any triangle with vertices p_s, p_m, p_t, where $x(p_s) < x(p_m) < x(p_t)$ and $y(p_s) < y(p_m) < y(p_t)$. Also, let D be a closed disk in the interior of Δ such that, for any point p in D, we have $x(p_m) < x(p) < x(p_t)$ and $y(p_m) < y(p) < y(p_t)$; see Figs 4a and 4c. We prove by induction that G admits a planar straight-line dominance drawing such that:

- s lies at p_s, m lies at p_m, and t lies at p_t; and
- every internal vertex of G lies in the interior of D.

The statement clearly implies the theorem. In the base case, in which $n = 3$, the triangle Δ is the required drawing of G and the statement is trivially true.

Suppose now that $n > 3$. Let r be the first stacked vertex in the construction of G; that is, r is the unique vertex of G adjacent to s, m, and t. Note that the edge (m, r) is directed away from m, given that G is an upper st-plane 3-tree. Let G_1, G_2, and G_3 be the subgraphs

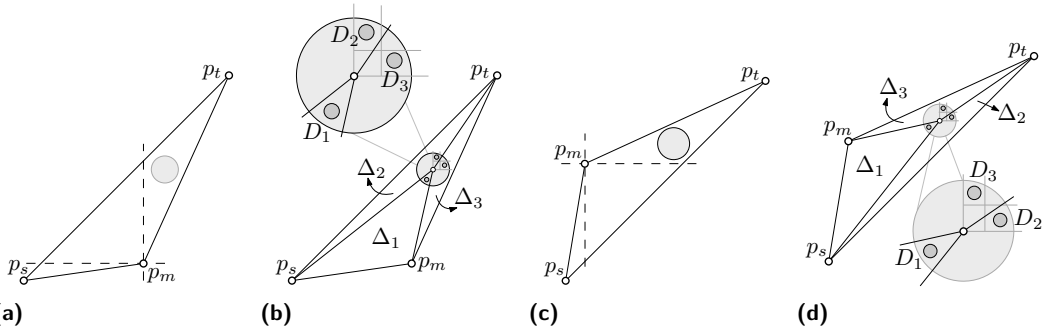


Figure 4 (a) and (c) Triangle Δ and disk D for the input to the induction. (b) and (d) Placing point p_r (white) and disks D_1, D_2, and D_3 (gray) inside D; an enlarged view of the placement of r and of disks D_1, D_2, and D_3 inside D is also shown.

of G inside the cycles (s, m, r), (s, r, t), and (m, r, t), respectively. Note that G_1 is an upper sr-plane 3-tree, G_2 is an upper st-plane 3-tree, and G_3 is an upper mt-plane 3-tree. Also, each of G_1, G_2, and G_3 has less than n vertices. Let p_r be any point inside D and let Δ_1, Δ_2, and Δ_3 be the triangles (p_s, p_m, p_r), (p_s, p_r, p_t), and (p_m, p_r, p_t), respectively. Place r at p_r; by the properties of D, we have $x(p_m) < x(p_r)$ and $y(p_m) < y(p_r)$, which complies with the orientation of (m, r). Let D_1, D_2, and D_3 be closed disks such that (see Figs 4b and 4d):

- disk D_1 lies in the interior of $\Delta_1 \cap D$, disk D_2 lies in the interior of $\Delta_2 \cap D$, and disk D_3 lies in the interior of $\Delta_3 \cap D$;
- for any point $p \in D_2 \cup D_3$, we have $x(p_r) < x(p)$ and $y(p_r) < y(p)$; and
- for any point $p_2 \in D_2$ and any point $p_3 \in D_3$, if the clockwise order of the vertices of Δ is p_s, p_t, p_m, then we have $x(p_2) < x(p_3)$ and $y(p_3) < y(p_2)$, otherwise we have $y(p_2) < y(p_3)$ and $x(p_3) < x(p_2)$.

Clearly, disks D_1, D_2, and D_3 with the above properties always exist. By induction, G_1, G_2, and G_3 have planar straight-line dominance drawings Γ_1, Γ_2, and Γ_3 with s, m, r, and t drawn at p_s, p_m, p_r, and p_t, respectively, so that the internal vertices of G_1, G_2, and G_3 lie in the interior of D_1, D_2, and D_3, respectively. This results in a straight-line drawing Γ of G.

Since p_r, D_1, D_2, and D_3 lie in the interior of D, all the internal vertices of G lie in the interior of D, as required. The upward planarity of Γ follows from the ones of Γ_1, Γ_2, and Γ_3. In order to prove that Γ is a dominance drawing, consider any pair of vertices u and v.

- If u and v belong to the same graph G_i, for some $i \in \{1, 2, 3\}$, then their placement complies with their dominance relationship, by induction.
- If one of u and v is s, say $u = s$, then u is a predecessor of v, and indeed we have $x(u) < x(v)$ and $y(u) < y(v)$. The case $u = t$ can be discussed similarly.
- If neither of u and v is s or t, and one of u and v is m, say $u = m$, then u is a predecessor of v, since G is an upper st-plane 3-tree. Since $x(p_m) < x(p)$ and $y(p_m) < y(p)$, for any point $p \in D$, we have $x(u) < x(v)$ and $y(u) < y(v)$.
- If u is an internal vertex of G_1 and v is an internal vertex of G_2 or G_3, then u is a predecessor of v, since G is an upper st-plane 3-tree. Since, for any point $p \in D_1$ and any point $q \in D_2 \cup D_3$, we have $x(p) < x(p_r) < x(q)$ and $y(p) < x(y_r) < y(q)$, the placement of u and v complies with their dominance relationship.
- Finally, if u is an internal vertex of G_2 and v is an internal vertex of G_3, then u and v are incomparable, since G is an upper st-plane 3-tree. Since, for any point $p_2 \in D_2$ and any point $p_3 \in D_3$, we have $x(p_2) < x(p_3)$ and $y(p_3) < y(p_2)$, or $y(p_2) < y(p_3)$ and $x(p_3) < x(p_2)$, the placement of u and v complies with their dominance relationship.

This completes the induction and the proof of the theorem. ◀

An analogous result holds true for st-plane 3-trees such that, at every stacking operation, the edge that can be directed either way is always directed out of the newly inserted vertex.

Trying to use a similar strategy in order to construct a planar straight-line dominance drawing of every st-plane 3-tree might be tempting. However, the "types" of internal vertices in a general st-plane 3-tree are more than three. Namely, referring to the notation introduced in the proof of the theorem, the internal vertices of G_2 and G_3 are not all successors of r, but rather some are predecessors, some are successors, and some are incomparable to r. Hence, the "three-disks schema" fails, and more complex geometric invariants seem to be needed.

4.2 Sink-dominant st-plane 3-trees

We next look at the st-plane 3-trees in which every stacking operation happens in a face incident to the sink t of the graph. This results in an st-plane 3-tree in which the sink is adjacent to every vertex. We call this a *sink-dominant st-plane 3-tree*. It is easy to observe that every n-vertex maximal st-plane graph in which the sink has degree $n-1$ is a sink-dominant st-plane 3-tree (and vice versa). We have the following.

▶ **Theorem 4.** *Sink-dominant st-plane 3-trees admit planar straight-line dominance drawings.*

Proof. Let G be an n-vertex sink-dominant st-plane 3-tree whose outer face is delimited by the cycle (s, m, t).

Assumption. If $n > 3$, then let r be the internal vertex of G adjacent to s, m and t. If r is a predecessor of m, as in Fig 5a, we add a new source s' adjacent to s, m and t in the outer face of G, so that the outer face of the resulting graph G' is delimited by the 3-cycle (s', s, t). Now m is the internal vertex of G' adjacent to s', s and t; furthermore, m is a successor of s. Hence, by possibly adding a vertex and three edges to G and changing some labels, we can assume w.l.o.g. that the internal vertex r that is adjacent to the three vertices s, m and t incident to the outer face of our input st-plane 3-tree G is a successor of m.

Inductive hypothesis. The proof is similar in spirit to, however more involved than, the proof of Theorem 3. Let Δ be any triangle with vertices p_s, p_m and p_t, where $x(p_s) < x(p_m) < x(p_t)$ and $y(p_s) < y(p_m) < y(p_t)$. If p_m lies above the line through p_s and p_t, we say that Δ is of *type A* (see Fig 5b), otherwise it is of *type B* (see Fig 5c). Let D and E be closed disks contained in the interior of Δ such that:

- if Δ is of type A, then D and E are horizontally aligned, that is, they have the same two vertical tangents, while if Δ is of type B, then D and E are vertically aligned;
- if Δ is of type A, then D is strictly below and to the right of p_m, while if Δ is of type B, then D is strictly above and to the left of p_m; and
- E is strictly above and to the right of p_m.

We prove, by induction on n, that G admits a planar straight-line dominance drawing such that:

- s lies at p_s, m lies at p_m, and t lies at p_t;
- every internal vertex of G that is a successor of m lies in the interior of E; and
- every internal vertex of G that is incomparable to m lies in the interior of D.

Note that, because of the assumption that r is a successor of m, no internal vertex of G is a predecessor of m.

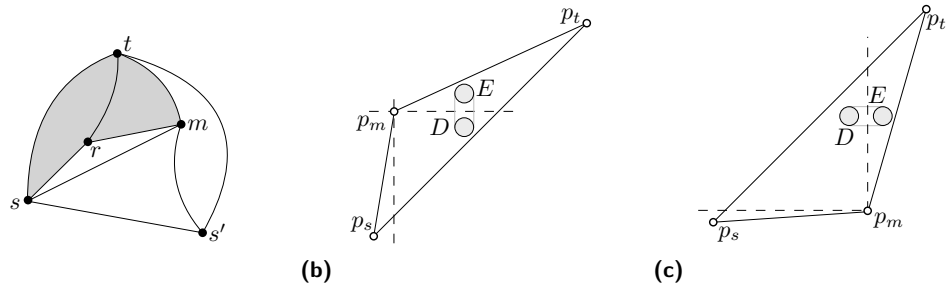


Figure 5 (a) Augmenting G so that the vertex adjacent to the three vertices on the outer face is a successor of two of them. (b) and (c) Triangle Δ and disks D and E for the input to the induction. In (b) Δ is of type A, while in (c) it is of type B.

The statement clearly implies the theorem. In the base case, in which $n = 3$, the triangle Δ is the required drawing of G and the statement is true. Suppose now that $n > 3$. We only show the construction for the case in which Δ is of Type B, as the other case is analogous.

Graph structure. Recall that r is the unique vertex of G adjacent to s, m and t, and that the edge (m, r) is directed towards r; refer to Fig 6.

Let $P_m := (v_0 = m, v_1, \ldots, v_\ell = r)$ be the longest directed path from m to r. Since every vertex is adjacent to t and since r is a successor of m, we have that P_m exists and is unique. For $j = 1, \ldots, \ell$, let M_j be the subgraph of G induced by the vertices inside or on the boundary of the 3-cycle (v_{j-1}, v_j, t) and note that M_j is a sink-dominant st-plane 3-tree. By the fact that P_m is the longest directed path from m to r, we have that, if M_j contains internal vertices, then the internal vertex of M_j that is adjacent to v_{j-1}, v_j and t is a successor of v_j. This implies that M_j can be drawn recursively.

Also, let $P_s := (u_0 = s, u_1, \ldots, u_k = r)$ be the longest directed path from s to r in G that does not pass through m. For $i = 1, \ldots, k$, the subgraph S_i of G induced by the vertices inside or on the boundary of the 3-cycle (u_{i-1}, u_i, t) is a sink-dominant st-plane 3-tree. Further, if S_i contains internal vertices, then the internal vertex of S_i that is adjacent to u_{i-1}, u_i and t is a successor of u_i, hence S_i can be drawn recursively.

Since every vertex of G is adjacent to t, the interior of the cycle $\mathcal{C}_{sm} := P_s \cup P_m \cup (s, m)$ does not contain any vertices, while it might contain some edges (and in fact it does, unless $\mathcal{C}_{sm} = (s, m, r)$). We are going to draw $\mathcal{C}_{sm}$ as a convex curve (hence the edges in its interior will not cause crossings).

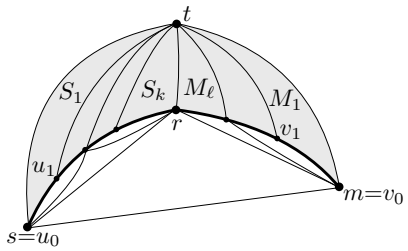


Figure 6 Paths P_m and P_s (as thick lines) and graphs $M_1, \ldots, M_\ell, S_1, \ldots, S_k$ (with gray interior).

Construction. We now draw P_s and P_m. We also draw disks inside the triangles representing the cycles (u_{i-1}, u_i, t), for $i = 1, \ldots, k$, and (v_{j-1}, v_j, t), for $j = 1, \ldots, \ell$, so that induction can be applied in order to draw the subgraphs S_i and M_j recursively. Refer to Fig 7 for an illustration of the relative placement of the vertices of P_s and P_m and of the desired disks.

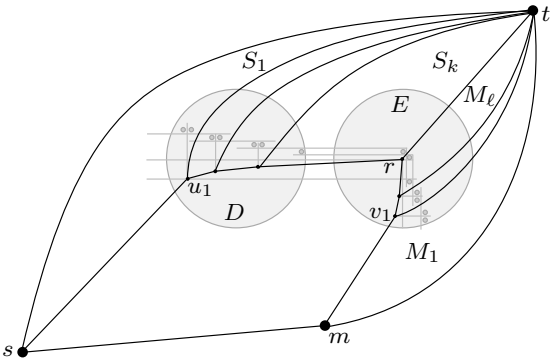


Figure 7 Drawing paths P_s and P_m, disks $D_1^u, E_1^u, \ldots, D_k^u, E_k^u$, and disks $D_1^v, E_1^v, \ldots, D_\ell^v, E_\ell^v$. For the sake of readability, some edges are drawn as curves, as the illustration is mainly meant to represent the relative placement of the vertices of the paths P_s and P_m and of the listed disks.

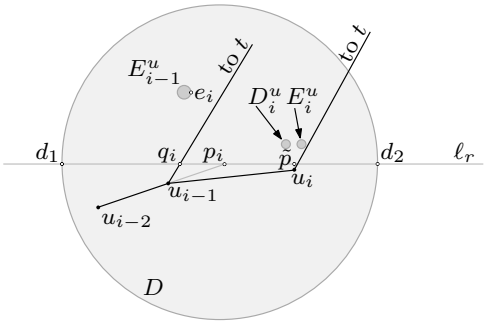


Figure 8 Drawing vertex u_i.

We start by placing r at the center of the disk E. Next, we draw the vertices $u_1, \ldots, u_{k-1}$ in this order inside D. Let σ_r be the intersection of the horizontal line ℓ_r through r with D, and let d_1 and d_2 be the leftmost and rightmost endpoints of σ_r, respectively. For $i = 1, \ldots, k-1$, by drawing u_i, we complete the drawing of the triangle Δ_i^u representing cycle (u_{i-1}, u_i, t). Then we also place suitable disks D_i^u and E_i^u inside Δ_i^u so that S_i can be drawn recursively.

When we have to draw u_i, for some $i \in \{1, \ldots, k-1\}$, we assume that (see Fig 8):

- (C1) the polygonal line $(u_0, \ldots, u_{i-1}, r)$ is convex and lies below ℓ_r;
- (C2) if $i > 1$, the line through u_{i-2} and u_{i-1} cuts σ_r in its interior, at a point p_i;
- (C3) if $i > 1$, the segment between u_{i-1} and t cuts σ_r in its interior, at a point q_i; and
- (C4) if $i > 1$, the disks D_{i-1}^u and E_{i-1}^u lie inside D and above ℓ_r.

We denote by e_i be the rightmost point of E_{i-1}^u. Note that conditions (C1)–(C4) are vacuous if $i = 1$ (i.e., before drawing u_1). In that case, for the sake of simplicity of the description, we let p_i, q_i, and e_i coincide with d_1.

We now explain how to draw u_i. Let $\overline{x} = \max\{x(p_i), x(q_i), x(e_i)\}$, let $\tilde{x} = (x(d_2) + \overline{x})/2$, where $\overline{x} < \tilde{x} < x(d_2)$, and let $\tilde{p}$ be the point of σ_r with $x(\tilde{p}) = \tilde{x}$. We place u_i at $(\tilde{x}, y(r) - \epsilon)$, where $\epsilon > 0$ is sufficiently small so that conditions (C1)–(C3) are satisfied when we have to draw u_{i+1}. Indeed, if $\epsilon = 0$, then u_i would be placed at $\tilde{p}$ and conditions (C1)–(C3) would be trivially satisfied when we have to draw u_{i+1}, hence they are also satisfied for some sufficiently small $\epsilon > 0$, by continuity.

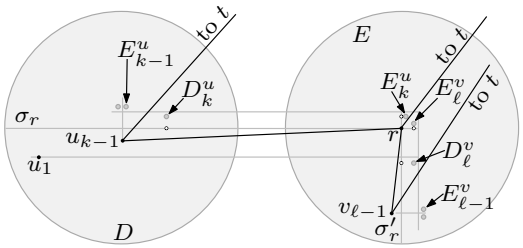


Figure 9 Illustration for the placement of the disks D^u_k, E^u_k, D^v_ℓ, and E^v_ℓ. White circles represent initial or intermediate placements for such disks.

We now place the disks D^u_i and E^u_i so that they have radius δ and centers at $(\tilde{x} \pm \epsilon', y(r) + \epsilon')$, where $\epsilon' > \delta > 0$ are sufficiently small so that:

- D^u_i and E^u_i lie inside the triangle $\Delta^u_i = (u_{i-1}, u_i, t)$;
- D^u_i and E^u_i are lower than D^u_{i-1} and E^u_{i-1}; and
- condition (C4) is satisfied when we have to draw u_{i+1}.

Indeed, if $\epsilon' = \delta = 0$ such disks would degenerate and coincide with $\tilde{p}$, which is inside D and also inside Δ^u_i, as the segment $\overline{u_{i-1}t}$ cuts σ_r at a point q_i to the left of $\tilde{p}$ and the segment $\overline{u_i t}$ cuts σ_r at a point to the right of $\tilde{p}$. Hence, such disks remain inside D and Δ^u_i if $\epsilon' > \delta > 0$ is sufficiently small, by continuity. Since $\epsilon' > \delta$, disks D^u_i and E^u_i lie above ℓ_r, hence ensuring condition (C4). Finally, choosing $\delta + \epsilon'$ smaller than the distance between E^u_{i-1} and ℓ_r ensures that D^u_i and E^u_i are lower than D^u_{i-1} and E^u_{i-1}.

We now draw the vertices $v_1, \ldots, v_{\ell-1}$ in this order. For $j = 1, \ldots, \ell - 1$, when we draw v_j, we have drawn the triangle Δ^v_j representing cycle (v_{j-1}, v_j, t). Then we also show how to place suitable disks D^v_j and E^v_j inside Δ^v_j so that M_j can be drawn recursively. This is done very similarly to the way vertices $u_1, \ldots, u_{k-1}$ and disks $D^u_1, E^u_1, \ldots, D^u_{k-1}, E^u_{k-1}$ were drawn (see again Fig 7), so we only highlight the differences here.

- First, all such vertices and disks lie inside E, rather than inside D.
- Second, the role previously played by σ_r is now played by a segment σ'_r along the vertical line ℓ'_r through r. The endpoints of σ'_r are the lowest intersection point e_1 of ℓ'_r with the boundary of E and the intersection point of ℓ'_r with the horizontal line through u_1. This is because the vertices $v_1, \ldots, v_{\ell-1}$ and the disks $D^v_1, E^v_1, \ldots, D^v_{\ell-1}, E^v_{\ell-1}, D^v_\ell$ have to be placed below (and to the right of) $u_1, \ldots, u_{k-1}$, so to satisfy the constraints of a dominance drawing. When a vertex v_j is drawn, the segment $\overline{v_j t}$ crosses the interior of σ'_r.
- Third, the triangles Δ^v_j are of Type A, unlike the triangles Δ^u_i which are of Type B. Hence, the disks D^v_j and E^v_j are horizontally aligned, to the right of σ'_r. The disks D^v_j and E^v_j are above and to the left of the disks D^v_{j-1} and E^v_{j-1}.

After the vertices $v_1, \ldots, v_{\ell-1}$ and the disks $D^v_1, E^v_1, \ldots, D^v_{\ell-1}, E^v_{\ell-1}$ have been drawn, it only remains to draw the disks D^u_k and E^u_k inside $\Delta^u_k = (u_{k-1}, u_k, t)$ and the disks D^v_ℓ and E^v_ℓ inside $\Delta^v_\ell = (v_{\ell-1}, v_\ell, t)$. See Fig 9. We have to place D^u_k and E^u_k above σ_r and below E^u_{k-1}, with D^u_k in D and E^u_k in E; also, E^u_k has to be to the right of r. Analogously, we have to place D^v_ℓ and E^v_ℓ in E, to the right of σ'_r and to the left of $E^v_{\ell-1}$, with E^v_ℓ above r. Finally, E^u_k has to be above and to the left of E^v_ℓ.

We can again use continuity arguments to prove that such disk placements exist. Indeed, $\overline{u_{k-1}t}$ cuts the interior of σ_r, hence D^u_k can be initially set to be a point in the interior of σ_r, to the right of $\overline{u_{k-1}t}$. Analogously, D^v_ℓ can be initially set to be a point in the interior of σ'_r above $\overline{v_{\ell-1}t}$. Disks E^u_k and E^v_ℓ are initially set to coincide with r. Now D^u_k and E^u_k can be moved upward of a sufficiently small distance so that D^u_k does not collide with $\overline{u_{k-1}t}$ and remains below E^u_{k-1}; note that now D^u_k and E^u_k are in the interior of Δ^u_k. Analogously, disks

D^v_ℓ and E^v_ℓ can be moved rightward, of a sufficiently small distance so that they still are to the left of $E^v_{\ell-1}$ and they now both lie in the interior of Δ^v_ℓ. Next, we move E^u_k rightward and E^v_ℓ upward so that they are to the right and above r, respectively. This movement is sufficiently small so that E^u_k remains in Δ^u_k and E^v_ℓ in Δ^v_ℓ, and so that E^u_k remains above and to the left of E^v_ℓ. Finally, we enlarge the disks so that they have a positive radius. Such a radius can be set to be sufficiently small so that all the above listed properties, which were satisfied before such an enlargement, are still maintained.

The drawing Γ of G is completed by drawing the subgraphs $M_1, \ldots, M_\ell$, $S_1, \ldots, S_k$ recursively, with triangles $\Delta^v_1, \ldots, \Delta^v_\ell, \Delta^u_1, \ldots, \Delta^u_k$ representing their outer faces, and with disks $D^v_1, E^v_1, \ldots, D^v_\ell, E^v_\ell, \ldots, D^u_1, E^u_1, \ldots, D^u_k, E^u_k$ inside such triangles.

Correctness. The drawing Γ is straight-line by construction.

The drawings of the subgraphs $M_1, \ldots, M_\ell, S_1, \ldots, S_k$ are planar by induction. Moreover, the construction guarantees that the cycle $\mathcal{C}_{sm}$ is represented by a convex curve which keeps in its exterior every edge from a vertex of $\mathcal{C}_{sm}$ to t. It follows that distinct subgraphs among $M_1, \ldots, M_\ell, S_1, \ldots, S_k$ do not cross each other, that the edges inside or on the boundary of $\mathcal{C}_{sm}$ do not cross the subgraphs $M_1, \ldots, M_\ell, S_1, \ldots, S_k$, and that the edges inside or on the boundary of $\mathcal{C}_{sm}$ do not cross each other. Hence, Γ is planar.

Finally, we prove that Γ is a dominance drawing.

- Vertices that are internal to the same subgraph among $M_1, \ldots, M_\ell, S_1, \ldots, S_k$ are in the correct dominance relationship, by induction.
- Vertices that are internal to distinct subgraphs among $M_1, \ldots, M_\ell, S_1, \ldots, S_k$ are incomparable. This is because, for any internal vertex v of a subgraph M_j or S_i, we have that t is the only vertex incident to the outer face of M_j or S_i, respectively, that is a successor of v, as a consequence of the fact that P_s and P_m are the longest paths between their end-vertices. By induction, vertices that are internal to distinct subgraphs among $M_1, \ldots, M_\ell, S_1, \ldots, S_k$ are placed into disks among $D^v_1, E^v_1, \ldots, D^v_\ell, E^v_\ell, D^u_1, E^u_1, \ldots, D^u_k$, E^u_k. Also, any two disks associated to distinct subgraphs among $M_1, \ldots, M_\ell, S_1, \ldots, S_k$ are one to the left and above the other one, hence such vertices are in the correct dominance relationship.
- By construction, $v_1, \ldots, v_{\ell-1}$ are to the right and below $u_1, \ldots, u_{k-1}$, which is the correct dominance relationship as any vertex among $v_1, \ldots, v_{\ell-1}$ is incomparable with any vertex among $u_1, \ldots, u_{k-1}$.
- Also by construction, we have that v_j is above and to the right of v_{j-1}, for $j = 1, \ldots, \ell$, and that u_i is above and to the right of u_{i-1}, for $i = 1, \ldots, \ell$, which is the correct dominance relationship because of the existence of the directed paths P_s and P_m.
- Each vertex u_i with $i = 1, \ldots, k$ is below and to the right of every disk among $D^u_1, E^u_1, \ldots, D^u_i$ and is below and to the left of every disk among $E^u_i, D^u_{i+1}, \ldots, D^u_k, E^u_k$; this is indeed the correct dominance relationship, as all the vertices in the former sequence of disks are incomparable to u_i, while all the vertices in the latter sequence of disks are successors of u_i. That the vertices among $v_1, \ldots, v_\ell$ are in the correct dominance relationship with respect to vertices inside disks $D^v_1, E^v_1, \ldots, D^v_\ell, E^v_\ell$ can be argued similarly.
- Each vertex u_i with $i = 1, \ldots, k-1$ is above and to the left of every disk among $D^v_1, E^v_1, \ldots, D^v_\ell$; this is indeed the correct dominance relationship, as u_i is incomparable to every vertex internal to a subgraph M_j with $j = 1, \ldots, \ell$, with the exception of the successors of r in M_ℓ, which are also successors of u_i; these vertices are in E^v_ℓ, which is indeed above and to the right of u_i. Similarly, each vertex v_j with $j = 1, \ldots, \ell-1$ is in the correct dominance relationship with respect to every vertex internal to a subgraph S_i with $i = 1, \ldots, k$. ◀

Clearly, an analogous result holds true for st-plane 3-trees in which the source is adjacent to every vertex.

5 Left-non-transitive st-plane graphs

We now consider *left-non-transitive st-plane graphs*. These are the st-plane graphs such that the left boundary of every face is not a single edge. We show the following.

▶ **Theorem 5.** *Left-non-transitive st-plane graphs admit planar straight-line dominance drawings.*

Proof. Consider a left-non-transitive st-plane graph G. We are going to use a *right-to-left path decomposition of* G. This consists of a sequence of directed paths $P_1, P_2, \ldots, P_k$ such that the following properties are satisfied.

- P_1 is the right boundary of the outer face of G;
- for $i = 1, \ldots, k$, the graph $G_i := P_1 \cup P_2 \cup \cdots \cup P_i$ is an st-plane graph;
- for $i = 2, \ldots, k$, the path P_i is the left boundary of a face of G whose right boundary belongs to the left boundary of the outer face of G_{i-1}; the internal vertices of P_i do not belong to G_{i-1}; and
- $G_k = G$.

This decomposition can be found by ordering the faces of G as in a DFS of the dual of G; for a formal proof see, e.g., [29].

We are going to construct a planar straight-line dominance drawing Γ_i of G_i, for $i = 2, \ldots, k$. Then Γ_k is the desired drawing of G.

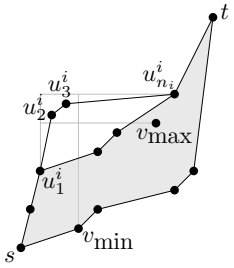


Figure 10 Constructing Γ_i from Γ_{i-1}. The interior of Γ_{i-1} is not shown and shaded gray.

For $i = 1, \ldots, k$, let $P_i = (u^i_1, u^i_2, \ldots, u^i_{n_i})$. Since G is left-non-transitive, $n_i \geq 3$ holds. The drawing Γ_1 of $G_1 = P_1$ is constructed as any straight-line drawing such that $x(u^1_1) < x(u^1_2) < \cdots < x(u^1_{n_1})$ and $y(u^1_1) < y(u^1_2) < \cdots < y(u^1_{n_1})$. Clearly, Γ_1 is a planar dominance drawing. Now suppose that a planar straight-line dominance drawing Γ_{i-1} of G_{i-1} has been constructed, for some $i \in \{2, \ldots, k\}$, so that no two vertices have the same x- or y-coordinate. We construct a planar straight-line dominance drawing Γ_i of G_i from Γ_{i-1} as follows; refer to Fig 10. Recall that u^i_1 and $u^i_{n_i}$ are vertices on the left boundary of G_{i-1}, while the internal vertices of P_i need to be inserted into Γ_{i-1} in order to define Γ_i. Among all the vertices of G_{i-1} that lie to the right of u^i_1 in Γ_{i-1}, let $v_{\min}$ be the one with smallest x-coordinate. Also, among all the vertices of G_{i-1} that lie below $u^i_{n_i}$ in Γ_{i-1}, let $v_{\max}$ be the one with largest y-coordinate. Note that $x(v_{\min}) \leq x(u^i_{n_i})$ and $y(u^i_1) \leq y(v_{\max})$. We assign coordinates to the internal vertices of P_i so that $x(u^i_1) < x(u^i_2) < \cdots < x(u^i_{n_i-1}) < x(v_{\min})$ and $y(v_{\max}) < y(u^i_2) < \cdots < y(u^i_{n_i-1}) < y(u^i_{n_i})$. This completes the construction of Γ_i.

We prove the planarity of Γ_i. Since the drawing of G_{i-1} in Γ_i coincides with Γ_{i-1} and since Γ_{i-1} is planar, it suffices to prove that the edges of P_i do not cross each other and do not cross Γ_{i-1}. The former follows directly from the fact that $x(u_1^i) < x(u_2^i) < \cdots < x(u_{n_i-1}^i) < x(u_{n_i}^i)$ and $y(u_1^i) < y(u_2^i) < \cdots < y(u_{n_i-1}^i) < y(u_{n_i}^i)$, by construction. We now deal with the latter.

- First, we prove that the edge (u_1^i, u_2^i) does not cross Γ_{i-1}. Let (u_1^i, w) be the edge of G_{i-1} outgoing from u_1^i and incident to the left boundary of G_{i-1}. Such an edge has the outer face of Γ_{i-1} to its left, when traversed from u_1^i to w. By construction, we have $x(u_2^i) < x(v_{\min}) \leq x(w)$ and $y(w) \leq y(v_{\max}) < y(u_2^i)$, hence the interval of x-coordinates spanned by the edge (u_1^i, u_2^i) is a subset of the one spanned by the edge (u_1^i, w) and the slope of the edge (u_1^i, u_2^i) is larger than the one of the edge (u_1^i, w). It follows that (u_1^i, u_2^i) lies in the outer face of Γ_{i-1}, and hence does not cross Γ_{i-1}.
- The proof that the edge $(u_{n_i-1}^i, u_{n_i}^i)$ does not cross Γ_{i-1} is analogous.
- Finally, consider any edge (u_j^i, u_{j+1}^i) with $2 \leq j \leq n_i - 2$. By construction, the interval of x-coordinates spanned by (u_j^i, u_{j+1}^i) is a subset of the interval $(x(u_1^i), x(w))$, where w is defined as above. Also by construction, we have that $y(u_1^i) < y(w) \leq y(v_{\max}) < y(u_j^i) < y(u_{j+1}^i)$. Hence, the edge lies above the edge (u_1^i, w), thus in the outer face of Γ_{i-1}, which is not crossed by it.

We now prove that Γ_i is a dominance drawing. Since the drawing of G_{i-1} in Γ_i coincides with Γ_{i-1} and since Γ_{i-1} is a dominance drawing, it suffices to prove that the placement of the internal vertices of P_i complies with the dominance relationships they are involved in. Consider any internal vertex u_j^i of P_i. For $h = 1, \ldots, j-1$, vertex u_h^i is a predecessor of u_j^i and indeed we have $x(u_h^i) < x(u_j^i)$ and $y(u_h^i) < y(u_j^i)$, by construction. Analogously, for $h = j+1, \ldots, n_i$, vertex u_h^i is a successor of u_j^i and indeed we have $x(u_j^i) < x(u_h^i)$ and $y(u_j^i) < y(u_h^i)$, by construction. Consider any vertex w of G_{i-1} different from u_1^i and $u_{n_i}^i$.

- First, if w is a predecessor of u_1^i, then it is also a predecessor of u_j^i and indeed we have $x(w) < x(u_j^i)$ and $y(w) < y(u_j^i)$, given that $x(w) < x(u_1^i)$ and $y(w) < y(u_1^i)$ (since Γ_{i-1} is a dominance drawing) and that $x(u_1^i) < x(u_j^i)$ and $y(u_1^i) < y(u_j^i)$ (as proved above).
- Second, if w is a successor of $u_{n_i}^i$, then it is also a successor of u_j^i and indeed we have $x(u_j^i) < x(w)$ and $y(u_j^i) < y(w)$ given that $x(u_{n_i}^i) < x(w)$ and $y(u_{n_i}^i) < y(w)$ (since Γ_{i-1} is a dominance drawing) and that $x(u_j^i) < x(u_{n_i}^i)$ and $y(u_j^i) < y(u_{n_i}^i)$ (as proved above).
- Finally, if w is neither a predecessor of u_1^i nor a successor of $u_{n_i}^i$, then it is incomparable with u_j^i. Note that $x(w) > x(u_1^i)$, as $x(w) < x(u_1^i)$ would imply $y(w) < y(u_1^i)$ (given that u_1^i is on the left boundary of G_{i-1}), which is not possible since w is incomparable with u_1^i and Γ_{i-1} is a dominance drawing. Analogously, we have $y(w) < y(u_{n_i}^i)$. By construction, we have $x(u_j^i) < x(v_{\min}) \leq x(w)$ and $y(u_j^i) > y(v_{\max}) \geq y(w)$, hence the placement of w and u_j^i complies with their dominance relationship.

This concludes the proof that Γ_i is a planar straight-line dominance drawing, hence the induction and the proof of the theorem. ◀

Clearly, an analogous result holds true for *right-non-transitive st-plane graphs*, which are *st*-plane graphs such that the right boundary of every face is not a single edge.

6 Conclusions and Open Problems

In this paper, we tackled the following problem: Does every *st*-plane graph admit a planar straight-line dominance drawing? While we were not able to solve this question in its generality, our research advanced the state of the art in many directions.

First, we have provided concrete evidence for the difficulty in constructing planar straight-line dominance drawings. Most notably, we proved that planar straight-line dominance drawings with prescribed y-coordinates do not always exist. Our research in this direction indicates that, if an algorithm that constructs a planar straight-line dominance drawing of every st-plane graph exists, then it should use substantially different ideas than known algorithms for the construction of upward planar straight-line drawings.

Second, we have described several classes of st-plane graphs that admit a planar straight-line dominance drawing. A difficult benchmark here is, in our opinion, provided by the st-plane 3-trees. Hence, we believe it would be a major milestone to understand whether these graphs always admit planar straight-line dominance drawings.

We conclude with one more open problem. Does every (undirected) maximal planar graph admit a planar straight-line dominance drawing? That is, does it admit an st-orientation such that the resulting st-plane graph has a planar straight-line dominance drawing?

References

1 Carlos Alegría-Galicia, Susanna Caroppo, Giordano Da Lozzo, Marco D'Elia, Giuseppe Di Battista, Fabrizio Frati, Fabrizio Grosso, and Maurizio Patrignani. Upward pointset embeddings of planar st-graphs. In Stefan Felsner and Karsten Klein, editors, *32nd International Symposium on Graph Drawing and Network Visualization, GD 2024, September 18-20, 2024, Vienna, Austria*, volume 320 of *LIPIcs*, pages 24:1–24:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.GD.2024.24.

2 Carlos Alegría-Galicia, Susanna Caroppo, Giordano Da Lozzo, Marco D'Elia, Giuseppe Di Battista, Fabrizio Frati, Fabrizio Grosso, and Maurizio Patrignani. Upward pointset embeddings of planar st-graphs. *Algorithmica*, 87(6):930–960, 2025. doi:10.1007/s00453-025-01302-2.

3 Patrizio Angelini, Therese Biedl, Markus Chimani, Sabine Cornelsen, Giordano Da Lozzo, Seok-Hee Hong, Giuseppe Liotta, Maurizio Patrignani, Sergey Pupyrev, Ignaz Rutter, and Alexander Wolff. The Price of Upwardness. In Stefan Felsner and Karsten Klein, editors, *32nd International Symposium on Graph Drawing and Network Visualization (GD 2024)*, volume 320 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.GD.2024.13.

4 Patrizio Angelini, Steven Chaplick, Sabine Cornelsen, and Giordano Da Lozzo. On upward-planar L-drawings of graphs. *J. Graph Algorithms Appl.*, 28(1):275–299, 2024. doi:10.7155/JGAA.V28I1.2950.

5 Michael J. Bannister, William E. Devanny, and David Eppstein. Small superpatterns for dominance drawing. In Michael Drmota and Mark Daniel Ward, editors, *Eleventh Workshop on Analytic Algorithmics and Combinatorics (ANALCO 2014)*, pages 92–103. SIAM, 2014. doi:10.1137/1.9781611973204.9.

6 Michael A. Bekos, Giordano Da Lozzo, Fabrizio Frati, Martin Gronemann, Tamara Mchedlidze, and Chrysanthi N. Raftopoulou. Recognizing DAGs with page-number 2 is NP-complete. In Patrizio Angelini and Reinhard von Hanxleden, editors, *30th International Symposium on Graph Drawing and Network Visualization (GD 2022)*, volume 13764 of *Lecture Notes in Computer Science*, pages 361–370. Springer, 2022. doi:10.1007/978-3-031-22203-0_26.

7 Michael A. Bekos, Giordano Da Lozzo, Fabrizio Frati, Martin Gronemann, Tamara Mchedlidze, and Chrysanthi N. Raftopoulou. Recognizing DAGs with page-number 2 is NP-complete. *Theor. Comput. Sci.*, 946:113689, 2023. doi:10.1016/J.TCS.2023.113689.

8 Paola Bertolazzi, Robert F. Cohen, Giuseppe Di Battista, Roberto Tamassia, and Ioannis G. Tollis. How to draw a series-parallel digraph. *Int. J. Comput. Geom. Appl.*, 4(4):385–402, 1994. doi:10.1142/S0218195994000215.

9 Sujoy Bhore, Giordano Da Lozzo, Fabrizio Montecchiani, and Martin Nöllenburg. On the upward book thickness problem: Combinatorial and complexity results. *Eur. J. Comb.*, 110:103662, 2023. doi:10.1016/J.EJC.2022.103662.

10 Carla Binucci, Giordano Da Lozzo, Emilio Di Giacomo, Walter Didimo, Tamara Mchedlidze, and Maurizio Patrignani. Upward book embeddability of st-graphs: Complexity and algorithms. *Algorithmica*, 85(12):3521–3571, 2023. doi:10.1007/S00453-023-01142-Y.

11 Carla Binucci, Walter Didimo, and Maurizio Patrignani. *st*-orientations with few transitive edges. *J. Graph Algorithms Appl.*, 27(8):625–650, 2023. doi:10.7155/JGAA.00638.

12 Steven Chaplick, Markus Chimani, Sabine Cornelsen, Giordano Da Lozzo, Martin Nöllenburg, Maurizio Patrignani, Ioannis G. Tollis, and Alexander Wolff. Planar L-drawings of directed graphs. *Comput. Geom. Topol.*, 2(1):7:1–7:15, 2023. doi:10.57717/cgt.v2i1.43.

13 Giuseppe Di Battista and Enrico Nardelli. Hierarchies and planarity theory. *IEEE Trans. Syst. Man Cybern.*, 18(6):1035–1046, 1988. doi:10.1109/21.23105.

14 Giuseppe Di Battista and Roberto Tamassia. Algorithms for plane representations of acyclic digraphs. *Theor. Comput. Sci.*, 61:175–198, 1988. doi:10.1016/0304-3975(88)90123-5.

15 Giuseppe Di Battista, Roberto Tamassia, and Ioannis G. Tollis. Area requirement and symmetry display of planar upward drawings. *Discret. Comput. Geom.*, 7:381–401, 1992. doi:10.1007/BF02187850.

16 Emilio Di Giacomo, Henry Förster, Daria Kokhovich, Tamara Mchedlidze, Fabrizio Montecchiani, Antonios Symvonis, and Anaïs Villedieu. On 1-bend upward point-set embeddings of st-digraphs. In José A. Soto and Andreas Wiese, editors, *16th Latin American Symposium on Theoretical Informatics (LATIN)*, volume 14578 of *LNCS*, pages 3–18. Springer, 2024. doi:10.1007/978-3-031-55598-5_1.

17 Peter Eades, Qing-Wen Feng, Xuemin Lin, and Hiroshi Nagamochi. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. *Algorithmica*, 44(1):1–32, 2006. doi:10.1007/S00453-004-1144-8.

18 Hossam A. ElGindy, Michael E. Houle, William J. Lenhart, Mirka Miller, David Rappaport, and Sue Whitesides. Dominance drawings of bipartite graphs. In *5th Canadian Conference on Computational Geometry*, pages 187–191. University of Waterloo, 1993.

19 István Fáry. On straight line represention of planar graphs. *Acta Scientiarum Mathematicarum*, 11:229–233, 1948.

20 Fabrizio Frati. Upward planarity testing of biconnected outerplanar DAGs solves partition. *Theor. Comput. Sci.*, 998:114541, 2024. doi:10.1016/J.TCS.2024.114541.

21 Seok-Hee Hong and Hiroshi Nagamochi. Convex drawings of hierarchical planar graphs and clustered planar graphs. *J. Discrete Algorithms*, 8(3):282–295, 2010. doi:10.1016/J.JDA.2009.05.003.

22 Bart M. P. Jansen, Liana Khazaliya, Philipp Kindermann, Giuseppe Liotta, Fabrizio Montecchiani, and Kirill Simonov. Upward and orthogonal planarity are W[1]-hard parameterized by treewidth. In Michael A. Bekos and Markus Chimani, editors, *31st International Symposium on Graph Drawing and Network Visualization (GD 2023)*, volume 14466 of *LNCS*, pages 203–217. Springer, 2023. doi:10.1007/978-3-031-49275-4_14.

23 Paul Jungeblut, Laura Merker, and Torsten Ueckerdt. Directed acyclic outerplanar graphs have constant stack number. In *64th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2023)*, pages 1937–1952. IEEE, 2023. doi:10.1109/FOCS57990.2023.00118.

24 Paul Jungeblut, Laura Merker, and Torsten Ueckerdt. A sublinear bound on the page number of upward planar graphs. *SIAM J. Discret. Math.*, 37(4):2312–2331, 2023. doi:10.1137/22M1522450.

25 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.

26 Evgenios M. Kornaropoulos and Ioannis G. Tollis. Weak dominance drawings for directed acyclic graphs. In Walter Didimo and Maurizio Patrignani, editors, *20th International Symposium on Graph Drawing (GD 2012)*, volume 7704 of *Lecture Notes in Computer Science*, pages 559–560. Springer, 2012. doi:10.1007/978-3-642-36763-2_52.

27 Panagiotis Lionakis, Giacomo Ortali, and Ioannis G. Tollis. Constant-time reachability in DAGs using multidimensional dominance drawings. *SN Comput. Sci.*, 2(4):320, 2021. doi:10.1007/S42979-021-00713-6.

28 Maarten Löffler. Strict Upward Planar Grid Drawings of Binary Trees with Minimal Area. In Stefan Felsner and Karsten Klein, editors, *32nd International Symposium on Graph Drawing and Network Visualization (GD 2024)*, volume 320 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:3. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.GD.2024.47.

29 Kurt Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, volume 3 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1984. doi:10.1007/978-3-642-69900-9.

30 Martin Nöllenburg and Sergey Pupyrev. On families of planar DAGs with constant stack number. In Michael A. Bekos and Markus Chimani, editors, *31st International Symposium on Graph Drawing and Network Visualization (GD 2023)*, volume 14465 of *LNCS*, pages 135–151. Springer, 2023. doi:10.1007/978-3-031-49272-3_10.

31 Giacomo Ortali and Ioannis G. Tollis. Multidimensional dominance drawings and their applications. *Foundations*, 1(2):271–285, 2021. doi:10.3390/foundations1020020.

32 Giacomo Ortali and Ioannis G. Tollis. Dominance drawings for DAGs with bounded modular width. In Leszek Gasieniec, editor, *48th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2023)*, volume 13878 of *LNCS*, pages 65–79. Springer, 2023. doi:10.1007/978-3-031-23101-8_5.

33 János Pach and Géza Tóth. Monotone drawings of planar graphs. *J. Graph Theory*, 46(1):39–47, 2004. doi:10.1002/JGT.10168.

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

35 Renê Rodrigues Veloso, Loïc Cerf, Wagner Meira Jr., and Mohammed J. Zaki. Reachability queries in very large graphs: A fast refined online search approach. In Sihem Amer-Yahia, Vassilis Christophides, Anastasios Kementsietsidis, Minos N. Garofalakis, Stratos Idreos, and Vincent Leroy, editors, *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014*, pages 511–522. OpenProceedings.org, 2014. doi:10.5441/002/EDBT.2014.46.

36 David R. Wood. A simple proof of the Fáry-Wagner theorem. *CoRR*, abs/cs/0505047, 2005. arXiv:cs/0505047.

Vantage Point Selection Algorithms for Bottleneck Capacity Estimation

Vikrant Ashvinkumar
Rutgers University, Piscataway, NJ, USA

Rezaul Chowdhury
Stony Brook University, NY, USA

Jie Gao
Rutgers University, Piscataway, NJ, USA

Mayank Goswami
Queens College, City University of New York, NY, USA

Joseph S. B. Mitchell
Stony Brook University, NY, USA

Valentin Polishchuk
Linköping University, Sweden

Abstract

Motivated by the problem of estimating bottleneck capacities on the Internet, we formulate and study the problem of vantage point selection. We are given a graph $G = (V, E)$ whose edges E have unknown capacity values that are to be discovered. Probes from a vantage point, i.e, a vertex $v \in V$, along shortest paths from v to all other vertices, reveal bottleneck edge capacities along each path. Our goal is to select k vantage points from V that reveal the maximum number of bottleneck edge capacities.

We consider both a non-adaptive setting where all k vantage points are selected before any bottleneck capacity is revealed, and an adaptive setting where each vantage point selection instantly reveals bottleneck capacities along all shortest paths starting from that point. In the non-adaptive setting, by considering a relaxed model where edge capacities are drawn from a random permutation (which still leaves the problem of maximizing the expected number of revealed edges NP-hard), we are able to give a $1 - 1/e$ approximate algorithm. In the adaptive setting we work with the least permissive model where edge capacities are arbitrarily fixed but unknown. We compare with the best solution for the particular input instance (i.e. by enumerating all choices of k tuples), and provide both lower bounds on instance optimal approximation algorithms and upper bounds for trees and planar graphs.

2012 ACM Subject Classification Networks → Network algorithms

Keywords and phrases Bottleneck capacity, Approximation algorithms, Instance optimality

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.6

Related Version *Full Version*: `https://arxiv.org/pdf/2506.21418`

Funding *Rezaul Chowdhury*: Partially supported by NSF (CCF-2318633).
Jie Gao: Ashvinkumar and Gao are supported by NSF through IIS-2229876, DMS-2220271, DMS-2311064, CCF-2208663, CCF-2118953.
Mayank Goswami: Supported by NSF grant CCF-2503086.
Joseph S. B. Mitchell: Partially supported by NSF (CCF-2007275).

Acknowledgements We thank an anonymous reviewer who caught an error in a claimed result that has now been omitted from this paper; we discuss and pose the question as the first open problem in Section 4. We also thank Srinivas Narayana for discussions on capacity discovery.

© Vikrant Ashvinkumar, Rezaul Chowdhury, Jie Gao, Mayank Goswami, Joseph S. B. Mitchell, and Valentin Polishchuk;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Network tomography [31, 9, 10] concerns the problem of inferring the internal topology and parameters of a network based on end-to-end measurements. For example, collecting IP, router, and provider-level network topologies has been an active research topic for more than 20 years. Long-term continuous measurement efforts such as CAIDA's Ark infrastructure [21] provide important information useful to many longitudinal analyses and network events of interest. One of the basic measurement tools is TraceRoute, which obtains the sequence of router interface IP addresses along the forward path to a destination by sending probe packets with varying time to live (TTL) values and examining the Internet Control Message Protocol (ICMP) responses. By using the transmission timestamp of each probe, TraceRoute can report the round trip time from the source to *each* node on the forward path. Although TraceRoute was initially designed for network administrators to make diagnoses on a small scale, recent efforts applied TraceRoute for Internet scale topology probing [20, 6].

Beyond network topology, estimating network bottleneck capacity is a classical research topic [1, 16, 5, 17]. The capacity of a link (edge) is the highest bit rate possible to transmit over the edge. The capacity of a path is determined by the *bottleneck* edge, the one on the path with the lowest capacity [28]. Estimating the bottleneck or available capacity between a pair of vertices is useful for many applications, such as routing management, intrusion detection and improving performance of transport protocols. In particular, knowledge of network bandwidth can be instrumental to client-side optimization in real-time throughput sensitive applications, such as video conferencing. A list of existing measurement tools for available bandwidth estimation is summarized in [29]. Existing techniques mostly focus on end-to-end capacity, using tools such as variable packet size, packet pair/train dispersion, or periodic streams. Common to these different implementations is the central idea of examining the *jumps* in round trip time (RTT) from the source to each hop of a path. Since the bottleneck edge determines the available capacity of the entire path, the bottleneck capacity can be discovered by examining packet latency in probes.

The common practice in large scale measurements of the Internet issues probes from a narrow range of vantage points (VPs), typically cloud systems or research universities, due to cost, convenience, accessibility and scalability [22]. For example, RIPE Atlas [3], a platform that supports user specified measurement requests, has over 9600 active measurement vantage points. But a user is limited to the total number of vantage points used in a measurement. Therefore, a user would naturally seek to maximize the amount of information one could obtain by optimizing the choice of vantage points [19].

In this paper we formulate mathematically the problem of choosing vantage points for network probing. Our goal is to use probing tools for network-wide capacity discovery. We consider a network with publicly known topology, where each edge/link has an unknown capacity. We would like to select (possibly from a given set of vertices) a set of k vantage points from which probing messages are sent out to other vertices in the network. We use a model of probing in which probing from a source s along a path to a destination t reveals the bottleneck capacity of the sub-path from s to v, for every intermediate vertex v on the path. In this model, if the bottleneck capacity drops from C_1 to C_2, in comparing paths $P(s, v)$ and $P(s, v')$ with v' being the immediate downstream vertex after v, then the edge (v, v') has capacity C_2. Our objective is to reveal the link capacity for as many edges in the network as possible.

In this paper we assume that the network topology is known, e.g., learned using the relatively mature methods for topology discovery. We also work mainly with an assumption that all paths along the probings are unique shortest paths. Thus the set of shortest paths

from one vertex to all other vertices in the network forms a shortest path tree. To examine the connection to reality, we first remark that modern Internet uses multi-path routing, where traffic to a destination can be spread on multiple paths for throughput and redundancy [4, 32]. On the other hand, the assumption that routes originating from a vantage point to the Internet form a tree-like structure is heavily exploited to reduce probing redundancy in modern TraceRoute tools, e.g., Doubletree [12] and its variants [11, 24]. Therefore it is reasonable to assume that using existing tools the routes probed starting from a vantage point indeed form a tree.

There has been a number of empirical work understanding vantage points on the Internet in terms of their characteristics and influence on data collected for network probing [8, 7, 25, 30]. None of the prior work is directly related to our problem formulation.

The problem of vantage point selection is not limited to the Internet domain. There are a number of other network scenarios where estimating bottleneck on the network is important. This problem is natural for transportation network for estimating traffic bottleneck although probing on the transportation network may have to rely on opportunistic inputs. In the traffic engineering literature, the traffic sensor location problem (TSLP) [27] is to determine how many sensors are required and where should they be deployed in order to best understand the traffic bottlenecks in road and transportation networks. For a blood vessel network, a medical procedure known as angiography considers probing of the blood vessel network of a patient by a dye injected into the bloodstream through a catheter, in order to determine the blocked vessel (the bottleneck). On an abstract level, determining the site for the injection(s) is similar to the bottleneck discovery problem in this paper.

1.1 Problem Definition

Here we formulate the problem of vantage point selection for bottleneck capacity discovery. We are given an undirected graph $G = (V, E)$ with each edge associated with an *unknown* positive capacity $c(\cdot)$. The goal is to discover these unknown capacities by using bottleneck queries from vantage points u, which reveals bottlenecks on the shortest path between u and v for all $v \in V$. The shortest path may be defined by the path of minimum number of hops or by another weight metric which is known (the only unknown is the capacities). Unless mentioned otherwise, we will assume (perhaps with slight perturbation of the edge weights) that all shortest paths are unique. The problem is then to select a set of k vertices S as vantage points such that when we issue queries from each vantage point, we can reveal a maximum number of the unknown edge capacity values.

We first discuss what edges are revealed from selecting S as a set of vantage points.

▶ **Definition 1** (Bottleneck Edges). *Denote by $P(s,t)$ the shortest path from s to t, and let $e_1, e_2, \ldots e_\ell$ be the edges along this path. We say that, for $i \in [\ell]$, e_i is a* bottleneck edge *along $P(s,t)$ if $c(e_i) < c(e_j)$ for all $j \in [\ell] \setminus \{i\}$, and $c(e_i)$ is a* bottleneck capacity*. The bottleneck edges from a set S to V are edges that are bottleneck along $P(s,t)$ for some $s \in S$ and some $t \in V$.*

Note that while we state the definition of a bottleneck edge with respect to a path $P(s,t)$, in accord with the standard definition of a bottleneck edge, we extend the definition to a set S to match our query model: for every $s \in S$, a single query comprises of $n-1$ subqueries finding the bottleneck edge in $P(s,t)$ for all $t \in V \setminus \{s\}$. For example, on the path graph $v_0, v_1, \ldots, v_n$ where edges have monotone increasing capacities, querying v_0 results in the revelation of all capacities whereas querying only reveals the capacities of the last edge.

With this background, we can loosely define the problem of bottleneck discovery.

▶ **Problem 1** (BottleneckDiscovery). *Given a non-negatively weighted (weights w) undirected graph $G = (V, E, w)$, and a positive integer k, find a subset of vertices $S \subseteq V$ with $|S| = k$ such that the number of revealed bottleneck edges is maximized.*

Non-Adaptive Setting. In this setting, we work with the assumption that the ordering of the edge capacities is uniformly randomly chosen from a permutation of $1, 2, \ldots, m = |E|$. Thus, each edge has equal chance of being the one with the smallest capacity among all edges in the network. Here we consider algorithms in a non-adaptive manner, where the k vantage points are selected all at once and probes are issued afterwards. We seek to maximize the *expected* number of edges revealed.

Adaptive Setting. In the adaptive setting, we work in the worst-case model where we assume the edge capacities to be arbitrarily fixed distinct real numbers, unknown to the algorithm. We also allow algorithms that run in an adaptive manner, where the vantage points are selected one by one, with probes from a vantage point issued immediately upon selection, before we select the next vantage point.

Here we aim to perform well on any specific input. In other words, we study the *instance-optimal* setting. Traditionally, instance optimality is studied by comparing the output of an algorithm to the optimal solution for the input instance. In this spirit, we assume an algorithm OPT that is already aware of all the capacities, and for a specific input instance I, let $OPT(I)$ denote the maximum number of edges revealed by k vantage points on instance I, i.e., obtained by enumerating all subsets of k vantage points and keeping the best choice. In contrast, a capacity-unaware algorithm will be called (α, β)-*instance optimal* if after selecting at most αk vantage points (possibly in an adaptive manner) on the input instance I, it can reveal at least $OPT(I)/\beta$ many edge capacities. We can think of $\alpha \geq 1$ as the resource augmentation factor and $\beta \geq 1$ as the approximation factor for the objective function.

We close this subsection with the following observations on the problem.

▶ **Observation 2.** *Let T_s be the shortest path tree rooted at a vantage point $s \in S$.*

- *The capacities of all edges in T_s that are incident to s are revealed.*
- *In any root to leaf path from s to t on T_s, the capacities of the edges revealed form a record setting decreasing subsequence, where the capacity of the ith revealed edge is strictly smaller than all edges earlier on the path (and of course smaller than that of the $(i-1)$st revealed edge).*
- *If u is on the shortest path from s to w, all edges that s can reveal along the path from u to w can instead be revealed if u is selected as a vantage point.*

1.2 Overview of Results

Non-Adaptive Setting (with Stochastic Capacities). We start with the nonadaptive setting with stochastic capacity assumption, where k vantage points are selected before any information on the bottleneck capacities is revealed. Here, the ordering of the capacities c is assumed to be a random permutation of $[m]$, and we look for an algorithm that maximizes the expected performance. This problem is NP-hard, since the vertex cover problem is a special case. We therefore aim for approximation algorithms and compare with the optimal algorithm which works under the same assumption.

A first observation is that the expected number of revealed edges obtained using a subset S of vantage points is a monotone and submodular function with respect to S. That is, adding a new vantage point w is always beneficial and adding w to a set Y rather than X, with

$X \subseteq Y$, has a diminishing return. This observation allows us to use a greedy algorithm to select the next vantage point that maximizes the expected marginal increase of the objective function, which yields a $(1-1/e)$-approximation algorithm, by known results for maximizing coverage for submodular functions [26]. To obtain our approximation result, then, we must address the non-trivial calculation of the expected marginal increase of adding a vantage point to the set being constructed greedily. A crucial step here is to efficiently count the number of capacity assignments to the graph such that an edge e is revealed as the result of a new vantage point. We show that the algorithm can be implemented in time $\widetilde{O}(kmn^3)$, with n as the number of vertices, m as the number of edges and k as the total number of vantage points selected.

We remark that one can also resort to a sampling-based approach (e.g., see Proposition 3 in [23]), which does not provide better running time in general and can only provide an approximate solution which leads to a $1-1/e-\varepsilon$ approximation factor. Our deterministic algorithm provides more insight into the structure of the problem.

Adaptive Setting (with Worst-Case Capacities). In the adaptive setting, we prove the following results. First, we show interesting lower bounds for the tradeoffs of parameters α, β and k for (α, β)-instance optimal algorithms, where OPT selects k vantage points (with the full knowledge of capacities). One trivial algorithm is to select all n vertices as vantage points, as opposed to the k vertices selected by OPT, obtaining $\alpha = n/k$ and $\beta = 1$. We thus ask if we can lower α with respect to the other parameters. Our lower bounds suggest that α cannot be too small. For deterministic adaptive algorithms, we show that $\alpha^2\beta = \Omega(n/k)$; for randomized adaptive ones, $\alpha\beta = \widetilde{\Omega}(\sqrt{n/k})$. These lower bounds apply for algorithms that can perform any finite computation between the selection of the k vintage points.

We then show that both bounds are tight when the underlying graph is a tree, i.e., we give vantage point selection algorithm achieving the same tradeoffs. For planar graphs, we give a deterministic algorithm that achieves a tradeoff of $\alpha\beta = O((n/k)^{2/3})$. Finally, while we leave open the question of existence of an $\alpha\beta = o(n)$ algorithm for general graphs, we show that if the shortest path trees are not unique, and each shortest path tree may break ties independent from others, there is a stronger lower bound of $\alpha\beta = \Omega(n^{1-\varepsilon})$, for all $\varepsilon > 0$.

We present the algorithmic results in the main body of the paper. We postpone the hardness proof for both settings and some proof details to the Appendix.

2 BottleneckDiscovery in the Non-Adaptive Setting (with Stochastic Capacities)

In this section we show a constant approximation to BottleneckDiscovery in the non-adaptive setting with stochastic capacities, which is NP-hard (Appendix A).

▶ **Theorem 3.** *There is an algorithm that gives a* $(1-1/e)$ *approximation to* BottleneckDiscovery *in time* $\widetilde{O}(kmn^3)$*, where* e *is the base of the natural logarithm.*

Define $R(S)$ to be the set of edges that are revealed with vertices S selected, and let $f(S) = \mathbb{E}[|R(S)|]$. We prove Theorem 3 in two parts. First, we show that f is monotone submodular, and hence a greedy algorithm for selection into S yields a $(1-1/e)$ approximation by the result of [26]. This is the less interesting part and so the details are defered to Section B.1. We then design a value oracle for each iteration of the greedy algorithm, which is implemented by using polynomial multiplications.

Setup. Recall that $R(S)$ is the set of edges whose capacities are revealed from selecting vantage points S, and we would like to select vertex w to maximize $f(S \cup \{w\})$. By linearity of expectation, $f(S \cup \{w\}) = \sum_e \Pr[e \in R(S \cup \{w\})]$. It is thus enough to show how to compute $\Pr[e \in R(S \cup \{w\})]$ for an arbitrary subset $S \subseteq V$, after which we can sum over all the edges at the expense of a multiplicative m factor blowup in the running time.

(Overview) Computing $\Pr[e \in R(S)]$. For the remainder of this section, let us fix an edge $e = (u, v)$ for which we want to compute $p = \Pr[e \in R(S)]$, and let us also have S fixed. First, we show what vertices $s \in S$ contribute to p.

▶ **Definition 4** ((Non-)essential vantage points). *A vantage point $s \in S$ is said to be* non-essential *if at least one of the following holds:*

- *$P(s, u)$ does not go through (v, u) and $P(s, v)$ does not go through (u, v).*
- *If $P(s, u)$ goes through (v, u), there is some $s' \neq s$ where $s' \in P(s, u) \cap S$.*
- *If $P(s, v)$ goes through (u, v), there is some $s' \neq s$ where $s' \in P(s, v) \cap S$.*

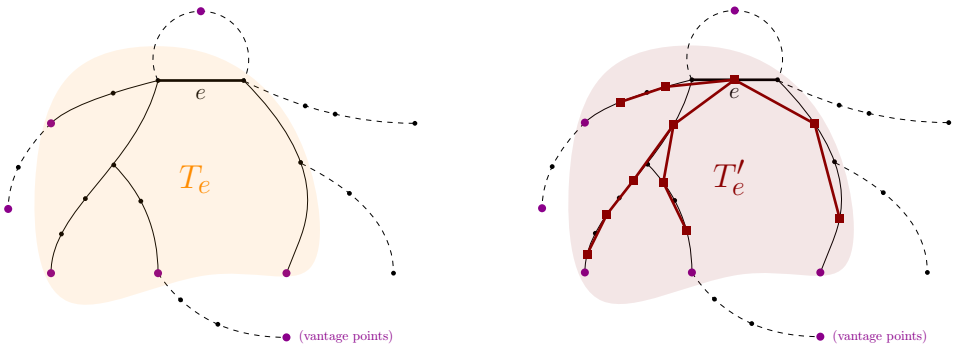
If none of the above hold, we say that $s \in S$ is essential.

It is reasonably clear that non-essential vantage points do not have any bearing on the value of p. We now state a few observations with proofs in Appendix B.

▶ **Observation 5.** *Let $S' \subseteq S$ be the set of all essential vantage points. Then $\Pr[e \in R(S)] = \Pr[e \in R(S')]$.*

Now we define the minimal subgraph required to compute $\Pr[e \in R(S)]$.

▶ **Definition 6** (T_e, the tree rooted at e). *Let $S' \subseteq S$ be the set of all essential vantage points. We define $T_e = \bigcup_{s' \in S'} P(s', u) + e$ with (edge-)root e (see Figure 1a).*



(a) T_e, whose edges are solid. Vantage points are in magenta.

(b) T'_e, whose vertices (squares) and edges (thick lines) are dark red.

Figure 1 Depictions of T_e and T'_e, the latter of which is convenient to use for CountGoodLabellings.

▶ **Observation 7.** *T_e is a tree, with all its leaves belonging to S. Moreover, T_e can be computed in $\widetilde{O}(m)$ time.*

▶ **Observation 8.** *e is a bottleneck edge for S if and only if $c(e)$ is the smallest capacity on some (edge-)root to (vertex-)leaf path on T_e.*

Put otherwise, using Observation 8, we have reduced the problem of computing $\Pr[e \in R(S)]$ to that of counting labellings of the edges of T_e with $[|V(T_e)| - 1]$ such that no two edges receive the same label, and e has the smallest label on some root to leaf path of T_e (recall that T_e is an edge-rooted tree). These labellings can be thought of as the order statistic of edge capacities in T_e.

Counting T_e Labellings. For convenience, we work on the "edge-tree" T'_e rooted at e

$$T'_e = (E(T_e), \{(f, \text{parent-edge}_{T_e}(f)) \mid f \in E(T_e)\}),$$

which allows us to think in the parlance of labelling vertices instead of edges (see Figure 1b). One can see that counting vertex-labellings on T'_e such that there is some root to leaf path where the label of e is smallest is the same as counting similarly constrained edge-labellings on T_e. This leaves us with the following problem.

▶ **Problem 2** (CountGoodLabellings). *Given a rooted tree $T = (V, E)$ with root v_1, how many bijective labellings of V with $[|V|]$ result in the label of v_1 being the minimum label of some root-to-leaf path?*

To solve CountGoodLabellings, however, we first consider a more constrained auxiliary problem.

▶ **Problem 3** (CountBlackWhiteColorings). *Given a rooted tree $T = (V, E)$ with root v_1, compute, for each $t \in [|V|]$, how many black-white colorings of V there are such that:*

- *Exactly t vertices are colored black;*
- *There is no root-to-leaf path comprising of only vertices colored white.*

Conceptually, we can think of vertices colored black as those whose labels are smaller than the root's, and vertices colored white as those whose labels are no smaller than the root's. A solution to CountBlackWhiteColorings counts bad events, since, under this viewpoint we want some root-to-leaf path with all vertices colored white.

▶ **Observation 9.** *Let $B[0 \dots t]$ be a solution to* CountBlackWhiteColorings *on $T = (V, E)$. Then*

$$\sum_{0 \le t \le |V|} \left(\binom{|V|}{t} - B[t] \right) (|V| - 1 - t)! t!$$

gives a solution to CountGoodLabellings.

Proof. There are $\binom{|V|}{t} - B[t]$ colorings such that exactly t vertices are colored black and there is some white root-to-leaf path. For each such coloring, there are exactly $(|V| - 1 - t)!t!$ labellings of the vertices such that white vertices have larger labels than the root's label; black vertices have smaller labels than the root's label; and there is some root-to-leaf path where the root has the smallest label. The result then follows from adding up good labellings where exactly t vertices have a smaller label than that of the root. ◀

We now supply a polynomial time algorithm for CountBlackWhiteColorings, which returns a polynomial $\sum_t B[t]x^t$.

```
CountBlackWhiteColorings (T = (V, E), v ∈ V)
  if v is leaf
      return x
  for u ∈ children(v)
      P_u(x) ← CountBlackWhiteColorings(T, u)
  T' ← subtree rooted at v
  return Π_u P_u(x) + x(1 + x)^{|V(T')|−1}
```

▷ Claim 10. CountBlackWhiteColorings returns a polynomial such that the coefficient of x^t is the number of colorings of V there are such that exactly t vertices are colored black and there is no white root-to-leaf path.

Proof. The claim is certainly true for trees with one vertex. More generally, consider any vertex v. If v is colored black, we can color its descendants in any way and there would be no white root-to-leaf path. The coefficient of x^t in $x(1+x)^{|V(T')|-1}$ counts the number of ways to color t vertices black, with v being colored black. If, on the other hand, v is colored white, there must be no root-to-leaf path in all of the subtrees rooted at all of its children. We can recursively compute the polynomials $P_u(x)$ with respect to each of v's children u, and multiply them to get the polynomial which counts colorings in this case. ◁

▷ Claim 11. CountBlackWhiteColorings runs in time $\widetilde{O}(|V(T)|^2)$.

Proof. There are $|V(T)|$ polynomial multiplications involved in the total computation, where each polynomial has degree at most $|V(T)|$. Using [18] to bound the running time of polynomial multiplication completes the proof. ◁

CountGoodLabellings $(T=(V,E), v \in V)$
  $P(x) \leftarrow$ CountBlackWhiteColorings (T, v)
  return $\sum_{0 \le t \le |V(T)|} \left(\binom{|V(T)|}{t} - [x^t]P(x) \right) (|V(T)| - 1 - t)!t!$

Armed with CountBlackWhiteColorings, we may now proceed to state complete the algorithm CountGoodLabellings. By Observation 9 and Claim 10, CountGoodLabellings solves CountGoodLabellings and has running time $\widetilde{O}(|V(T)|^2)$.

Back to Computing $\Pr[e \in R(S)]$. We are finally ready to compute the desired quantity, $\Pr[e \in R(S)]$:

(i) Compute T'_e;

(ii) Return CountGoodLabellings$(T'_e, e)/\left(|V(T'_e)|!(m - |V(T'e)|)!\right)$.

Finishing up the proof of Theorem 3 is straightforward and deferred to Appendix B.

3 Adaptive Setting (with Worst-Case Capacities)

We use notation developed in Section 1.1. We want to select k vantage points from V in a given graph $G = (V, E, w)$ with unknown, but fixed, worst-case capacities c. Recall that an algorithm is called (α, β)-instance optimal if it selects at most αk vantage points and reveals at least OPT/β many bottleneck edges, where OPT is the number of edges revealed (after selecting k vantage points) by an algorithm that is also given all the capacities as input. Here, $\alpha \ge 1$ and $\beta \ge 1$.

3.1 Lower bounds

In the following lower bounds, we do not limit the computation time of an algorithm – it can be adaptive, and can do any finite computation between rounds.

▶ **Theorem 12** (Deterministic Algorithm Tradeoff). *For any $1 \le k \le n$, there exists a graph $G = (V, E, w, c)$ for which any deterministic, adaptive, (α, β)-instance optimal algorithm must satisfy $\alpha^2\beta \ge \Omega(n/k)$.*

Proof. We will assume k divides n for simplicity. Let G be the graph with k connected components P_i, $1 \le i \le k$, where each P_i is a path on n/k vertices. Let $\mathcal{A}$ be a deterministic and adaptive algorithm. Consider the following adversarial strategy. For any P_i, whenever $\mathcal{A}$ selects a vantage point v_0^i that is the first vantage point on P_i, the adversary reveals only the two neighboring edges on v, giving them the lowest two capacities out of the $n/k - 1$ edges

in P_i. At any point in the execution of $\mathcal{A}$, let $\{v_0^i, \cdots, v_{j-1}^i\}$ be the vantage points selected on P_i so far, and assume $\mathcal{A}$ selects a new vantage point v_j^i on P_i. If v_j^i is not a neighbor of v_k^i for any $1 \leq k \leq j-1$, the adversary reveals the two edges incident to v_j^i, giving them the next lowest capacities of all the edges revealed in P_i so far. Otherwise, the adversary reveals either one or zero edges (depending on whether one or both neighbors of v_j^i was already selected). We observe that the adversary's strategy is consistent: for any P_i, all edges yet to be revealed have a higher capacity than those revealed.

Let λ_i be the number of vantage points selected on P_i at the end of $\mathcal{A}$. We have that $\sum_{i=1}^{k} \lambda_i = \alpha k$, where α is the resource augmentation factor. The adversary now reveals the remaining edges in the following way. Fix a path P_i. If $\lambda_i = 0$, the adversary gives all edges in P_i decreasing capacities from the first vertex in P_i (and therefore selecting that vertex would have revealed all $n/k - 1$ edges). Otherwise, if $\lambda_i > 0$, there is a contiguous set of $\ell_i = (n/k-1)/(\lambda_i+1) - 2$ edges such that none of the endpoints have been selected by $\mathcal{A}$ as vantage points. The adversary gives them decreasing weights, starting from, say, the first vertex in this set of edges. Selecting this first vertex would have revealed not only the ℓ_i edges with decreasing capacities, but also the two "outer edges", since their capacities are lower. Thus selecting this vertex reveals $\ell_i + 2 = (n/k-1)/(\lambda_i+1)$ edge capacities. We now have that $OPT \geq \sum_{i=1}^{k} (\ell_i + 2)$.

Clearly $\mathcal{A}$ reveals at most $2\alpha k$ edges, at most two per vantage point selected. Thus

$$\beta \geq \frac{\sum_{i=1}^{k} \frac{(n/k-1)}{\lambda_i+1}}{2\alpha k}, \text{ and } \alpha k = \sum_{i=1}^{k} \lambda_i.$$

The second equation implies that there exists a set $S \subset \{1, \cdots, k\}$ such that $|S| = k/2$ and $\lambda_j \leq 2\alpha$ for all $j \in S$. We will use S to lower bound the sum in the first inequality. We have

$$\alpha\beta \geq \frac{\sum_{i=1}^{k} \frac{(n/k-1)}{\lambda_i+1}}{2k} \geq \frac{\sum_{i \in S} \frac{(n/k-1)}{\lambda_i+1}}{2k} \geq \frac{|S| \frac{n/k-1}{2\alpha+1}}{2k} = \frac{k/2}{2k} \cdot \frac{n/k-1}{2\alpha+1} \geq \frac{1}{4} \cdot \frac{n/2k}{3\alpha},$$

and thus $\alpha^2\beta \geq n/24k$, proving the theorem. ◀

▶ Remark. Note that the above theorem is vacuous when $\alpha > \sqrt{n/k}$, as β is always at least 1. We leave open the existence of a lower bound that implies $\beta = \omega(1)$ when $\alpha > \sqrt{n/k}$. As we will see, the trade off in Theorem 12 is tight for trees, so such a lower bound (if it exists) would involve an instance with cycles.

Next, we consider a randomized algorithm $\mathcal{A}$ that selects at most αk vantage points. Given an instance $G = (V, E, w, c)$, different runs of $\mathcal{A}$ may reveal different numbers of edge capacities. We define β for a randomized algorithm to be OPT divided by the expected number of edge capacities revealed by $\mathcal{A}$ over all runs of $\mathcal{A}$ on G (expectation over random coins of $\mathcal{A}$). The proof of the following theorem can be found in Appendix C.

▶ **Theorem 13** (Randomized Algorithm Tradeoff)**.** *For any $1 \leq k \leq n$, there exists a graph $G = (V, E, w, c)$ for which any randomized, adaptive, (α, β)-instance optimal algorithm must satisfy $\alpha\beta \geq \Omega\left(\sqrt{\frac{n}{k}} \frac{1}{(1+\log(n/k))}\right) = \widetilde{\Omega}(\sqrt{n/k})$.*

▶ Remark.

1. As with Theorem 12, note that Theorem 13 is vacuous when $\alpha > \sqrt{n/k}$, as β is always at least 1. We leave open the existence of a lower bound that implies $\beta = \omega(1)$ when $\alpha > \sqrt{n/k}$. As we will see, the tradeoff in Theorem 13 is tight (upto the log factor) for trees, so such a lower bound (if it exists) would involve an instance with cycles.

2. A comparison of the tradeoffs in Theorems 12 and 13 reveals that the tradeoff in Theorem 12 is larger than that in Theorem 13 as long $\alpha < \sqrt{n/k}$ which by the above remark is the setting for Theorem 13. For example, when $\alpha = k = O(1)$, the first tradeoff gives $\beta = \Omega(n)$ whereas the second tradeoff gives $\beta = \Omega(\sqrt{n})$. This indicates that randomization may help. Indeed, it does, as we show next.

3.2 Upper bounds

For general graphs, observe that there is always an algorithm that achieves the tradeoff of $\alpha\beta = O(n)$, simply because $OPT \leq k(n-1)$ (each vantage point reveals at most $n-1$ edge capacities in its shortest path tree), and the trivial algorithm that selects any set of αk vantage points reveals $\Omega(\alpha k)$ many capacities. Obviously, $\alpha\beta = O(n)$ is much worse than the tradeoffs in Theorems 12 and 13. We first show that Theorems 12 and 13 are tight for trees, thus implying that strengthening Theorems 12 and 13 will require instances with cycles.

▶ **Theorem 14** (Tight tradeoffs for trees). *Let $T = (V, E, w)$ be a weighted tree on n vertices with fixed, unknown capacities c on edges.*

1. *For any $1 \leq k \leq n$, there exists a deterministic algorithm $\mathcal{A}_{det}$ that for any given $1 \leq \alpha \leq \sqrt{n/k}$, selects αk vantage points in T such that $\mathcal{A}_{det}$ is (α, β)-instance optimal achieving the exact tradeoff in Theorem 12, $\alpha^2\beta = O(n/k)$.*
2. *For any $1 \leq k \leq n$, there exists a randomized algorithm $\mathcal{A}_{rand}$ that for any given $1 \leq \alpha \leq \sqrt{n/k}$, selects αk vantage points in T such that $\mathcal{A}_{rand}$ is (α, β)-instance optimal with $\alpha\beta = O(\sqrt{n/k})$. This tradeoff is only a factor $\log(n/k)$ worse than that in Theorem 13.*

Proof. Both claims will require the notion of a "cover" S, which we describe first. We take the tree $T(r)$ rooted at an arbitrary vertex r. Take the lowest vertex v with at least $\sqrt{n}$ descendants with v inclusive (i.e., any child of v has fewer than $\sqrt{n}$ descendants). We put v in set S. We remove v and its descendants and repeat until we are left with at most $\sqrt{n}$ vertices. Clearly there are at most $\sqrt{n}$ vertices in S. We call this set S a *cover* of the tree. Sometimes we will need to vary the number of descendants: we will denote by $S(\gamma)$ the cover of size at most n/γ, generated in the same way as above starting with the lowest vertex v with at least γ descendants. The covers satisfy the following lemma.

▶ **Lemma 15.** *For any vertex r' taken as the root of the tree, all but γ vertices are descendants of vertices in $S(\gamma)$.*

Proof. $S(\gamma)$ satisfies the requirement for the root r chosen during the construction of $S(\gamma)$ by design. Now suppose we have a different root $r' \neq r$. We argue that $S(\gamma)$ again covers all but γ vertices in the descendants of $S(\gamma)$ for $T(r')$ rooted at r'. Consider the tree $T(r)$ rooted at r. We run a case study.

If r' is not a descendant of any vertex v in $S(\gamma)$, then all descendants of $S(\gamma)$ in $T(r)$ remain to be descendants of $S(\gamma)$ in tree $T(r')$. Thus we are done.

If r' is a descendant of a vertex v in $S(\gamma)$. Wlog we take v as the lowest such ancestor of r'. We take the child v' of v who is the ancestor of r' (or r' itself). All vertices that are not in the subtree of v' are now descendants of v in $T(r')$. Thus they are "covered". Now we focus on the vertices in the subtree of v' in $T(r)$. Among these vertices, those that are descendants of some other vertex w of $S(\gamma)$ in this subtree will continue to be descendant of w in $T(r')$ – since r' is not in the subtree with root w. Thus the only vertices left would be those that are not in subtree of other vertices in $S(\gamma)$. There can only be at most $\gamma - 1$ vertices – otherwise v' would have been selected to $S(\gamma)$ in the computation of $S(\gamma)$. ◀

Proof of Theorem 14 Claim 1. The algorithm $\mathcal{A}$ selects a cover, with any $r \in T$ as the root in the above cover-selection procedure and $\gamma = n/(\alpha k)$. Thus the size of the cover is at most αk. If the size of the cover is strictly smaller than αk, we select some arbitrary vantage points to make the total number of selected vantage points equal to αk.

Let m be the total number of edge capacities revealed by OPT. Consider a vertex r' chosen by the OPT solution. By Lemma 15, in the tree rooted at r', all but γ vertices of T are descendants of $S(\gamma)$. Thus all the edges that are revealed by r' in the OPT solution, except those that are not descendants of $S(\gamma)$, will be revealed as well by our algorithm $\mathcal{A}$. Since OPT chooses k vantage points, algorithm $\mathcal{A}$ would only miss at most $k\gamma$ edges and thus reveal at least $m - k\gamma$ edges.

Now we do a case analysis. If $m \geq 2k\gamma$, $m - k\gamma \geq m/2$. Thus $\beta \leq m/(m - k\gamma) \leq 2$. $\alpha^2\beta \leq n/k$, since $1 \leq \alpha \leq \sqrt{n/k}$. On the other hand, if $m \leq 2k\gamma$, algorithm $\mathcal{A}$ selects αk vantage points, just the edges incident to these points reveal at least $\alpha k/2$ many edge capacities (in the worst case, these vertices form a matching). Therefore $\beta \leq \frac{m}{\alpha k/2} \leq \frac{2k\gamma}{\alpha k/2} = \frac{4n}{\alpha^2 k}$. This immediately gives $\alpha^2\beta \leq 4n/k$. Thus the exact tradeoff in Theorem 12 is achieved, and this finishes the proof of Claim 1. ◁

Proof of Theorem 14 Claim 2. The randomized algorithm is very simple: it selects a set of αk vantage points uniformly at random (without repetition) from $S(\sqrt{n/k})$. The proof is similar to the deterministic case. Let m be the number of edge capacities revealed in OPT. Let us consider the expected number of edges covered by the αk vantage points. Note that $S(\sqrt{n/k})$ has $\sqrt{nk}$ vertices; if all selected as vantage points, would reveal at least $m - k\sqrt{n/k} = m - \sqrt{nk}$ many edges revealed by OPT. By linearity of expectation the expected number of edges that are revealed by a random choice of αk vantage points from $S(\sqrt{n/k})$ is at least $\frac{\alpha k}{\sqrt{nk}}(m - \sqrt{nk})$. Again we consider two cases. If $m \geq 2\sqrt{nk}$, $m - \sqrt{nk} \geq m/2$. Thus $\beta \leq 2\sqrt{nk}/(\alpha k)$, or, equivalently, $\alpha\beta \leq 2\sqrt{n/k}$. If $m \leq 2\sqrt{nk}$, our algorithm reveals at least $\alpha k/2$ edges since we select αk vantage points. Thus $\beta \leq 2m/(\alpha k) \leq 4\sqrt{(n/k)}/\alpha$. Or, $\alpha\beta \leq 4\sqrt{(n/k)}$. This finishes the proof. ◁

◀

We remark that all algorithms are non-adaptive with polynomial run time. This is slightly surprising, since the lower bounds were on algorithms not limited by computation. We conjecture that while for trees such non-adaptive algorithms suffice, adaptivity may be needed to achieve the tradeoffs (if it is possible at all) for general graphs.

Planar Graphs. We observed in the beginning of this section that for general graphs, the trivial algorithm achieves $\alpha\beta = O(n)$, and if this is tight, a matching lower bound would involve instances with cycles. However, we show next that such an instance cannot be planar. We leave as an interesting open question whether one can design a deterministic algorithm for general graph with $\alpha\beta = o(n)$.

▶ **Theorem 16.** *Given a weighted planar graph $G = (V, E)$ with fixed unknown capacities, and a $1 \leq k \leq n$, there exists a deterministic (α, β)-instance optimal algorithm with $\alpha = O((n/k)^{2/3})$ and $\beta = O(1)$, and hence $\alpha\beta = O((n/k)^{2/3})$.*

3.3 Stronger lower bound on general graph: Multiple shortest paths

In all of the discussion so far, we have assumed that shortest paths are unique. If there are ties and each shortest path tree may break ties independently of other trees, we can have a stronger lower bound of $\beta \geq \Omega(n^{1-\varepsilon})$ when $\alpha = k = 1$, for any $\varepsilon > 0$, between the instance optimal solution and the best solution by any (possibly randomized, adaptive) algorithm.

We first consider a grid graph of $\sqrt{n} \times \sqrt{n}$ vertices. We wrap it around to be a torus – the top boundary is identified as the bottom boundary and the left boundary is identified as the right boundary. The shortest path trees at all vertices, topologically, are identical – each tree includes the horizontal edges on the row containing the root v, and for each vertex with the same y-coordinate with v, there are two chains of length $\sqrt{n}/2$ each. See Figure 2 (left).

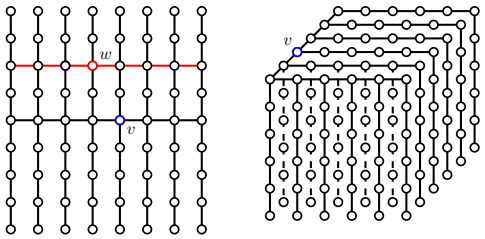


Figure 2 (Left) The shortest path tree rooted at v is shown by edges in black. The edges in red have lower capacity than edges in black. (Right) A 3D grid graph.

Consider one specific vertex v (in blue) and the shortest path tree rooted at v. We take the $n-1$ largest capacity values and assign them on the tree rooted at v, sorted in decreasing capacities in a round robin manner moving away from v. Specifically, the immediate neighbors of v are given the highest capacities, those two hops away from v in the tree are given the next highest capacities, and so on. This way, all edges in the tree rooted at v (called $T(v)$) can be revealed if the optimal solution chooses v. For all other edges we assign the remaining capacity values randomly.

Now consider another vertex w. If w does not have the same y-coordinate as v (not on the same row as y), the shortest path rooted at w has all edges on the same row as w (shown as edges in red in Figure 2 (left)), which have lower capacity than the edges on tree $T(v)$. Therefore, all edges other than the vertical edges on the same column as w cannot be revealed. Thus, w reveals at most $\sqrt{n}$ edges. If w stays on the same row of v, w may be able to reveal up to order n edges but there are only $\sqrt{n}$ such vertices. Since all shortest paths look exactly the same topologically, any algorithm trying to guess the optimal tree $T(v)$ reveals in expectation $O(\sqrt{n})$ edges[1].

This example can be generalized to a d-dimensional cube of $n^{1/d}$ vertices in each dimension. See Figure 2 (right) for an example for $d = 3$. The tree rooted at a vertex v takes all edges with the same coordinate of x_1 as v. For each vertex u of the same x_1 coordinate as v, take the chain of vertices that share the same x_2 coordinate as u, and so on. Use the same allocation of capacities, the optimal tree can reveal $n-1$ edges while any algorithm without the knowledge of capacity distribution reveals $O(n^{1/d})$ edges. This leaves a gap of $O(n^{1-1/d}) = O(n^{1-\varepsilon})$, for any $\varepsilon = 1/d > 0$, between the instance optimal solution and the best oblivious algorithm.

[1] Specifically, suppose an algorithm chooses a vertex v with probability $p(v)$. An adversary would place the vertex on the row with the minimum total probability (which is no greater than $1/\sqrt{n}$) among all possible rows.

4 Open Problems

We conclude the paper with a couple of open problems.

The Adaptive Setting (with Stochastic Capacities). There is an intermediate model that is not addressed in this paper, which is the adaptive setting with stochastic capacities, where the ordering of capacities are a random permutation. There are various notions of stochastic submodularity (e.g. [2, 14]) for adaptive algorithms that admit good approximations via myopic algorithms. Unfortunately, they do not seem to hold for this problem.

Adaptive Setting with Worst-Case Capacities. A notable open problem is to find algorithms that achieve the lower bounds (Theorems 12 and 13) for a general graph. One special parameter range is when $\beta = 1$, i.e., there are k vertices that, if chosen as vantage points, can reveal OPT number of edges. The lower bounds for both deterministic and randomized algorithms suggest that $\alpha = \Omega(\sqrt{n/k})$. Now, can we find $o(n/k)$ vertices as vantage points that reveal at least OPT edge capacities? Our results show that for a tree graph our algorithm with $O(\sqrt{n/k})$ vantage points suffice, and for a planar graph our algorithm with $O((n/k)^{2/3})$ vantage points suffice. We do not know any non-trivial bound of $o(n/k)$ (and $\Omega(\sqrt{n/k})$) for a general graph nor a lower bound example that requires $\omega(\sqrt{n/k})$ vantage points.

References

1 Mark Allman and Vern Paxson. On estimating end-to-end network path properties. *SIGCOMM Comput. Commun. Rev.*, 29(4):263–274, August 1999. doi:10.1145/316188.316230.

2 Arash Asadpour, Hamid Nazerzadeh, and Amin Saberi. Stochastic submodular maximization. In *Internet and Network Economics: 4th International Workshop, WINE 2008, Shanghai, China, December 17-20, 2008. Proceedings 4*, pages 477–489. Springer, 2008. doi:10.1007/978-3-540-92185-1_53.

3 RIPE Atlas. User-defined measurements. https://atlas.ripe.net/docs/getting-started/user-defined-measurements.html.

4 Brice Augustin, Timur Friedman, and Renata Teixeira. Multipath tracing with paris traceroute. In *2007 Workshop on End-to-End Monitoring Techniques and Services*, pages 1–8, May 2007. doi:10.1109/E2EMON.2007.375313.

5 S Banerjee and A K Agrawala. Estimating available capacity of a network connection. In *Proceedings IEEE International Conference on Networks 2000 (ICON 2000). Networking Trends and Challenges in the New Millennium*, pages 131–138, September 2000.

6 Robert Beverly. Yarrp'ing the Internet: Randomized High-Speed active topology discovery. In *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, pages 413–420, New York, NY, USA, November 2016. Association for Computing Machinery. URL: http://dl.acm.org/citation.cfm?id=2987479.

7 Jan Böttger. *Understanding benefits of different vantage points in today's Internet*. PhD thesis, Technische Universität Berlin, 2017.

8 Valentin Burger, Matthias Hirth, Christian Schwartz, Tobias Hoßfeld, and Phuoc Tran-Gia. Increasing the coverage of vantage points in distributed active network measurements by crowdsourcing. In *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, pages 151–161. Springer International Publishing, 2014. doi:10.1007/978-3-319-05359-2_11.

9 Rui Castro, Mark Coates, Gang Liang, Robert Nowak, and Bin Yu. Network tomography: Recent developments. *SSO Schweiz. Monatsschr. Zahnheilkd.*, 19(3):499–517, August 2004.

10 A Coates, A O Hero, III, R Nowak, and Bin Yu. Internet tomography. *IEEE Signal Process. Mag.*, 19(3):47–65, May 2002. doi:10.1109/79.998081.

11 Benoit Donnet, Bradley Huffaker, Timur Friedman, and K C Claffy. Increasing the coverage of a cooperative internet topology discovery algorithm. In *NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, pages 738–748. Springer Berlin Heidelberg, 2007.

12 Benoit Donnet, Philippe Raoult, Timur Friedman, and Mark Crovella. Efficient algorithms for large-scale topology discovery. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '05, pages 327–338, New York, NY, USA, June 2005. Association for Computing Machinery. doi:10.1145/1064212.1064256.

13 Greg N Federickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, December 1987. doi:10.1137/0216064.

14 Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011. doi:10.1613/JAIR.3278.

15 Michael T Goodrich. Planar separators and parallel polygon triangulation (preliminary version). In *Proceedings of the twenty-fourth annual ACM symposium on Theory of Computing*, STOC '92, pages 507–516, New York, NY, USA, July 1992. Association for Computing Machinery. doi:10.1145/129712.129762.

16 Cesar D Guerrero and Miguel A Labrador. On the applicability of available bandwidth estimation techniques and tools. *Comput. Commun.*, 33(1):11–22, January 2010. doi:10.1016/J.COMCOM.2009.08.010.

17 K Harfoush, A Bestavros, and J Byers. Measuring bottleneck bandwidth of targeted path segments. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, volume 3, pages 2079–2089 vol.3, March 2003.

18 David Harvey and Joris Van Der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193(2):563–617, 2021.

19 Thomas Holterbach, Emile Aben, Cristel Pelsser, Randy Bush, and Laurent Vanbever. Measurement vantage point selection using a similarity metric. In *Proceedings of the Applied Networking Research Workshop*, ANRW '17, pages 1–3, New York, NY, USA, July 2017. Association for Computing Machinery. doi:10.1145/3106328.3106334.

20 Yuchen Huang, Michael Rabinovich, and Rami Al-Dalky. FlashRoute: Efficient traceroute on a massive scale. In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, pages 443–455, New York, NY, USA, October 2020. Association for Computing Machinery. doi:10.1145/3419394.3423619.

21 Y. Hyun and K. Claffy. Archipelago measurement infrastructure. http://www.caida.org/projects/ark/, 2015.

22 Jordan Jueckstock, Shaown Sarker, Peter Snyder, Panagiotis Papadopoulos, Matteo Varvello, Benjamin Livshits, and Alexandros Kapravelos. The blind men and the internet: Multi-Vantage point web measurements, May 2019. arXiv:1905.08767.

23 Mohammad Karimi, Mario Lucic, Hamed Hassani, and Andreas Krause. Stochastic submodular maximization: The case of coverage functions. *Advances in Neural Information Processing Systems*, 30, 2017.

24 Bo Li, Jingsha He, and Henghua Shi. Improving the efficiency of network topology discovery. In *2008 The 3rd International Conference on Grid and Pervasive Computing - Workshops*, pages 189–194, May 2008. doi:10.1109/GPC.WORKSHOPS.2008.34.

25 Franziska Lichtblau. From the edge to the core : towards informed vantage point selection for Internet measurement studies, 2021.

26 G L Nemhauser, M L Fisher, and L A Wolsey. *An Analysis of Approximations for Maximizing Submodular Set Functions.* Center for Operations Research & Econometrics, 1976.

27 Mahmoud Owais. Traffic sensor location problem: Three decades of research. *Expert Systems with Applications*, 208:118134, 2022. doi:10.1016/j.eswa.2022.118134.

28 R Prasad, C Dovrolis, M Murray, and K Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE Netw.*, 17(6):27–35, November 2003. doi:10.1109/MNET.2003.1248658.

29 Dixon Salcedo, Cesar D. Guerrero, and Roberto Martinez. Available bandwidth estimation tools metrics, approaches and performance. *International J. Commun. Netw. Inf. Secur.*, December 2018.

30 Yuval Shavitt and Udi Weinsberg. Quantifying the importance of vantage point distribution in internet topology mapping (extended version). *IEEE Journal on Selected Areas in Communications*, 29(9):1837–1847, 2011. doi:10.1109/JSAC.2011.111008.

31 Y Vardi. Network tomography: Estimating Source-Destination traffic intensities from link data. *J. Am. Stat. Assoc.*, 91(433):365–377, March 1996.

32 Kévin Vermeulen, Justin P Rohrer, Robert Beverly, Olivier Fourmaux, and Timur Friedman. Diamond-miner: Comprehensive discovery of the internet's topology diamonds. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 479–493, 2020.

33 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, pages 222–227, 1977. doi:10.1109/SFCS.1977.24.

A Proof of hardness

▶ **Theorem 17.** *The decision version of* BottleneckDiscovery *is* NP*-hard.*

Proof. We give a reduction from VertexCover to BottleneckDiscovery. Let $G = (V, E), k$ be an instance of VertexCover. We transform it to the following instance of BottleneckDiscovery: $G = (V, E, w), k$ where the weights w are of the form $1 + 2^{-i}$ for edge e_i so as to ensure unique shortest paths. We claim that G has a vertex cover of size at most k if, and only if, there exists a set of at most k vantage points that reveal $|E|$ edges in expectation (and hence $|E|$ edges no matter what the capacities are).

($\implies$) If G does indeed have a vertex cover C of size k, then setting $S = C$ reveals every edge as a bottleneck (since edges incident to any $s \in S$ are assured to be revealed) no matter what the edge capacities are.

($\impliedby$) If, conversely, G has no vertex cover of size k, then consider any S of size k. Since S is not a vertex cover, there is at least one edge e whose endpoints are not in S. With probability at least $1/m$, $c(e)$ is the highest capacity among all the edges, and e cannot be revealed unless one of its endpoints was selected. The expected number of revealed bottlenecks for S is thus strictly less than $|E|$. ◀

Next, we prove hardness for the adaptive setting with worst-case capacities.

▶ **Theorem 18.** *There does not exist a deterministic adaptive polynomial time algorithm $\mathcal{A}$ satisfying $\mathcal{A}(G, c) = OPT(G, c)$ for all graphs G and all capacities c, unless $P = NP$.*

Proof. In the following, we show that if such an $\mathcal{A}$ exists, then we can solve vertex-cover in polynomial time. Assume we are given a connected graph G on n vertices and m edges, and a number k, and asked if G has a vertex cover of size exactly k.

The idea is to run $\mathcal{A}$ on G, starting from unknown capacities, but reveal the capacities adaptively to $\mathcal{A}$ in such a fashion that $\mathcal{A}$ is left with no choice but to solve the decision version of vertex cover. Let the current set of vantage points selected by $\mathcal{A}$ be $V_i = \{v_1, \cdots, v_i\}$. Say in the next round $\mathcal{A}$ selects a vantage point v. For all the $d_{v,\overline{V_i}}$ edges between v and a vertex not in V_i, we assign them the next highest $d_{v,\bar{V}_i}$ many capacities (from some arbitrarily

selected alphabet) arbitrarily. This results in a consistent set of capacities: any edge capacities not yet revealed must be higher than those already revealed, and our strategy satisfies this invariant.

At the end of the execution of $\mathcal{A}$, we check if the set of vantage points S selected by $\mathcal{A}$ is a vertex cover. If so, we answer YES to the decision version of vertex-cover; otherwise, we answer NO.

Let k^* be the size of a minimum vertex cover in G. If $k \geq k^*$, then $OPT(G, c) = m$ for any permutation of the capacities, in particular the capacities arrived at the end of $\mathcal{A}$'s execution, c_{end}. Therefore, in this case, $\mathcal{A}(G, c_{end}) = m$ also, and that the set S of vantage points selected must be a vertex cover, for if an edge e is not covered by S, then its capacity is larger than all revealed edges, and could not have been revealed to A.

On the other hand if $k < k^*$, then the set of vantage points selected by $\mathcal{A}$ cannot be a vertex-cover by definition of k^*, and so we answer correctly. Hence, the theorem is proved. ◀

B Omitted Proofs: BottleneckDiscovery in the Stochastic, Non-Adaptive Setting

B.1 Submodularity of f

▶ **Lemma 19.** *$f(X)$ is monotone and submodular as a function of X:*

1. *Monotonicity: $f(X \cup \{w\}) \geq f(X)$;*
2. *Submodularity: If $X \subseteq Y$, then $f(X \cup \{w\}) - f(X) \geq f(Y \cup \{w\}) - f(Y)$.*

Proof. We prove each claim separately.

1. Let X_e (resp. X_e^w) be the indicator for e being revealed from selecting X (resp. $X \cup \{w\}$). Then, by linearity, $f(X) = \sum_e \mathbb{E}(X_e) = \sum_e \Pr[X_e = 1]$. It is clear that $\Pr[X_e^w = 1] \geq \Pr[X_e = 1]$, and so f is monotone.
2. Let X_e (resp. $X_e^w, Y_e, Y_e^w, (Y \backslash X)_e, w_e$) be the indicator for e being revealed from selecting X (resp. $X \cup \{w\}, Y, Y \cup \{w\}, Y \setminus X, \{w\}$). Then

$$\begin{aligned}
f(X \cup \{w\}) - f(X) &= \sum_e \Pr[X_e^w = 1] - \Pr[X_e = 1] && \text{(Linearity of expectation)} \\
&= \sum_e \Pr[X_e = 0 \wedge w_e = 1] && \text{(Law of total probability)} \\
&= \sum_e \Pr[X_e = 0 \mid w_e = 1] \Pr[w_e = 1] \\
&\geq \sum_e \Pr[X_e = 0 \wedge (Y \setminus X)_e = 0 \mid w_e = 1] \Pr[w_e = 1] \\
&= \sum_e \Pr[Y_e = 0 \mid w_e = 1] \Pr[w_e = 1] = f(Y \cup \{w\}) - f(Y),
\end{aligned}$$

and so f is submodular. ◀

By the results on maximizing submodular functions [26], incrementally building up a set S by adding the vantage point maximizing $f(S \cup \{w\})$ until $|S| = k$ yields a $(1 - 1/e)$ approximation algorithm.

▶ **Observation 5.** *Let $S' \subseteq S$ be the set of all essential vantage points. Then* $\Pr[e \in R(S)] = \Pr[e \in R(S')]$.

Proof. In short, this follows from Observation 2.

Let $s' \in S'$.

Querying s' can only reveal bottleneck edges in $T_{s'}$, the shortest path tree rooted at s'. Consequently, if $P(s', u)$ does not go through (v, u) and $P(s', v)$ does not go through (u, v), then $\Pr[e \in R(S)] = \Pr[e \in R(S - s')]$.

On the other hand, note that at most one of $P(s', u)$ goes through (v, u) and $P(s'v)$ goes through (u, v). Without loss of generality, say $P(s', u)$ goes through (v, u). Since $s' \in S'$, there must be a vertex $s \in S$ where $s \neq s'$ such that $s \in P(s', u)$. The event that querying s' reveals (v, u) as a bottleneck is a subset of the event that querying s reveals (v, u) hence $\Pr[e \in R(S)] = \Pr[e \in R(S - s')]$.

Finally, observe that a vertex being non-essential only depends on G and e, and since s' was chosen arbitrarily, $\Pr[e \in R(S)] = \Pr[e \in R(S \setminus S')]$. ◀

▶ **Observation 7.** *T_e is a tree, with all its leaves belonging to S. Moreover, T_e can be computed in $\widetilde{O}(m)$ time.*

Proof. T_e being a tree follows immediately since, in our model, shortest paths are unique. That all its leaves are in S follows from Definition 4, the definition of essential vantage points.

To compute T_e, we need only find T_u (resp. T_v), the shortest path tree rooted at u (resp. v), which we can compute in $\widetilde{O}(m)$ time using Dijkstra's algorithm. We can then find T_e by running a breadth first search in T_u and T_v, truncating at

- Subtrees which do not have (u, v) as an ancestor;
- Vertices $s \in S$;

and finally iteratively removing leaves that are not in S. Since $P(s, u) = P(u, s)$, this produces $\bigcup_{s' \in S'} P(s', u) + e$. The running time follows. ◀

▶ **Observation 8.** *e is a bottleneck edge for S if and only if $c(e)$ is the smallest capacity on some (edge-)root to (vertex-)leaf path on T_e.*

Proof. This follows, in short, from Observation 2.

($\implies$) Suppose e is a bottleneck edge for S. Let $s \in S$ be the vantage point that is closest to e, for which e is a bottleneck edge on, say, $P(s, v)$. Since s is the closest vantage point, it is essential and so $P(s, v) \subseteq T_e$. In particular, s is a leaf with $c(e)$ being the smallest capacity on $P(v, s)$.

($\impliedby$) The converse is proved similarly. Suppose for a leaf $s \in S$, the capacity $c(e)$ is smallest in say $P_{T_e}(s, v)$. Then e is a bottleneck edge of $P(s, v) = P_{T_e}(s, v)$. ◀

Proof of Theorem 3. The correctness and approximation factor of GreedyNonAdaptive have been established earlier. For running time, GreedyNonAdaptive takes k rounds, and computing each step takes nm inner-iterations, with each iteration computing $\Pr[e \in R(S)]$ for some $e \in E$ and $S \subseteq S'$. Each such computation takes $\widetilde{O}(m)$ time to compute T'_e (Observation 7) and $\widetilde{O}(n^2)$ time to run CountGoodLabellings. ◀

C Omitted Proofs: Adaptive setting with worst-case capacities

C.1 Lower Bound Proofs in Section 3.1

Proof of Theorem 13. We will first invoke Yao's principle [33], and try to lower bound the performance of a deterministic algorithm on the following distribution of inputs. In fact, our distribution inputs will all have the same underlying graph, but different set of capacities.

We will assume k divides $\sqrt{n}$ for simplicity. Let G be the disconnected graph with $\sqrt{kn}$ components P_i, $1 \leq i \leq \sqrt{kn}$, where each P_i is a path on $\sqrt{n/k}$ vertices. Consider assigning capacities in the following randomized procedure. First, select a random set $S \subset \{1, \cdots, \sqrt{kn}\}$, $|S| = k$. For any $i \in S$, assign the edges in P_i decreasing capacities, starting from the leftmost vertex. We will call these the "good" paths. For any $i \notin S$, assign the edges in P_i a random permutation of integers in the range $[(i-1)\sqrt{n/k}, i\sqrt{n/k})]$. We will call these the "bad" paths. The optimal adaptive algorithm will select the left endpoints of all k good paths and thus reveal roughly $k \cdot \sqrt{n/k} = \sqrt{kn}$ edges.

Our hard distribution $\mathcal{D}$ will be the uniform distribution over all capacities obtained following the above procedure on G. We now upper bound the expected number of edges revealed by any deterministic algorithm $\mathcal{A}$ over the distribution of inputs $\mathcal{D}$. Recall that $\mathcal{A}$ is allowed to select αk many vantage points. First, observe that if $\alpha \geq \sqrt{n/k}$, the theorem is trivially true since $\beta \geq 1$. So we will assume $\alpha < \sqrt{n/k}$, which means that $\alpha k < \sqrt{kn}$, the total number of paths. Thus, $\mathcal{A}$ does not have enough vantage points to cover all paths.

We can assume that $\mathcal{A}$ does not select two vantage points on the same bad path, as this will only boost the chance of $\mathcal{A}$ hitting the good paths. To argue this assumption, first, we boost $\mathcal{A}$ by giving it for free all edge capacities if it selects any (not necessarily the first) vantage point on a good path, so there is never a need to select another vantage point on a good path. If $\mathcal{A}$ selects a vantage point on a bad path, in expectation it reveals $O(\log(\sqrt{n/k})) = O(\log(n/k))$ many edge capacities. This is because on a random permutation of length n there is a decreasing sequence of $O(\log n)$ record-setting edge capacities, in expectation. This follows from Lemma 2: the probability that the ith edge has a capacity lower than the previous $i-1$ edges is at most $1/(i-1)$, and the expected number of edges revealed can be shown to be upper bounded by $2(1 + 1/2 + \cdots + 1/(n/2)) = O(\log n)$. $\mathcal{A}$ does not have enough vantage points to cover all paths, and therefore it is always optimal to not place another vantage point on a bad path, and instead place it on an uncovered path.

We have that the expected number of good paths revealed by $\mathcal{A}$ is $\alpha k(k/\sqrt{kn}) = \alpha k\sqrt{k/n}$, and thus the expected number of good edges revealed is $(\alpha k\sqrt{k/n})(\sqrt{n/k}) = \alpha k$.

Now observe that the expected number of bad edges revealed is $O(\alpha k \log(n/k))$, at most $O(\log(n/k))$ per vantage point on bad path. Thus the expected number of total edges revealed is still $O(\alpha k \log(n/k))$, whereas $OPT \geq \sqrt{kn}$. Thus $\beta = \Omega(\sqrt{kn}/(\alpha k \log(n/k)))$, proving the theorem. ◀

C.2 Upper Bound Proofs in Section 3.2

Proof of Theorem 16. We use the r-division of a planar graph [15, 13]. Specifically, an r-division is a subdivision of G into $O(n/r)$ pieces satisfying three conditions: Each piece has $O(r)$ vertices; Each piece has $O(\sqrt{r})$ boundary vertices (i.e., vertices shared with other pieces); Each piece has $O(1)$ holes (faces of the piece that are not faces of G. Such a r-division can be computed in $O(n)$ time [15].

Suppose the optimal solution chooses k vertices S, our algorithm basically chooses all vertices on the boundary of all pieces in an r-division, for a value r to be decided later. There are $O(n/r) \cdot O(\sqrt{r}) = O(n/\sqrt{r})$ boundary vertices. $\alpha = O(n/(k\sqrt{r}))$.

For any vertex v in a piece R without any vantage point from the optimal solution S, the shortest path from any vantage point $u \in S$ (which is outside the piece) to v has to go through one of the boundary vertices of the piece R, say w. Thus, all edges that are revealed along the path $P(w, v)$, which is a subpath of $P(u, v)$, are also revealed by our algorithm (by w in particular). Therefore the only edges our algorithm may have missed would be the

edges of the shortest path tree rooted at u that stay inside R, for each $u \in S$. There are at most $O(r)$ such edges for each vantage point $u \in S$ and a total of $O(rk)$ such edges for all vertices in S.

Suppose that the optimal solution reveals m edges. We will reveal at least $m - crk$ edges, for some constant c. If $m \geq 2crk$, $m - crk \geq m/2$. Thus $\beta \leq 2$ and $\alpha\beta \leq O(n/(k\sqrt{r}))$. If $m \leq 2crk$, we reveal at least $\Omega(\alpha k)$ edges since we select αk vertices. Thus $\beta \leq O((2crk)/(\alpha k))$. Or, $\alpha\beta \leq O(r)$. Now we take $r = (n/k)^{2/3}$ to balance the two terms. This gives an upper bound for $\alpha\beta = O((n/k)^{2/3})$. ◀

Approximation and Parameterized Algorithms for Covering with Disks of Two Types of Radii

Sayan Bandyapadhyay
Portland State University, OR, USA

Eli Mitchell
Department of Computer Science, Portland State University, OR, USA

Abstract

We study the Discrete Covering with Two Types of Radii problem motivated by its application in wireless networks. In this problem, the goal is to assign either small-range high frequency or large-range low frequency to each access point, maximizing the number of users in high-frequency regions while ensuring that each user is in the range of an access point. Unlike other weighted covering problems, our problem requires satisfying two simultaneous objectives, which calls for novel approaches that leverage the underlying geometry of the problem. In our work, we present two new algorithms: the first is a polynomial-time $(2.5+\epsilon)$-approximation, and the second is an exact algorithm for sparse instances, which is fixed-parameter tractable (FPT) in the number of large-radius disks. We also prove that such an FPT algorithm is impossible for general instances lacking sparsity, assuming the Exponential Time Hypothesis. Before our work, the best-known polynomial-time approximation factor was 4 for the problem.

Our approximation algorithm results from a fine-grained classification of points that can contribute to the gain of a solution. Based on this classification, we design two sub-algorithms with interdependent guarantees to recover the respective class of points as gain. Our algorithm exploits further properties of Delaunay triangulations to achieve the improved bound. The FPT algorithm is based on branching that utilizes the sparsity of the instances to limit the overall search space.

2012 ACM Subject Classification Theory of computation → Computational geometry; Mathematics of computing → Approximation algorithms

Keywords and phrases Covering, Disks, Approximation, FPT

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.7

Funding *Sayan Bandyapadhyay*: The work has been supported by the NSF grant 2311397.

1 Introduction

Set cover is among the most popular optimization problems, where given a ground set of elements and a family of subsets of the ground set, the goal is to find a minimum-sized subfamily of those subsets whose union contains all elements of the ground set. The problem admits a polynomial-time (poly-time) $O(\log n)$-approximation, and this bound is known to be tight [16, 14]. Set cover has been studied in various restricted settings with the goal of achieving better approximation factors. One such interesting setting is *geometric covering*, where elements are points in a geometric space, and subsets are induced by geometric objects such as disks and squares. Such restricted covering problems can admit poly-time constant-approximations [11, 9] or even approximation schemes [17, 27, 10], as one can exploit the natural constraints imposed by geometric spaces and objects. Indeed, this turned out to be a very popular area of research, as many practical problems can be described in geometric terms, e.g., in areas such as sensor networks, computational biology, image processing, and VLSI design [29, 6, 15, 13, 17].

© Sayan Bandyapadhyay and Eli Mitchell;
licensed under Creative Commons License CC-BY 4.0

While several formulations of geometric set cover are now well-studied, their real applications are sometimes hindered by practical constraints. There exist well-established techniques like local search [27, 19, 7, 1] to compute near-optimal solutions to fundamental geometric problems, but real-life constraints can invalidate the assumptions needed by these approaches. In this work, we study one such problem motivated by applications in wireless and sensor networks, which was introduced by Maheshwari et al. [22]. Informally, given a set of access points and users, the goal is to assign either low-range high frequency (i.e., high speed) or large-range low frequency to each access point, maximizing the number of users in high-frequency range while ensuring that each user is in the range of at least one access point of either kind.

We use the following notation to define the problem. For any point x and a real $\rho > 0$, let $D(x, \rho)$ denote the closed disk with center x and radius ρ. In Discrete Covering with Two Types of Radii (DC-2), we are given two point sets P (of n "users") and $A = \{a_1, a_2, \ldots, a_m\}$ (of m "access points") in the plane, and two real numbers ρ_1 and ρ_2, such that $0 < \rho_1 < \rho_2$ and $P \subseteq \bigcup_{i=1}^{m} D(a_i, \rho_2)$. The goal is to select for each $i = 1, 2, \ldots, m$, a value $r_i \in \{\rho_1, \rho_2\}$ such that $P \subseteq \bigcup_{i=1}^{m} D(a_i, r_i)$ and the *gain*, i.e., the size of the set $\{p \in P :$ there is an index $1 \leq i \leq m$ such that $r_i = \rho_1$ and $p \in D(a_i, r_i)\}$ is maximized. Thus, we want to select for each $1 \leq i \leq m$, either the *small* disk (of radius ρ_1) centered at a_i or the *large* disk (of radius ρ_2) centered at a_i, such that the set P is covered by the chosen disks and the number of points in P each of which is contained in at least one chosen small disk is maximized.

The problem is known to be NP-hard [22] similar to most of the covering problems with disks. Bandyapadhyay et al. [2] designed a poly-time 4-approximation for DC-2. In their approach, they converted the problem to a weighted covering problem with one type of radii (large) by assigning unit weight for each user point contained in a small disk, to a representative large disk. To solve this covering problem, they compute the Delaunay triangulation of the large disk centers. In particular, this triangulation is a planar graph, and hence can be colored by 4 colors. Consequently, a vertex cover of weight at most $\frac{3}{4}$ of the total weight can be computed, and the corresponding large disks are shown to cover all user points. Thus, at least $\frac{1}{4}$ of the total weight of the large disks remains unused while constructing the cover, which leads to the 4-approximation. Obtaining an improved approximation factor remained an interesting open question, as many covering problems involving unit disks admit PTASes [27, 21, 10].

As mentioned in [2], DC-2 can be interpreted as a combination of two problems: Minimum Weight Unit Disk Cover (WUDC) and Maximum Coverage with Unit Disks (MCUD). In WUDC, the goal is to select a subset of unit disks that cover all input points such that the sum of the weights of the selected disks is minimized. In MCUD, for a given k, the goal is to select a subset of k unit disks that cover the maximum number of points possible. Both of these problems are well-studied in computational geometry.

Mustafa and Ray [27] developed a PTAS for WUDC when all weights are equal, i.e., for the unweighted version. Their approach, which relies on a local search scheme, is inadequate for addressing weighted instances. Finding the optimal cover even for weighted unit disks has long been known to be strongly NP-hard [18]. Regardless, approaches have been developed in the last two decades for computing approximate solutions for weighted covering problems for broad classes of objects, such as those with low union complexity [8, 11, 30, 4]. Chan et al. [9] designed constant approximations for covering with weighted disks. Subsequently, Li and Jin [21] designed a PTAS for WUDC, utilizing a shifting strategy [18] that partitions the plane into squares and addresses a restricted version of the problem within each square. We

note that this approach cannot be used directly for DC-2 when users are covered by multiple small disks, because the assignment of user weights to small disks cannot be done arbitrarily, and so the weight of a small disk in the optimal solution is an unknown in DC-2.

Chaplick et al. [10] designed a local search PTAS for the MCUD problem. It follows that for some k, there are k small disks in any optimal solution to DC-2 such that they cover the points that contribute to the optimal gain. However, the best set of k small disks that maximizes the coverage might not lead to a feasible solution, as the remaining $n - k$ large disks might not cover the points not in the union of the chosen small disks. Dealing with two different but dependent tasks is the main challenge in tackling DC-2. The need to satisfy one objective while optimizing another necessarily means that any decision made at the local scope can have global consequences. For example, selecting one access point for a small disk may force a neighbor to be selected as a large disk to maintain coverage, and such a dependency may further propagate to disks that are far away. As such, any solution for this kind of constrained problem must take into account its global properties while making any decision to achieve good approximation.

We also study DC-2 from the perspective of parameterized complexity. Designing parameterized algorithms for geometric problems is an active area of research that is becoming increasingly popular. A problem is called fixed-parameter tractable (FPT) in a parameter k if it can be solved in time $f(k) \cdot n^{O(1)}$ for a computable function f, where n is the input size. A known fact is that if a problem with parameter k is W[1]-hard, then it cannot be FPT in k [12]. Marx [23] considered the problem of covering with unit squares. Here the size of a cover is a natural parameter. He proved that the problem is W[1]-hard. One consequence of this is that the optimization version does not admit an Efficient PTAS or EPTAS [23], i.e., the $n^{O(1/\epsilon^2)}$-time PTAS due to Hochbaum and Maass [17] cannot be improved to an EPTAS. In a different work, Marx [24] studied the parameterized complexity of common optimization problems in geometric intersection graphs and obtained a mixed set of algorithmic and hardness results. In a follow-up work, Marx and Pilipczuk [26] designed a framework for designing sub-exponential ($n^{O(\sqrt{k})}$) time algorithms for a wide range of geometric facility location problems. Applying this framework, they designed an $n^{O(\sqrt{k})}$-time algorithm for covering with unit disks. This result is also tight, assuming ETH [24] (also see this survey [25]). The framework is based on Voronoi diagram separators. We note that this framework is not useful for DC-2, again due to the mixed nature of the problem. Kowalska and Pilipczuk [20] recently studied parameterized approximation algorithms for covering with line segments. Lastly, Banik et al. [3] studied the parameterized complexity of a conflict-free version of geometric covering.

1.1 Our results and contributions

In our work, we design for any $\epsilon > 0$, an $(n + m)^{O(1/\epsilon^9)}$-time approximation algorithm for DC-2 with the improved approximation factor of $(2.5 + \epsilon)$. The algorithm is based on a fine-grained classification of user points that may contribute to the gain of a solution. As mentioned before, the best-known $\frac{1}{4}$ factor for DC-2 is based on a weight assignment scheme that assigns 1 weight for each user point contained in a small disk, to a large disk. Then using the Delaunay triangulation of the large disk centers, a cover by large disks can be computed where at least $\frac{1}{4}$ of the total weight remains unused. In our work, we identify a subset of user points for which $\frac{1}{2}$ of the total weight can be saved by using the same Delaunay triangulation based algorithm, but along with a new weight assignment scheme that exploits further properties of the triangulation. Specifically, this is the subset of user points that are contained in 2 or more small disks. Unfortunately, the $\frac{1}{2}$ weight saving in this way is

not possible for the other user points contained in exactly 1 small disk and we still gain $\frac{1}{4}$ of the total weight for them. However, in a separate approach, we show that the points of the latter type that contribute to an optimal solution can be recovered almost exactly by treating them as the weights on the large disks and solving an instance of WUDC. Hence, our main algorithm runs two subalgorithms and returns the larger gain. Our analysis shows that this is at least $\frac{1}{2.5+\epsilon}$ times the optimal gain, which improves on the previous factor of $\frac{1}{4}$.

We also study the parameterized complexity of the problem, with k being the number of large disks to be selected. This is a natural choice of parameter, as in practice, one would like to open as few large disks as possible to increase the overall quality of the coverage. In particular, the large disks correspond to low-speed antennas, and hence, the selection of too many such disks might slow down the data transfer, leading to a subpar experience for customers. We first show that the problem is W[1]-hard when k is the number of large disks to be selected. In fact, we prove that assuming ETH, there is no $f(k) \cdot (n+m)^{o(\sqrt{k})}$ time algorithm for the problem for any computable function f. In stark contrast, we show that the problem is fixed-parameter tractable (FPT) in k if each point is contained in, at most, a constant number of large disks. We define the notion of s-sparse instances (see Section 3 for a formal definition) and show that the problem can be solved in $s^{O(k)}(m+n)^{O(1)}$ time on any such instance, where $s \leq m$. This result indicates that the W[1]-hardness arises solely from the absence of sparsity in the instances. We note that the problem remains NP-hard for $O(1)$-sparse instances [22].

While the $(2.5+\epsilon)$-approximation works for any instance of DC-2, in practice, the FPT algorithm may yield vastly better run time when points are low-depth. DC-2 is directly motivated by wireless networks, where low-depth points frequently occur in certain environments like sparsely-populated cities. As such, we chose to describe both algorithms with the hope of improving the variety of results available for practical applications.

Organization. The approximation algorithm is described in Section 2. In Section 3, we describe the FPT algorithm for sparse instances. The hardness result appears in Section 4.

2 An Approximation Algorithm for DC-2

Our main algorithm is an aggregate of two separate algorithms. The first is based on the Delaunay triangulation of large disk centers. The second algorithm is based on a reduction to the Minimum Weight Unit Disk Cover problem. Thereafter, we combine the two algorithms to obtain our main algorithm which has the desired approximation guarantee.

A set of disks D' is said to *cover* a set of points P' if the union of the disks in D' contains the points in P'. Let S (resp. L) be the set of small (resp. large) disks centered at the access points in A. Let $V \subseteq P$ such that each point in V does not lie in any (small) disks of S. Wlog, we assume that each point of V is in at least two disks of L. It must be in at least one disk, otherwise there is no feasible cover. Also, if it is in exactly one disk, this disk will always be part of the solution. Let $N \subseteq P$ such that each point in N lies in at least one (small) disk of S. Thus, (V, N) is a partition of P. Let $N_1 \subseteq N$ such that each point in N_1 lies in exactly one disk of S. Similarly, let $N_2 \subseteq N$ such that each point in N_2 lies in at least two disks of S. Thus, (N_1, N_2) is a partition of N. Let $n_1 = |N_1|$ and $n_2 = |N_2|$. Fix an optimal solution of DC-2, i.e., the selected subset of disks of $S \cup L$. Let OPT be the set of points in N covered by the optimal solution. Let $\text{OPT}_1 = N_1 \cap \text{OPT}$ and $\text{OPT}_2 = N_2 \cap \text{OPT}$. Also, fix an error parameter $\epsilon > 0$.

▶ **Observation 1.** *The gain of any solution is at most* $n_1 + n_2$.

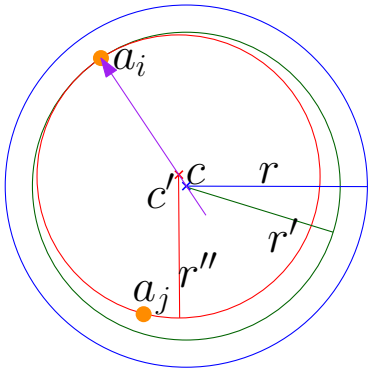


Figure 1 Figure illustrating the proof of Lemma 3.

As each point in N is contained in a small disk $D(a_i, \rho_1)$ and also in the corresponding large disk $D(a_i, \rho_2)$, the following observation holds.

▶ **Observation 2.** *Consider any subset $D' \in S \cup L$ where for each $a_i \in A$, either $D(a_i, \rho_1)$ or $D(a_i, \rho_2)$ is in D'. Then D' is a cover for N.*

2.1 The first algorithm

Let $\mathcal{T}$ be the Delaunay triangulation of the points in A. In particular, $\mathcal{T}$ is a connected planar graph where the vertices are the points in A and each edge is a segment connecting two points of A. We prove the following lemma.

▶ **Lemma 3.** *For any disk $D(c, r)$ in the plane that contains at least two points of A, there exist two points a_i, a_j in $A \cap D(c, r)$ and a path π in $\mathcal{T}$ between a_i and a_j such that π is contained in $D(c, r)$.*

Proof. Continuously decrease the radius of $D(c, r)$ until it contains a point, say a_i, of A on its boundary. This process gives us a disk $D(c, r') \subseteq D(c, r)$ that contains all the points in $D(c, r) \cap A$. If there is no other point $a_j \in A$ than a_i on the boundary of $D(c, r')$, shrink $D(c, r')$ along $\overrightarrow{ca_i}$ keeping the point a_i fixed on its boundary until it contains another point on its boundary and denote it as a_j. The resulting disk $D(c', r'') \subseteq D(c, r')$ contains all the points in $D(c, r) \cap A$ and has a_i and a_j both on its boundary (Figure 1). We prove that there is a path π in $\mathcal{T}$ between a_i and a_j such that π is contained in $D(c', r'')$, proving the lemma. The proof is similar to that of Theorem 1 [5], adapted in our context.

The proof is by induction on the number of points in $D(c', r'') \cap A$. If there is no point of A in the interior of $D(c', r'')$, then $\{a_i, a_j\}$ is an edge of $\mathcal{T}$ and we are done. Otherwise, suppose there is a point a_l in the interior of $D(c', r'')$. Shrink $D(c', r'')$ along $\overrightarrow{c'a_i}$, keeping the point a_i fixed on its boundary until it contains a_l on its boundary. This process gives us a disk $D(c_1, r_1) \subseteq D(c', r'')$ that contains a_i and a_l on its boundary and does not contain a_j. Similarly, shrink $D(c', r'')$ along $\overrightarrow{c'a_j}$, keeping the point a_j fixed on its boundary until it contains a_l on its boundary. This process gives us a disk $D(c_2, r_2) \subseteq D(c', r'')$ that contains a_j and a_l on its boundary and does not contain a_i. By induction, there is a path π_1 in $\mathcal{T}$ between a_i and a_l such that π_1 is contained in $D(c_1, r_1) \subseteq D(c, r)$ and a path π_2 in $\mathcal{T}$ between a_j and a_l such that π_2 is contained in $D(c_2, r_2) \subseteq D(c, r)$. Merging the two paths π_1 and π_2, we obtain the desired path π. ◀

▶ **Corollary 4.** *For any point $p \in V$, there exist two points $a_i, a_j \in A$ such that the edge $\{a_i, a_j\}$ is in $\mathcal{T}$ and p is in both large disks $D(a_i, \rho_2)$ and $D(a_j, \rho_2)$.*

Proof. Consider the disk $D(p, \rho_2)$. As p is in at least two large disks, there must be at least two points of A in this disk. Then by Lemma 3, there are two points a_l, a_t in $A \cap D(p, \rho_2)$ and a path π in $\mathcal{T}$ between a_l and a_t such that π is contained in $D(p, \rho_2)$. Consider any edge $\{a_i, a_j\}$ on π. Then as a_i, a_j are in $D(p, \rho_2)$, p is in both large disks $D(a_i, \rho_2)$ and $D(a_j, \rho_2)$. ◀

▶ **Corollary 5.** *For any point $p \in N_2$, there exist two points $a_i, a_j \in A$ such that the edge $\{a_i, a_j\}$ is in $\mathcal{T}$ and p is in both small disks $D(a_i, \rho_1)$ and $D(a_j, \rho_1)$.*

Proof. Consider the disk $D(p, \rho_1)$. As p is in at least two small disks, there must be at least two points of A in this disk. Then by Lemma 3, there are two points a_l, a_t in $A \cap D(p, \rho_1)$ and a path π in $\mathcal{T}$ between a_l and a_t such that π is contained in $D(p, \rho_1)$. Consider any edge $\{a_i, a_j\}$ on π. Then as a_i, a_j are in $D(p, \rho_1)$, p is in both small disks $D(a_i, \rho_1)$ and $D(a_j, \rho_1)$. ◀

Now, we describe the pseudocode of the algorithm, which is self-explanatory.

Algorithm 1 Weighted-Extraction.

Input: The sets A, P, N_1, N_2, small radius ρ_1 and large radius ρ_2.
Output: A set *sol* of large and small disks covering P.

Compute the Delaunay triangulation $\mathcal{T}$ of the points in A.
Compute a valid four coloring of the vertices of $\mathcal{T}$.
for $a_i \in A$ **do**
    $w_i \leftarrow 0$ ▷ Intialize the weights
    for $p_1 \in N_1$ **do**
        if $p_1 \in D(a_i, \rho_1)$ **then** $w_i \leftarrow w_i + 1$
for $p_2 \in N_2$ **do**
    Compute two points a_i, a_j with the properties described in Corollary 5.
    $w_i \leftarrow w_i + 1$
    $w_j \leftarrow w_j + 1$
Compute the color class C of vertices (access points) of $\mathcal{T}$ with maximum weight (w.r.t. $w_i : a_i \in A$).
$sol \leftarrow \emptyset$
for $a_i \in A$ **do**
    if $a_i \in C$ **then**
        $sol \leftarrow sol \cup D(a_i, \rho_1)$
    else
        $sol \leftarrow sol \cup D(a_i, \rho_2)$
return *sol*

The four coloring of a planar graph can be computed in polynomial time, for example using the algorithm in [28]. Hence, the algorithm runs in polynomial time.

▶ **Lemma 6.** *The solution sol computed by* Weighted-Extraction *covers all points of P.*

Proof. Note that we pick either $D(a_i, \rho_1)$ or $D(a_i, \rho_2)$ in the solution for each access point a_i. Thus, by Observation 2, the points in N are covered by the selected disks. Now, consider any point $p \in V$. We prove that p is covered in the solution. By Corollary 4, we know that there

exist two points $a_i, a_j \in A$ such that the edge $\{a_i, a_j\}$ is in $\mathcal{T}$ and p is in both large disks $D(a_i, \rho_2)$ and $D(a_j, \rho_2)$. Note that the points in C form an independent set of the graph $\mathcal{T}$ by all sharing the same color. Hence, the set of points $A \setminus C$ is a vertex cover of $\mathcal{T}$. It follows that at least one of a_i and a_j is in $A \setminus C$. As we pick large disks for all access points in $A \setminus C$, p must be covered by $D(a_i, \rho_2)$ or $D(a_j, \rho_2)$ in the solution. ◀

We need the following lemma to analyze the gain of the solution.

▶ **Lemma 7.** *Let W be the total weight of the points in C. Then the set of small disks centered at the access points in C contains at least W points of N.*

Proof. First, note that the points in C form an independent set of the graph $\mathcal{T}$. Now, consider any point $p \in N$. We claim that it contributes 1 unit of weight to at most one point of C. If p is in N_1, by our construction, it contributes either 1 or 0 depending on whether the center of the unique small disk that contains p is in C or not. If p is in N_2, then by our construction it contributes 1 unit each to two access points a_i and a_j. However, the edge $\{a_i, a_j\}$ is contained in $\mathcal{T}$. As C is an independent set, it can contain at most one of a_i and a_j. Hence, in this case, p contributes at most 1 unit as well. It follows that each 1 unit of weight of C comes from a unique point of N. Now, by our construction, 1 unit of weight for a point p of N is assigned to an access point a_i only if p is in the small disk $D(a_i, \rho_1)$. Hence, for each unit of weight of the points in C, there is a unique point in N that is contained in a small disk centered at a point of C. So, the number of points contained in the set of small disks centered at the access points in C is at least W. ◀

We note that in the above, the number of points covered by small disks may be more than W, as the weight of each point contained in a small disk centered at a point of C might not be assigned to the points of C. Hence, W is only a lower bound.

▶ **Corollary 8.** *The solution computed by* Weighted-Extraction *has a gain of at least* $\mathrm{OPT}_1/4 + \mathrm{OPT}_2/2$.

Proof. First, note that by our construction, the sum of the weights of all points in A is $n_1 + 2n_2$. Now, as we pick C to be the maximum weight color class, the total weight of the points in C must be at least the average weight, which is $(n_1 + 2n_2)/4 = n_1/4 + n_2/2$. By Lemma 7, the gain of the solution, that is the number of points of N covered by the small disks with centers at points in C, is at least $n_1/4 + n_2/2$. As $\mathrm{OPT}_1 \leq n_1$ and $\mathrm{OPT}_2 \leq n_2$, the gain is at least $\mathrm{OPT}_1/4 + \mathrm{OPT}_2/2$. ◀

2.2 The second algorithm

This algorithm has three steps. In the first step, we construct an instance of the Minimum Weight Unit Disk Cover problem from the given instance of DC-2.

In Minimum Weight Unit Disk Cover (WUDC), given a set $\mathcal{D}$ of unit disks along with a weight function $w : \mathcal{D} \to \mathbb{R}^+$ and a set T of points in the plane, the goal is to find a subset $\mathcal{D}' \subseteq \mathcal{D}$, such that $\mathcal{D}'$ covers T and the sum of the weights of the disks in $\mathcal{D}'$ is minimized. We refer to this sum of weights as the *cost* of the solution $\mathcal{D}'$.

The construction of the instance $\mathcal{I}$ of WUDC is as follows. T is set to be V. $\mathcal{D}$ is the set of all large disks in L. The weight of each large disk $D(a_i, \rho_2)$ is the number of points of N_1 in the corresponding small disk $D(a_i, \rho_1)$, i.e., $w(D(a_i, \rho_2)) = |N_1 \cap D(a_i, \rho_1)|$.

In the second step of the algorithm, a $(1+\epsilon)$-approximate solution $\mathcal{D}'$ to WUDC on $\mathcal{I}$ is computed using the PTAS in [21], which runs in time $|\mathcal{D}|^{O(1/\epsilon^9)}$. In the third step, a solution to DC-2 is computed as follows. First, add all the large disks in $\mathcal{D}'$ to the solution set. Next, for each large disk $D(a_i, \rho_2)$ in $\mathcal{D} \setminus \mathcal{D}'$, add the corresponding small disk $D(a_i, \rho_1)$ to the solution set.

The above algorithm runs in $(n+m)^{O(1/\epsilon^9)}$ time. Next, we analyze this algorithm. Let OPT' be the optimal cost of $\mathcal{I}$.

▶ **Observation 9.** *The solution computed by the above algorithm covers all points of P.*

Proof. We pick either $D(a_i, \rho_1)$ or $D(a_i, \rho_2)$ in the solution for each access point a_i. Thus, by Observation 2, the points in N are covered by the selected disks. We also see that the solution $\mathcal{D}'$ to WUDC covers the points in T. As we include the (large) disks of $\mathcal{D}'$ in the DC-2 solution and $T = V$, the points of V are also covered. ◀

▶ **Lemma 10.** OPT' *is at most* $n_1 - \mathrm{OPT}_1$.

Proof. Consider the optimal solution of DC-2. The solution covers all points of P and in particular the points of V. As each point of V is not in any small disk of S, it must be covered by a selected large disk. Thus, the set of selected large disks, say L', is a valid cover for the points of V, and so is a feasible solution to $\mathcal{I}$. We show that the cost of this solution is $n_1 - \mathrm{OPT}_1$, proving the lemma.

Now, the gain of the DC-2 solution from the points in N_1 is OPT_1. Thus the union of small disks that are not selected (corresponding to the large disks in L') contains $n_1 - \mathrm{OPT}_1$ points of N_1. By construction of $\mathcal{I}$, these points are the only points that are assigned to the large disks in L'. Hence, the cost of L', that is the sum of the weights of the large disks in L' is $n_1 - \mathrm{OPT}_1$. ◀

By the above lemma and the fact that $\mathcal{D}'$ is a $(1+\epsilon)$-approximate solution, we have the following observation.

▶ **Observation 11.** *The cost of $\mathcal{D}'$ is at most* $(1+\epsilon)\ \mathrm{OPT}' \leq (1+\epsilon)(n_1 - \mathrm{OPT}_1)$.

▶ **Lemma 12.** *The gain of the DC-2 solution computed by the second algorithm is at least* $(1+\epsilon)\mathrm{OPT}_1 - 4\epsilon \cdot \mathrm{OPT}$.

Proof. Note that the gain is at least the number of points of N_1 in the small disks that are not selected in the DC-2 solution. Equivalently, the gain is at least the sum of the weights of the large disks that are in $\mathcal{D} \setminus \mathcal{D}'$. By Observation 11, it follows that the gain is at least

$$\begin{aligned} n_1 - (1+\epsilon)(n_1 - \mathrm{OPT}_1) &= n_1 - n_1 + \mathrm{OPT}_1 - \epsilon n_1 + \epsilon \cdot \mathrm{OPT}_1 \\ &\geq \mathrm{OPT}_1 - 4\epsilon \cdot \mathrm{OPT} + \epsilon \cdot \mathrm{OPT}_1 \\ &= (1+\epsilon)\mathrm{OPT}_1 - 4\epsilon \cdot \mathrm{OPT} \end{aligned}$$

The inequality follows, as by Corollary 8, $\mathrm{OPT} \geq n_1/4 + n_2/2 \geq n_1/4$. Hence, we obtain the desired lemma. ◀

2.3 Combining the two algorithms

Our main algorithm is a combination of the two above-mentioned algorithms. We invoke both on the DC-2 instance and return the solution with the larger gain.

▶ **Lemma 13.** *The gain of the larger-gain solution is at least* $\mathrm{OPT}/(2.5+\epsilon)$.

Proof. Let $0 \leq \alpha \leq 1$ be such that $\text{OPT}_1 = \alpha \cdot \text{OPT}$. By Lemma 8 and 12, the gain of the larger solution is at least

$$\begin{aligned} &\max\{\alpha \cdot \text{OPT}/4 + (1-\alpha)\text{OPT}/2, (1+\epsilon)\alpha \cdot \text{OPT} - 4\epsilon \cdot \text{OPT}\} \\ =&\max\{\text{OPT}/2 - \alpha \cdot \text{OPT}/4, (1+\epsilon)\alpha \cdot \text{OPT} - 4\epsilon \cdot \text{OPT}\} \end{aligned}$$

This function of α is minimized when

$$\begin{aligned} &\text{OPT}/2 - \alpha \cdot \text{OPT}/4 = (1+\epsilon)\alpha \cdot \text{OPT} - 4\epsilon \cdot \text{OPT} \\ \text{Or, } &(5/4+\epsilon)\alpha = 1/2 + 4\epsilon \\ \text{Or, } &\alpha = (2+16\epsilon)/(5+4\epsilon) \end{aligned}$$

Thus, the gain is at least $((1+\epsilon)(2+16\epsilon)/(5+4\epsilon) - 4\epsilon)\text{OPT} = \text{OPT}/(5/2 + O(\epsilon))$. By scaling ϵ by a constant, we obtain the desired bound. ◀

We state our main result of this section in the following theorem.

▶ **Theorem 14.** *There is an* $(n+m)^{O(1/\epsilon^9)}$*-time* $(2.5+\epsilon)$*-approximation algorithm for* DC-2.

3 An FPT Algorithm for Sparse Instances

The parameterized version of DC-2 includes a fixed parameter k, where in addition to the original goal, we are assured that $r_i = \rho_2$ for at most k access points a_i. We refer to this problem as Par-DC-2. Here, we reuse the notation from the previous section. Let OPT be the gain of any optimal solution. Recall the sets of small disks S and large disks L, and the sets of vulnerable points V and non-vulnerable points N. An instance of Par-DC-2 is called *s-sparse* for an integer $s \geq 2$ if each point of V is in at most s large disks of L. We note that in such an instance, a point of N can lie in more than s large (or small) disks. We prove that Par-DC-2 is fixed-parameter tractable in $k+s$.

For a set $S' \subseteq S$, let the *gain of* S' be the number of points of N contained in the disks of S'. For a subset of large disks $L' \subseteq L$, let $\text{sm}(L')$ be the set of corresponding small disks.

Next, we describe our algorithm. Our algorithm is based on a recursive branching strategy. Algorithm 2 shows the pseudo-code of a recursive subroutine that is the main procedure used by our algorithm. The subroutine Recursive-Branching(V', L', l) takes as input a subset of points $V' \subseteq V$ that are yet to be covered, a subset $L' \subseteq L$ of disks already chosen to cover, and an integer $l > 0$ that denotes the depth of recursion. The subroutine returns a True/False flag. If the flag is True, it also outputs a set of k large disks $L'_1 \supset L'$ such that $L'_1 \setminus L'$ covers V', and the gain g corresponding to the solution L'_1, i.e., the gain of $\text{sm}(L \setminus L'_1)$. Inside the subroutine, we first check if V' is empty, and if that is the case, L' itself is returned as the solution along with its gain. If the depth $l = k+1$, but $V' \neq \emptyset$, the False flag is returned, indicating there is no feasible solution along this branch, so it should be pruned here. Otherwise, $l \leq k$ and $V' \neq \emptyset$. In that case, we consider any arbitrary point $p \in V'$. There are at most s large disks in L that cover p. We branch on each such large disk D. In each branch, we remove the points of V' that are covered by D and recurse on the set of remaining points with $L' \cup \{D\}$ being the new set of selected disks and $l+1$ as the new depth. We also keep track of the largest gain solution returned by these calls and return it if the flag counter b is set to True. If b is False, we return the False flag.

The main algorithm makes a call to Recursive-Branching$(V, \emptyset, 1)$. If the call returns True, it outputs the solution where the large disks of L'_1 are selected. Since there are no remaining points in V to cover, the remaining small disks $\text{sm}(L \setminus L'_1)$ may be selected to cover a number of points in N equal to the gain of L'_1. Otherwise, it concludes that there is no feasible solution with k large disks.

Algorithm 2 Recursive-Branching(V', L', l).

Input: A set of points $V' \subseteq V$, a subset $L' \subseteq L$, and an integer $l > 0$.
Output: A flag True/False, and if the flag is True, a set of k large disks $L'_1 \supset L'$ such that $L'_1 \setminus L'$ covers V' and the gain g corresponding to L'_1; otherwise, $\emptyset$ and 0.

if $V' = \emptyset$ **then**
    Let g_1 be the gain of the small disks in $S \setminus \text{sm}(L')$
    return True, L', g_1
if $l = k + 1$ **then**
    return False, $\emptyset, 0$ ▷ Executing this line implies $V' \neq \emptyset$.
Let $p \in V'$ be any arbitrary point.
Let L_p be the set of large disks in L that contain p.
$b \leftarrow$ False
$g \leftarrow 0$
$L'_1 \leftarrow \emptyset$
for each disk $D \in L_p$ **do**
    Let $V_D \subset V'$ be the subset of points of V' not in D
    $b_D, L_D, g_D \leftarrow$ Recursive-Branching$(V_D, L' \cup \{D\}, l + 1)$
    if $b_D =$ True **then**
        $b \leftarrow$ True
        if $g < g_D$ **then**
            $g \leftarrow g_D$
            $L'_1 \leftarrow L_D$
if $b =$ False **then**
    return False, $\emptyset, 0$
else
    return True, L'_1, g

Next, we analyze the algorithm. Consider the recursion tree $\mathcal{R}$ corresponding to Recursive-Branching$(V, \emptyset, 1)$. We define the level of the nodes in $\mathcal{R}$ as follows. The level of the root is 1. The level of any other node is 1 plus the level of its parent. Let L^* denote the large disks selected in the optimal solution. In our analysis, we assume that L^* does not contain any redundant disk, i.e., removing any disk from L^* leaves V uncovered. If it contains such a redundant disk, then we can try our algorithm with a smaller value $k' < k$.

We prove the correctness of Recursive-Branching with the following lemma.

▶ **Lemma 15.** *For each $1 \leq l \leq k$, there is a node u in $\mathcal{R}$ of level l with the property that a call is made at u to* Recursive-Branching$(V_D, L' \cup \{D\}, l + 1)$ *such that $V \setminus V_D$ is the subset of points of V covered by the disks of $L' \cup \{D\}$, and $L' \cup \{D\}$ is a subset of L^* with l disks.*

Proof. We use induction on the level $l \geq 1$. In the base case, consider $l = 1$. The root node at level 1 selects a point $p \in V$ and recurses for each disk $D \in L_p$. Now, p must be covered by a large disk $D^* \in L^*$ that is in L_p. Consider the call Recursive-Branching$(V_{D^*}, \{D^*\}, 2)$ made at the root. By definition, $V \setminus V_{D^*}$ is the set of points in V that are covered by D^*. Hence, the statement is true in the base case. Now, suppose the statement is true for any level $l < k$. We prove that the statement is true for level $l + 1$ as well.

By induction argument, there is a node u' of level $l < k$ that makes a call to Recursive-Branching$(V_D, L' \cup \{D\}, l + 1)$ such that $V \setminus V_D$ is the subset of points of V covered by the disks of $L' \cup \{D\}$, and $L' \cup \{D\}$ is a subset of L^* with $l < k$ disks. Denote the child of u' in $\mathcal{R}$ by u that corresponds to this call and thus receives the set of large disks $L' \cup \{D\} \subset L^*$.

Now, $L' \cup \{D\}$ does not cover V, as otherwise, L^* contains a redundant disk. Thus, we have that $V_D \neq \emptyset$. Suppose p' is the point of V_D arbitrarily chosen during the execution of the call corresponding to u'. Then a recursive call is made for each $D' \in L_{p'}$. Now, by our assumption, p' is not covered by the disks in $L' \cup \{D\} \subset L^*$. Hence, there must be a disk $D^* \in L^* \setminus (L' \cup \{D\})$ that covers p' and is contained in $L_{p'}$. Consider the call Recursive-Branching$(V_{D^*}, L' \cup \{D\} \cup \{D^*\}, l+2)$ made at u. By definition, $V_D \setminus V_{D^*}$ is the subset of points of V_D contained in D^*. As any point in $V \setminus V_D$ is in a disk of $L' \cup \{D\}$, $V \setminus V_{D^*}$ is the exact subset of points of V covered by the disks of $L' \cup \{D\} \cup \{D^*\}$. Also, $|L' \cup \{D\} \cup \{D^*\}| = l+1$. Hence, the statement is also true for u. ◀

The next lemma shows that the algorithm returns a solution with optimal gain.

▶ **Lemma 16.** Recursive-Branching$(V, \emptyset, 1)$ *always returns the True flag and the corresponding solution has the gain value* OPT*.*

Proof. Consider the node in $\mathcal{R}$ of level k that makes a call to Recursive-Branching$(V_D, L' \cup \{D\}, k+1)$ where $L' \cup \{D\} = L^*$. The existence of such a node follows due to Lemma 15. Now, note that L^* covers V, and thus $V_D = \emptyset$. Hence, this call returns True, L^*, OPT to its parent node. Now, inside each call, we keep track of the maximum gain solution returned by all recursive calls and return it to the parent. Hence, the root node always returns True and the gain of the solution returned must be at least OPT. ◀

We will show that the total number of nodes of $\mathcal{R}$ is bounded by $s^{O(k)}$. As the algorithm spends only a polynomial time at each node, the desired result follows. Now, note that at each node, the algorithm can make at most s calls, as each point of V is in at most s disks of L. Moreover, as we prune any branch at level $k+1$, the maximum level is $k+1$. Thus, the total number of nodes is $s^{O(k)}$. We state our result in the following theorem.

▶ **Theorem 17.** Par-DC-2 *can be solved in time* $s^{O(k)}(m+n)^{O(1)}$ *on any* s*-sparse instance.*

4 Parameterized Hardness of DC-2

We give a reduction from Dominating set for unit disks. In this problem, we are given a set of n unit disks (radius 1) $\mathcal{D}$ in the plane and a parameter k, and the goal is to select a subset $\mathcal{D}' \subseteq \mathcal{D}$ of k disks such that for each disk D in $\mathcal{D}$, $D \in \mathcal{D}'$ or there is a disk D' in $\mathcal{D}'$ such that $D \cap D' \neq \emptyset$.

Marx [24] proved that Dominating set for unit disks is W[1]-hard. In fact, he proved that assuming ETH, the problem cannot be solved in $f(k) \cdot n^{o(\sqrt{k})}$ time for any computable function f. Next, we describe our reduction.

Let $\mathcal{I}$ be the given instance of Dominating set for unit disks. Let C be the set of centers of the disks in $\mathcal{I}$. The distance between a point p and a set S is defined as $\min_{q \in S} ||p-q||$. For any point $c \in C$, let δ_c be the minimum distance between c and the disks in $\{D(c',2) \mid c' \in C \text{ and } c \notin D(c',2)\}$ (see Figure 2). Let $\delta_1 = \min_{c \in C} \delta_c$. Also, let $\delta_2 = \min_{c,c' \in C} ||c-c'||$ with $c \neq c'$. Denote $\delta = \min\{\delta_1, \delta_2\}$. Also, let $\epsilon = \delta/6$. We set $A = C$, $P = \{(x+\epsilon, y) \mid (x,y) \in C\}$, $\rho_1 = \epsilon/2$, and $\rho_2 = 2+\epsilon$. Also, we use the same parameter k. Let $\mathcal{I}'$ be the constructed instance of Par-DC-2. The construction takes $|\mathcal{D}|^{O(1)}$ time.

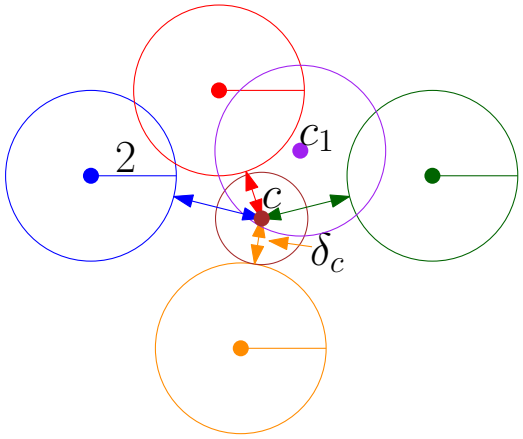


Figure 2 Figure showing the calculation of δ_c: $D(c_1, 2)$ is not considered here, as $c \in D(c_1, 2)$.

▶ **Observation 18.** *For any $c \in C$, the small disk $D(c, \rho_1) \cap P = \emptyset$.*

Proof. Let $c = (x, y)$. Consider any $p = (x_1, y_1) \in P$. First, consider the case when $x_1 = x + \epsilon$ and $y_1 = y$, then $||c - p|| = \epsilon > \rho_1$. Thus, $p \notin D(c, \rho_1)$. Next, consider the case when $x_1 \neq x + \epsilon$ or $y_1 \neq y$. This implies that there is another point $c' = (x', y') \neq c$ such that p was constructed from c', i.e., $x_1 = x' + \epsilon$ and $y_1 = y'$. Now, suppose $||c - p|| \leq \rho_1$. Then, $||c - c'|| \leq ||c - p|| + ||p - c'|| \leq \rho_1 + \epsilon < \delta \leq \delta_2$. In particular, $||c - c'|| < \delta_2$. But, by the definition of δ_2, this is a contradiction. So, $||c - p|| > \rho_1$ in this case, and so $p \notin D(c, \rho_1)$. As the two cases are exhaustive, the observation follows. ◀

The above observation implies that the gain of any solution is 0. Thus, the hardness comes from the ability to cover the points of P using k large disks.

▶ **Lemma 19.** *$\mathcal{I}$ has a dominating set of size k if and only if there is a feasible solution for $\mathcal{I}'$.*

Proof. First, suppose $\mathcal{I}$ has a dominating set of size k. Let C' be the set of centers of the k disks in the dominating set. We claim that the k large disks at the centers in C' cover the points in P. Consider a point $p = (x_1, y_1) \in P$ and let $c = (x, y) \in C$ be such that $x_1 = x + \epsilon$ and $y_1 = y$. We know that there is $c' \in C'$ such that $D(c', 1)$ intersects $D(c, 1)$. Thus $||c - c'|| \leq 2$. So, $||c' - p|| \leq ||c' - c|| + ||c - p|| \leq 2 + \epsilon$. Hence, the large disk at $c' \in C'$ covers p.

Next, suppose there is a feasible solution for $\mathcal{I}'$. Thus, there is a size k subset $C' \subseteq A$ such that the large disks at C' cover the points in P. We claim that the set of disks $\mathcal{D}' \subseteq \mathcal{D}$ at the centers in C' form a dominating set. Consider any disk $D(c, 1) \in \mathcal{D}$. Suppose $D(c, 1) \notin \mathcal{D}'$. Consider the point $p = (x_1, y_1) \in P$ such that $c = (x, y)$, $x_1 = x + \epsilon$ and $y_1 = y$. Then there is $c' \in C'$ such that $D(c', \rho_2)$ covers p. We claim that $c \in D(c', 2)$. Suppose that is not true. Then as per the definition of δ_c, c is at least δ_c distance away from $D(c', 2)$. Thus, $||c - c'|| \geq 2 + \delta_c \geq 2 + \delta = 2 + 6\epsilon$. However, $||c - c'|| \leq ||c - p|| + ||p - c'|| \leq \epsilon + \rho_2 = 2 + 2\epsilon$. We obtain a contradiction, and hence $c \in D(c', 2)$. Equivalently, $D(c, 1) \cap D(c', 1) \neq \emptyset$, and so $D(c', 1)$ dominates $D(c, 1)$. Hence, $\mathcal{D}'$ is a dominating set of size k. ◀

Based on the above discussion, we have the following theorem.

▶ **Theorem 20.** Par-DC-2 *is* W[1]*-hard. Moreover, assuming* ETH, Par-DC-2 *cannot be solved in $f(k) \cdot (n + m)^{o(\sqrt{k})}$ time for any computable function f.*

References

1 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.

2 Sayan Bandyapadhyay, Anil Maheshwari, Sasanka Roy, Michiel Smid, and Kasturi R. Varadarajan. Geometric covering via extraction theorem. 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 7:1–7:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ITCS.2024.7.

3 Aritra Banik, Fahad Panolan, Venkatesh Raman, Vibha Sahlot, and Saket Saurabh. Parameterized complexity of geometric covering problems having conflicts. *Algorithmica*, 82(1):1–19, 2020. doi:10.1007/S00453-019-00600-W.

4 Nikhil Bansal and Kirk Pruhs. Weighted geometric set multi-cover via quasi-uniform sampling. *Journal of Computational Geometry*, 7(1):221–236, 2016. doi:10.20382/JOCG.V7I1A11.

5 Ahmad Biniaz. Plane hop spanners for unit disk graphs: Simpler and better. *Computational Geometry: Theory and Applications*, 89, 2020. doi:10.1016/J.COMGEO.2020.101622.

6 Lorna Booth, Jehoshua Bruck, Massimo Franceschetti, and Ronald Meester. Covering algorithms, continuum percolation and the geometry of wireless networks. *The Annals of Applied Probability*, 13(2):722–741, 2003.

7 Norbert Bus, Shashwat Garg, Nabil Mustafa, and Saurabh Ray. Improved local search for geometric hitting set. In *Proc. of the 32st International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2015. doi:10.4230/LIPIcs.STACS.2015.184.

8 Timothy M. Chan and Elyot Grant. Exact algorithms and apx-hardness results for geometric packing and covering problems. *Computational Geometry*, 47(2, Part A):112–124, 2014. Special Issue: 23rd Canadian Conference on Computational Geometry (CCCG11). doi:10.1016/j.comgeo.2012.04.001.

9 Timothy M. Chan, Elyot Grant, Jochen Könemann, and Malcolm Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1576–1585. SIAM, 2012. doi:10.1137/1.9781611973099.125.

10 Steven Chaplick, Minati De, Alexander Ravsky, and Joachim Spoerhase. Approximation schemes for geometric coverage problems. In *26th Annual European Symposium on Algorithms (ESA 2018)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.17.

11 Kenneth L. Clarkson and Kasturi R. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete & Computational Geometry*, 37(1):43–58, 2007. doi:10.1007/S00454-006-1273-8.

12 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.

13 Ling Ding, Weili Wu, James Willson, Lidong Wu, Zaixin Lu, and Wonjun Lee. Constant-approximation for target coverage problem in wireless sensor networks. In *2012 Proceedings IEEE INFOCOM*, pages 1584–1592. IEEE, 2012. doi:10.1109/INFCOM.2012.6195527.

14 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 624–633, 2014. doi:10.1145/2591796.2591884.

15 Alon Efrat, Frank Hoffman, Klaus Kriegel, Christof Schultz, and Carola Wenk. Geometric algorithms for the analysis of 2d-electrophoresis gels. In *Proceedings of the fifth annual international conference on Computational biology*, pages 114–123, 2001. doi:10.1145/369133.369181.

16 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

17 Dorit S. Hochbaum and Wolfgang 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.

18 Dorit S Hochbaum and Wolfgang Maass. Fast approximation algorithms for a nonconvex covering problem. *Journal of Algorithms*, 8(3):305–323, 1987. doi:10.1016/0196-6774(87)90012-5.

19 Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. *J. Algorithms*, 37(1):146–188, 2000. doi:10.1006/JAGM.2000.1100.

20 Katarzyna Kowalska and Michał Pilipczuk. Parameterized and approximation algorithms for coverings points with segments in the plane. In *41st International Symposium on Theoretical Aspects of Computer Science*, 2024. doi:10.4230/LIPIcs.STACS.2024.47.

21 Jian Li and Yifei Jin. A PTAS for the weighted unit disk cover problem. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP*, volume 9134 of *Lecture Notes in Computer Science*, pages 898–909. Springer, 2015. doi:10.1007/978-3-662-47672-7_73.

22 Anil Maheshwari, Saeed Mehrabi, Sasanka Roy, and Michiel Smid. Covering points with concentric objects. In *Proceedings of the 32nd Canadian Conference on Computational Geometry*, pages 436–452, 2016.

23 Dániel Marx. Efficient approximation schemes for geometric problems? In *European Symposium on Algorithms*, pages 448–459. Springer, 2005. doi:10.1007/11561071_41.

24 Dániel Marx. Parameterized complexity of independence and domination on geometric graphs. In *Parameterized and Exact Computation: Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006. Proceedings 2*, pages 154–165. Springer, 2006. doi:10.1007/11847250_14.

25 Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008. doi:10.1093/COMJNL/BXM048.

26 Dániel Marx and Michał Pilipczuk. Optimal parameterized algorithms for planar facility location problems using voronoi diagrams. *ACM Transactions on Algorithms (TALG)*, 18(2):1–64, 2022. doi:10.1145/3483425.

27 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010. doi:10.1007/S00454-010-9285-9.

28 Neil Robertson, Daniel P Sanders, Paul Seymour, and Robin Thomas. Efficiently four-coloring planar graphs. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 571–575, 1996. doi:10.1145/237814.238005.

29 Anju Sangwan and Rishi Pal Singh. Survey on coverage problems in wireless sensor networks. *Wireless Personal Communications*, 80:1475–1500, 2015. doi:10.1007/S11277-014-2094-3.

30 Kasturi Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 641–648, 2010. doi:10.1145/1806689.1806777.

Tight Bounds on the Number of Closest Pairs in Vertical Slabs

Ahmad Biniaz
School of Computer Science,
University of Windsor, Canada

Prosenjit Bose
School of Computer Science, Carleton University,
Ottawa, Canada

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

Jean-Lou De Carufel
School of Electrical Engineering and
Computer Science, University of Ottawa, Canada

John Iacono
Department of Computer Science,
Université libre de Bruxelles, Brussels, Belgium

Anil Maheshwari
School of Computer Science, Carleton University,
Ottawa, Canada

Saeed Odak
School of Electrical Engineering and
Computer Science, University of Ottawa, Canada

Michiel Smid
School of Computer Science, Carleton University,
Ottawa, Canada

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

Abstract

Let S be a set of n points in $\mathbb{R}^d$, where $d \geq 2$ is a constant, and let $H_1, H_2, \ldots, H_{m+1}$ be a sequence of vertical hyperplanes that are sorted by their first coordinates, such that exactly n/m points of S are between any two successive hyperplanes. Let $|A(S,m)|$ be the number of different closest pairs in the $\binom{m+1}{2}$ vertical slabs that are bounded by H_i and H_j, over all $1 \leq i < j \leq m+1$. We prove tight bounds for the largest possible value of $|A(S,m)|$, over all point sets of size n, and for all values of $1 \leq m \leq n$.

As a result of these bounds, we obtain, for any constant $\varepsilon > 0$, a data structure of size $O(n)$, such that for any vertical query slab Q, the closest pair in the set $Q \cap S$ can be reported in $O(n^{1/2+\varepsilon})$ time. Prior to this work, no linear space data structure with sublinear query time was known.

2012 ACM Subject Classification Theory of computation → Communication complexity; Mathematics of computing → Combinatorial algorithms

Keywords and phrases closest pair, vertical slab, data structure

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.8

Related Version *Full Version*: https://arxiv.org/abs/2502.17600

Funding *Ahmad Biniaz*: Research supported by NSERC.
Prosenjit Bose: Research supported by NSERC.
Chaeyoon Chung: Research supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT).
Jean-Lou De Carufel: Research supported by NSERC.
John Iacono: This work was supported by the Fonds de la Recherche Scientifique-FNRS.
Anil Maheshwari: Research supported by NSERC.
Saeed Odak: Research supported by NSERC.
Michiel Smid: Research supported by NSERC.
Csaba D. Tóth: Research supported in part by the NSF award DMS-2154347.

© Ahmad Biniaz, Prosenjit Bose, Chaeyoon Chung, Jean-Lou De Carufel, John Iacono, Anil Maheshwari, Saeed Odak, Michiel Smid, and Csaba D. Tóth;
licensed under Creative Commons License CC-BY 4.0

Acknowledgements Research on this paper was conducted, in part, at the Eleventh Annual Workshop on Geometry and Graphs, held at the Bellairs Research Institute in Barbados, March 8–15, 2024. The authors are grateful to the organizers and to the participants of this workshop.

1 Introduction

Throughout this paper, we consider point sets in $\mathbb{R}^d$, where the dimension d is an integer constant. For any real number a, we define the *vertical hyperplane* H_a to be the set

$$H_a = \{(x_1, x_2, \ldots, x_d) \in \mathbb{R}^d : x_1 = a\}.$$

Note that this is a hyperplane with normal vector $(1, 0, 0, \ldots, 0)$. For any two real numbers a and b with $a < b$, we define the *vertical slab* $[\![H_a, H_b]\!]$ to be the set

$$[\![H_a, H_b]\!] = \{(x_1, x_2, \ldots, x_d) \in \mathbb{R}^d : a \leq x_1 \leq b\}.$$

Let S be a set of n points in $\mathbb{R}^d$, in which no two points have the same first coordinate and all $\binom{n}{2}$ pairwise Euclidean distances are distinct.

For any two real numbers a and b with $a < b$, we define $CP(S, H_a, H_b)$ to be the closest-pair among all points in the set $[\![H_a, H_b]\!] \cap S$, i.e., all points of S that are in the vertical slab $[\![H_a, H_b]\!]$. If $[\![H_a, H_b]\!] \cap S$ has size at most one, then $CP(S, H_a, H_b) = \infty$.

Clearly, there are $\Theta(n^2)$ combinatorially different[1] sets of the form $[\![H_a, H_b]\!] \cap S$. Sharathkumar and Gupta [5] have shown that, for $d = 2$, the size of the set

$$\{CP(S, H_a, H_b) : a < b\}$$

is only $O(n \log n)$. That is, even though there are $\Theta(n^2)$ combinatorially different vertical slabs with respect to S, the number of different closest pairs in these slabs is only $O(n \log n)$.

In this paper, we generalize this result to the case when the dimension d can be any constant and the slabs $[\![H_a, H_b]\!]$ come from a restricted set.

Let m be an integer with $1 \leq m \leq n$, and let $a_1 < a_2 < \cdots < a_{m+1}$ be real numbers such that for each $i = 1, 2, \ldots, m$, there are exactly[2] n/m points of S in the interior of the vertical slab $[\![H_{a_i}, H_{a_{i+1}}]\!]$. Observe that this implies that all points in S are in the interior of the vertical slab $[\![H_{a_1}, H_{a_{m+1}}]\!]$.

We define

$$A(S, m) = \{CP(S, H_{a_i}, H_{a_j}) : 1 \leq i < j \leq m + 1\}.$$

That is, $|A(S, m)|$ is the number of *different* closest pairs over all $\binom{m+1}{2}$ slabs bounded by vertical hyperplanes whose first coordinates belong to $\{a_1, a_2, \ldots, a_{m+1}\}$. Finally, we define

$$f_d(n, m) = \max\{|A(S, m)| : |S| = n\}.$$

Using this notation, Sharathkumar and Gupta [5] have shown that $f_2(n, n) = O(n \log n)$.

In dimension $d = 1$, it is easy to see that $f_1(n, m) = \Theta(m)$. Our main results are the following tight bounds on $f_d(n, m)$, for any constant $d \geq 2$ and any m with $1 \leq m \leq n$:

▶ **Theorem 1.** *Let $d \geq 2$ be a constant, and let m and n be integers such that $1 \leq m \leq n$.*

1. *If $m = O(\sqrt{n})$, then $f_d(n, m) = \Theta(m^2)$.*
2. *If $m = \omega(\sqrt{n})$, then $f_d(n, m) = \Theta(n \log(m^2/n))$.*
3. *In particular, if $m = n$, then $f_d(n, m) = \Theta(n \log n)$.*

[1] The slabs $[\![H_a, H_b]\!]$ and $[\![H_{a'}, H_{b'}]\!]$ are combinatorially different if their intersections with S are different.
[2] In order to avoid floors and ceilings, we assume for simplicity that n is a multiple of m.

1.1 Motivation

In the *range closest pair problem*, we have to store a given set S of n points in $\mathbb{R}^d$ in a data structure such that queries of the following type can be answered: Given a query range R in $\mathbb{R}^d$, report the closest pair among all points in the set $R \cap S$.

Many results are known for different classes of query ranges. We mention some of the currently best data structures. Xue *et al.* [9] present data structures for the case when $d = 2$ and the query ranges are quadrants, halfplanes, or axes-parallel rectangles. Again for the case when $d = 2$, data structures for query regions that are translates of a fixed shape are given by Xue *et al.* [8]. Some results in any constant dimension $d \geq 3$ are given by Chan *et al.* [3]. Xue [7] considers colored point sets, where the goal is to report the closest pair of points with different colors that are inside a query range. For constant dimension $d \geq 2$, [7] presents data structures for different types of query regions that report $(1 + \varepsilon)$-approximations for the closest pair with different colors. References to many other data structures can be found in [3, 8, 9].

Most of the currently known data structures use super-linear space. To the best of our knowledge, linear-sized data structures are known only for the following classes of regions, all in dimension $d = 2$: Quadrants and halfplanes [9], and translates of a fixed polygon (possibly with holes) [8]. In all these three cases, the query time is $O(\log n)$.

If each query range R is a vertical slab $[\![H_a, H_b]\!]$, we refer to the problem as the *vertical slab closest pair problem*. In dimension $d = 1$, it is easy to obtain a data structure of size $O(n)$ such that the closest pair in any "vertical slab" (i.e., interval on the real line) can be computed in $O(\log n)$ time. In dimension $d = 2$, Sharathkumar and Gupta [5] gave a data structure of size $O(n \log^2 n)$ that allows queries to be answered in $O(\log n)$ time. Xue *et al.* [9] improved the space bound to $O(n \log n)$, while keeping a query time of $O(\log n)$. Both these results use the fact that $f_2(n, n) = O(n \log n)$. In fact, both data structures explicitly store the set $\{CP(S, H_a, H_b) : a < b\}$, whose size is equal to $f_2(n, n)$ in the worst case.

The starting point of our work was to design a data structure of size $O(n)$ for vertical slab closest pair queries. This led us to the problem of determining the asymptotic value of the function $f_d(n, m)$. Using our bounds in Theorem 1, we will obtain the following result.

▶ **Theorem 2.** *Let $d \geq 2$ be an integer constant and let $\varepsilon > 0$ be a real constant. For every set S of n points in $\mathbb{R}^d$, there exists a data structure of size $O(n)$ that allows vertical slab closest pair queries to be answered in $O(n^{1/2+\varepsilon})$ time.*

Note that, prior to our work, no $O(n)$-space data structure with a query time of $o(n)$ was known for $d \geq 2$.

Organization. In Section 2, we will present the upper bounds in Theorem 1 on $f_d(n, m)$. The corresponding lower bounds will be given in Section 3. The data structure in Theorem 2 will be presented in Section 4. We conclude in Section 5 with some open problems.

Notation and Terminology. Throughout the rest of this paper, the notions of left and right in $\mathbb{R}^d$ will always refer to the ordering in the first coordinate. That is, if $p = (p_1, p_2, \ldots, p_d)$ and $q = (q_1, q_2, \ldots, q_d)$ are two points in $\mathbb{R}^d$ with $p_1 < q_1$, then we say that p is to the *left* of q, and q is to the *right* of p. For a vertical hyperplane H_a, we say that p is to the *left* of H_a if $p_1 < a$. If $p_1 > a$, then p is to the *right* of H_a.

The Euclidean distance between any two points p and q in $\mathbb{R}^d$ will be denoted by $||p - q||$. The length, or norm, of any vector v will be denoted by $||v||$.

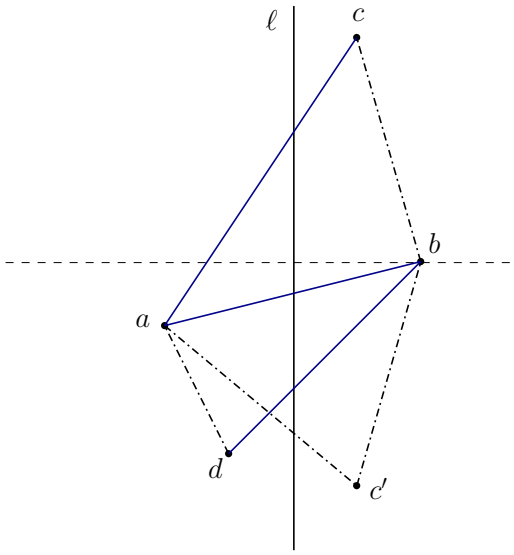


Figure 1 The pairs in $A(S, n)$ with positive slope that cross ℓ do not contain a cycle.

2 Upper bounds on $f_d(n, m)$

Let $d \geq 2$ be a constant, let m and n be integers with $1 \leq m \leq n$, and let S be a set of n points in $\mathbb{R}^d$. Let $a_1 < a_2 < \cdots < a_{m+1}$ be real numbers such that for each $i = 1, 2, \ldots, m$, there are exactly n/m points in S between the vertical hyperplanes H_{a_i} and $H_{a_{i+1}}$.

For any m, it is clear that $f_d(n, m) = O(m^2)$, because there are $\binom{m+1}{2}$ vertical slabs of the form $[\![H_{a_i}, H_{a_j}]\!]$. Thus, the upper bound in Theorem 1 holds when $m = O(\sqrt{n})$. In the rest of this section, we assume that $m = \omega(\sqrt{n})$.

The following lemma was proved by Sharathkumar and Gupta [5] for the case when $d = 2$. This lemma will be the key tool to prove our upper bound on $f_d(n, m)$.

▶ **Lemma 3.** *Let S be a set of n points in $\mathbb{R}^d$ and let $\mathcal{CP}$ be the set of segments corresponding to the elements of $A(S, n)$. That is, for each pair in $A(S, n)$, the set $\mathcal{CP}$ contains the line segment connecting the two points in this pair. For any vertical hyperplane H, the number of elements of $\mathcal{CP}$ that cross H is $O(n)$.*

We first present a proof of this lemma for the case when $d = 2$. We believe that our proof is simpler than the one in [5]. Afterwards, we present a proof for any dimension $d \geq 2$.

Proof of Lemma 3 when $d = 2$. We write ℓ for the vertical line. We define a graph, G^+, with vertex set S. Each segment of $\mathcal{CP}$ with a positive slope represents an edge in the graph G^+. Let F be the subgraph of G^+ induced by the segments of $\mathcal{CP}$ that cross ℓ. We will show that F does not contain a cycle.

Suppose, to the contrary, that there is a cycle C in F. Let a and b be the endpoints of the shortest edge in C such that a is to the left of ℓ and b is to the right of ℓ. Let ac and bd be the other edges of the cycle that are incident to a and b, respectively. Since both ab and ac represent pairs in $\mathcal{CP}$ and both have a positive slope, we have $a_x < c_x < b_x$ and $a_y < b_y < c_y$. Similarly, we have $a_x < d_x < b_x$ and $d_y < a_y < b_y$; see Figure 1.

Let c' be the reflection of the point c with respect to the horizontal line through b. Note that $||b - c'|| = ||b - c|| > ||b - d||$, because bd represents a pair in $A(S, n)$ and the vertical slab $[\![b_x, d_x]\!]$ contains the point c. Since $||b - c'|| > ||b - d||$, we have $c'_y < d_y$. We also have $d_y < a_y$ and $a_x < d_x < c_x = c'_x$. It follows that $||a - d|| < ||a - c'||$.

Consider the bisector of the segment cc' (which is the horizontal line through b). Observe that the point a is located on the same side as c' with respect to this bisector. Therefore, $||a - c'|| < ||a - c||$. Combined with $||a - d|| < ||a - c'||$, this implies that $||a - d|| < ||a - c||$. This contradicts the facts that ac represents a pair in $A(S, n)$ and the point d is in the slab $[\![a_x, c_x]\!]$.

A similar argument shows that the segments in $\mathcal{CP}$ that cross ℓ and have negative slopes do not contain a cycle. Therefore, the total number of segments in $\mathcal{CP}$ that cross the line ℓ is $O(n)$. ◀

To prove Lemma 3 for dimensions $d \geq 2$, we will use the *Well-Separated Pair Decomposition (WSPD)*, as introduced by Callahan and Kosaraju [2]. Let S be a set of n points in $\mathbb{R}^d$ and let $s > 1$ be a real number, called the *separation ratio*. A WSPD for S is a set of pairs $\{A_i, B_i\}$, for $i = 1, 2, \ldots, k$, for some positive integer k, such that

1. for each i, $A_i \subseteq S$ and $B_i \subseteq S$,
2. for each i, there exist two balls D and D' of the same radius, say ρ, such that $A_i \subseteq D$, $B_i \subseteq D'$, and the distance between D and D' is at least $s \cdot \rho$, i.e., the distance between their centers is at least $(s + 2) \cdot \rho$,
3. for any two distinct points p and q in S, there is a unique index i such that $p \in A_i$ and $q \in B_i$ or vice-versa.

Consider a pair $\{A_i, B_i\}$ in a WSPD. If p and p' are two points in A_i and q is a point in B_i, then it is easy to see that

$$||p - p'|| \leq (2/s) \cdot ||p - q||. \tag{1}$$

▶ **Lemma 4** (Callahan and Kosaraju [2]). *Let S be a set of n points in $\mathbb{R}^d$, and let $s > 1$ be a real number. A well-separated pair decomposition for S, with separation ratio s, consisting of $O(s^d n)$ pairs, can be computed in $O(n \log n + s^d n)$ time.*

Proof of Lemma 3. Let $s > 2$ be a constant and consider a WSPD $\{A_i, B_i\}$, $i = 1, 2, \ldots, k$, for the point set S with separation ratio s, where $k = O(n)$; see Lemma 4. We define the following geometric graph G on the point set S. For each i with $1 \leq i \leq k$, let

- a_i be the rightmost point in A_i that is to the left of H,
- b_i be the leftmost point in B_i that is to the right of H,
- a'_i be the leftmost point in A_i that is to the right of H, and
- b'_i be the rightmost point in B_i that is to the left of H.

We add the edges $a_i b_i$ and $a'_i b'_i$ to the graph G. Note that some of these points may not exist, in which case we ignore the corresponding edge. It is clear that G has $O(n)$ edges. The lemma will follow from the fact that every segment in $\mathcal{CP}$ that crosses H is an edge in G.

Let pq be a pair in $\mathcal{CP}$ that crosses H, and let Q be a vertical slab such that pq is the closest pair in $Q \cap S$. We may assume, without loss of generality, that p is to the left of H and q is to the right of H. Let i be the index such that (i) $p \in A_i$ and $q \in B_i$ or (ii) $p \in B_i$ and $q \in A_i$. We may assume, without loss of generality, that (i) holds.

We claim that $p = a_i$. To prove this, suppose that $p \neq a_i$. Then, since p is to the left of a_i, a_i is in the slab Q. Since $s > 2$, Equation (1) yields $||p - a_i|| < ||p - q||$, which is a contradiction. By a symmetric argument, we have $q = b_i$. Thus, pq is an edge in G. ◀

Lemma 3 gives us a divide-and-conquer approach to prove an upper bound on $f_d(n, m)$:

▶ **Theorem 5.** *Let $d \geq 2$ be a constant, and let m and n be integers with $m = \omega(\sqrt{n})$ and $m \leq n$. Then $f_d(n, m) = O(n \log (m^2/n))$.*

Proof. Let S be a set of n points in $\mathbb{R}^d$ for which $f_d(n,m) = |A(S,m)|$. Let $a_1 < a_2 < \cdots < a_{m+1}$ be real numbers such that for each $i = 1, 2, \ldots, m$, there are exactly n/m points in S that are strictly inside the vertical slab $[\![H_{a_i}, H_{a_{i+1}}]\!]$.

Let $H = H_{a_{1+m/2}}$. Observe that $n/2$ points of S are to the left of H and $n/2$ points of S are to the right of H. Denote these two subsets by S^- and S^+, respectively. Each pair in $A(S,m)$ is either a pair in $A(S^-, m/2)$ or a pair in $A(S^+, m/2)$ or it crosses H. Using Lemma 3, it follows that

$$f_d(n,m) = |A(S,m)| = |A(S^-, m/2)| + |A(S^+, m/2)| + O(n) \leq 2 \cdot f_d(n/2, m/2) + O(n).$$

If we apply this recurrence k times, we get

$$f_d(n,m) \leq 2^k \cdot f_d(n/2^k, m/2^k) + O(kn).$$

For $k = \log(m^2/n)$, we have $n/2^k = n^2/m^2$ and $m/2^k = n/m$. Thus,

$$f_d(n,m) \leq \frac{m^2}{n} \cdot f_d\left(n^2/m^2, n/m\right) + O(n \log(m^2/n)).$$

Since $f_d\left(n^2/m^2, n/m\right) = O(n^2/m^2)$, we conclude that

$$f_d(n,m) = O(n + n\log(m^2/n)) = O(n \log(m^2/n)).$$ ◀

3 Lower bounds on $f_d(n,m)$

In this section, we prove the lower bounds on $f_d(n,m)$ as stated in Theorem 1. We will prove these lower bounds for the case when $d = 2$. It is clear that this will imply the same lower bound for any constant dimension $d \geq 2$.

▶ **Theorem 6.** *Let n and m be positive integers with $n \geq m(m+1)$. Then $f_2(n,m) = \binom{m+1}{2}$.*

Proof. It is clear that $f_2(n,m) \leq \binom{m+1}{2}$. To prove the lower bound, we will construct a set S of n points in $\mathbb{R}^2$ such that the $\binom{m+1}{2}$ vertical slabs have distinct closest pairs.

For $i = 1, 2, \ldots, m+1$, we take $a_i = i$ and consider the corresponding hyperplane H_i. Let $\mathcal{Q} = \{[\![H_i, H_j]\!] : 1 \leq i < j \leq m+1\}$ be the set of all possible vertical slabs. We define the *size* of a slab $[\![H_i, H_j]\!]$ to be the difference $j - i$ of their indices.

We start by constructing a set P of $m(m+1)$ points such that the slabs in $\mathcal{Q}$ contain distinct closest pairs in P, and for each $i = 1, 2, \ldots, m$, the slab $[\![H_i, H_{i+1}]\!]$ contains exactly $m+1$ points of P.

Note that the slab $[\![H_1, H_{m+1}]\!]$ has the largest size. Let p be an arbitrary point in $[\![H_1, H_2]\!]$ and let q be an arbitrary point in $[\![H_m, H_{m+1}]\!]$. We initialize $P = \{p, q\}$, $D = ||p - q||$, and delete the slab $[\![H_1, H_{m+1}]\!]$ from $\mathcal{Q}$.

As long as $\mathcal{Q}$ is non-empty, we do the following:

- Take a slab $[\![H_i, H_j]\!]$ of largest size in $\mathcal{Q}$.
- Let p be an arbitrary point in $[\![H_i, H_{i+1}]\!]$ such that p is above the bounding box of P, and the distance between p and any point in P is more than $D + 2$.
- Let q be an arbitrary point in $[\![H_{j-1}, H_j]\!]$ such that q is above the bounding box of P, the distance between q and any point in P is more than $D + 2$, and $||p - q|| = D + 1$.
- Add p and q to P.
- Set $D = ||p - q||$.
- Delete the slab $[\![H_i, H_j]\!]$ from $\mathcal{Q}$.

It is not difficult to see that the final point set P has the properties stated above.

To obtain the final point set S, of size n, we define a set P' of $n - m(m+1)$ points, such that each point in P' has distance more than D to all points of P, the closest pair distance in P' is more than D, and for each $i = 1, 2, \ldots, m$, the slab $[\![H_i, H_{i+1}]\!]$ contains $n/m - (m-1)$ points of P'. The point set $S = P \cup P'$ has the property that $|A(S, m)| = \binom{m+1}{2}$. ◀

▶ **Corollary 7.** *Let n and m be sufficiently large positive integers with $n < m(m+1)$ and $m \leq 3\sqrt{n}$. Then $f_2(n, m) = \Omega(m^2)$.*

Proof. For $i = 1, 2, \ldots, m+1$, we take $a_i = i$ and consider the corresponding hyperplane H_i.

Let $m' = \sqrt{n}/4$ and $n' = m'(m'+1)$. We apply Theorem 6, where we replace n by n' and m by m'. This gives us a set S' of n' points with $|A(S', m')| = f_2(n', m')$. The points of S' are between the hyperplanes H_1 and $H_{m'+1}$; for each $i = 1, 2, \ldots, m'$, the vertical slab $[\![H_i, H_{i+1}]\!]$ contains n'/m' points of S'. Note that

$$|A(S', m')| = \binom{m'+1}{2} = \Omega\left((m')^2\right).$$

Let D be the diameter of S'. Let S be the union of S' and a set of $n - n'$ additional points that have pairwise distances more than D, whose distances to the points in S' are more than D, and such that for each $i = 1, 2, \ldots, m$, the vertical slab $[\![H_i, H_{i+1}]\!]$ contains n/m points of S. It is clear that

$$f_2(n, m) \geq |A(S', m')| = \Omega((m')^2).$$

Note that this construction is possible, because (i) $n' < n$, (ii) $m' < m$, and (iii) $n'/m' < n/m$; these inequalities follow by straightforward algebraic manipulations, using the assumptions on n and m in the statement of the corollary. Finally, these assumptions imply that $m' \geq m/12$. We conclude that $f_2(n, m) = \Omega(m^2)$. ◀

Before we prove the lower bound for the remaining case, i.e., $m > 3\sqrt{n}$, we consider the case when $m = n$, which will serve as a warm up.

▶ **Theorem 8.** *We have $f_2(n, n) = \Omega(n \log n)$.*

Proof. We assume for simplicity that n is a sufficiently large power of two. We will construct a point set S of size n for which $|A(S, n)| = \Omega(n \log n)$.

Let $k = \log n$. For $i = 0, 1, \ldots, k-1$, let $x_i = 2^i$ and let $v_i = (x_i, y_i)$ be a vector, where the value of y_i is inductively defined as follows: We set $y_{k-1} = 0$. Assuming that $y_{k-1}, y_{k-2}, \ldots, y_{i+1}$ have been defined, we set y_i to an integer such that

$$||v_i|| > 2 \sum_{j=i+1}^{k-1} ||v_j||. \tag{2}$$

We define

$$S = \left\{ \sum_{i=0}^{k-1} \beta_i v_i : (\beta_0, \beta_1, \cdots, \beta_{k-1}) \in \{0, 1\}^k \right\}.$$

Note that each binary sequence of length k represents a unique point in S. Using this representation, each point of S corresponds to a vertex of a k-dimensional hypercube Q_k. We will prove below that each edge of Q_k corresponds to a closest pair in a unique vertical slab. Since Q_k has $k \cdot 2^{k-1} = \Omega(n \log n)$ edges, this will complete the proof.

Consider an arbitrary edge of Q_k. The two vertices of this edge are binary sequences of length k that have Hamming distance one, i.e., they differ in exactly one bit. Let t be the position at which they differ. Observe that $0 \leq t \leq k-1$. Let r and s be the points of S that correspond to the two vertices of this edge. Then v_t is either $r-s$ or $s-r$. We will prove that r and s form the closest pair in the vertical slab $[\![H_{r_1}, H_{s_1}]\!]$, where r_1 and s_1 are the first coordinates of r and s, respectively (assuming that $r_1 < s_1$). Note that r_1 and s_1 are integers.

Let p and q be two points in $[\![H_{r_1}, H_{s_1}]\!] \cap S$ such that $\{p,q\} \neq \{r,s\}$. We have to show that $||r-s|| < ||p-q||$. Since p and q are points in S, we can write them as

$$p = \sum_{i=0}^{k-1} \beta_{p,i} v_i \text{ and } q = \sum_{i=0}^{k-1} \beta_{q,i} v_i.$$

Let ℓ be the smallest index for which $\beta_{p,\ell} \neq \beta_{q,\ell}$. Then

$$||p-q|| = \left|\left| \sum_{i=\ell}^{k-1} (\beta_{p,i} - \beta_{q,i}) v_i \right|\right|.$$

By the triangle inequality, we have $||a+b|| \leq ||a|| + ||b||$ and $||a+b|| \geq ||a|| - ||b||$ for any two vectors a and b. These two inequalities, together with (2), imply that

$$||p-q|| \geq ||v_\ell|| - \sum_{i=\ell+1}^{k-1} ||v_i|| > \frac{1}{2} \cdot ||v_\ell||.$$

Thus, it suffices to show that $||v_\ell|| \geq 2 \cdot ||r-s|| = 2 \cdot ||v_t||$. If we can show that $\ell < t$, then, using (2),

$$||v_\ell|| > 2 \sum_{i=\ell+1}^{k-1} ||v_i|| \geq 2 \cdot ||v_t||$$

and the proof is complete.

The horizontal distance between p and q (i.e., $|p_1 - q_1|$) is smaller than the horizontal distance between r and s. Recall that v_t is either $r-s$ or $s-r$. Thus, the horizontal distance between r and s is equal to the first coordinate of the vector v_t, which is x_t.

Let ℓ' be the largest index for which $\beta_{p,\ell'} \neq \beta_{q,\ell'}$. Note that $\ell \leq \ell'$. If $\ell = \ell'$, then

$$p - q = (\beta_{p,\ell} - \beta_{q,\ell})\, v_\ell$$

and

$$|p_1 - q_1| = |\beta_{p,\ell} - \beta_{q,\ell}| x_\ell = x_\ell.$$

Now assume that $\ell < \ell'$. We may assume, without loss of generality, that $\beta_{p,\ell'} = 1$ and $\beta_{q,\ell'} = 0$. Then (recall that $x_i = 2^i$),

$$|p_1 - q_1| \quad = \quad \left| x_{\ell'} + \sum_{i=\ell}^{\ell'-1} (\beta_{p,i} - \beta_{q,i}) x_i \right| = \left| 2^{\ell'} + \sum_{i=\ell}^{\ell'-1} (\beta_{p,i} - \beta_{q,i}) 2^i \right| \geq 2^{\ell'} - \sum_{i=\ell}^{\ell'-1} 2^i = 2^\ell = x_\ell.$$

We conclude that $x_\ell \leq |p_1 - q_1| < x_t$, which is equivalent to $\ell < t$. ◀

▶ **Theorem 9.** *Let n and m be positive integers with $3\sqrt{n} < m \leq n$. Then $f_2(n,m) = \Omega(n \log(m^2/n))$.*

Proof. Our approach will be to use multiple scaled and shifted copies of the construction in Theorem 8 to define a set S of n points in $\mathbb{R}^2$ for which $|A(S,m)| = \Omega(n \log(m^2/n))$.

For $i = 1, 2, \ldots, m+1$, we take $a_i = i$ and consider the corresponding vertical hyperplane H_i. For each $i = 1, 2, \ldots, m$, the point set S will contain exactly n/m points in the vertical slab $[\![H_i, H_{i+1}]\!]$.

Throughout the proof, we will use the following notation. Let $v_0, v_1, \ldots, v_{k-1}$ be a sequence of pairwise distinct vectors in the plane. The *hypercube-set* defined by these vectors is the point set

$$Q(v_0, v_1, \ldots, v_{k-1}) = \left\{ \sum_{i=0}^{k-1} \beta_i v_i : (\beta_0, \beta_1, \cdots, \beta_{k-1}) \in \{0,1\}^k \right\}.$$

For each $g = 1, 2, \ldots, n/m$, we define a hypercube-set Q_g:

- Let $k_g = \lfloor \log(m/(2g-1)) \rfloor$.
- For each $i = 0, 1, \ldots, k_g - 1$, let $x_{g,i} = (2g-1) \cdot 2^i$ and let $v_{g,i} = (x_{g,i}, y_{g,i})$ be a vector, whose second coordinate $y_{g,i}$ will be defined later.
- Let $Q_g = Q(v_{g,0}, v_{g,1}, \ldots, v_{g,k_g-1})$.

Since, for integers g, g', i, and i', $(2g-1) \cdot 2^i = (2g'-1) \cdot 2^{i'}$ if and only if $g = g'$ and $i = i'$, then all values $x_{g,i}$ are pairwise distinct.

To define the values $y_{g,i}$, we sort all vectors $v_{g,i}$ by their first coordinates. We go through the sorted sequence in decreasing order:

- For the vector with the largest first coordinate, we set its y-value to zero.
- For each subsequent vector $v_{g,i}$, we set $y_{g,i}$ to be an integer such that

$$||v_{g,i}|| > 2 \sum_{g',i'} ||v_{g',i'}||, \tag{3}$$

where the summation is over all pairs g', i' for which $y_{g',i'}$ has already been defined (i.e., $x_{g,i} < x_{g',i'}$).

We choose pairwise distinct real numbers $0 < \varepsilon_g < 1$, for $g = 1, 2, \ldots, n/m$, and set

$$\Delta = 1 + \max\{\mathrm{diam}(Q_g) : 1 \le g \le n/m\},$$

where $\mathrm{diam}(Q_g)$ denotes the diameter of the point set Q_g.

For each $g = 1, 2, \ldots, n/m$ and $i = 1, 2, \ldots, 2g-1$, let

$$S_{g,i} = Q_g + (i + \varepsilon_g, 2((g-1)^2 + i - 1)\Delta),$$

that is, $S_{g,i}$ is the translate of Q_g by the vector $(i + \varepsilon_g, 2((g-1)^2 + i - 1)\Delta)$. We define S' to be the union of all these sets $S_{g,i}$, i.e.,

$$S' = \bigcup_{g=1}^{n/m} \bigcup_{i=1}^{2g-1} S_{g,i}.$$

Note that the sets $S_{g,i}$ are pairwise disjoint: Indeed if $g \neq g'$ or $i \neq i'$, then the y-projections of $S_{g,i}$ and $S_{g',i'}$ (i.e., the sets of second coordinates) are disjoint by construction. Consequently, the size of the union of these point sets satisfies

$$|S'| = \sum_{g=1}^{n/m} \sum_{i=1}^{2g-1} |S_{g,i}| = \sum_{g=1}^{n/m} \sum_{i=1}^{2g-1} 2^{k_g} \le \sum_{g=1}^{n/m} (2g-1) \cdot \frac{m}{2g-1} = n.$$

For each $1 \leq g \leq n/m$, by construction of Q_g and the fact that the sets $S_{g,i}$ are disjoint translations of Q_g, each slab $[\![H_j, H_{j+1}]\!]$ contains at most one point of $\bigcup_{i=1}^{2g-1} S_{g,i}$. Therefore, each slab $[\![H_j, H_{j+1}]\!]$ contains at most n/m points of S'.

To obtain the final point set S of size n, we add $n - |S'|$ points to S' such that each slab $[\![H_j, H_{j+1}]\!]$ contains exactly n/m points of S, and the added points are sufficiently far from each other and from all points of S'.

In the rest of this proof, we will prove the following claim: For each $g = 1, 2, \ldots, n/m$, consider two binary strings of length k_g that differ in exactly one position (recall that the number of such pairs of strings is equal to $k_g \cdot 2^{k_g-1}$). These strings correspond to two points of the hypercube-set Q_g. Thus, for any $i = 1, 2, \ldots, 2g-1$, they correspond to two points, say r and s, in the set $S_{g,i}$. We claim that r and s form the closest pair in the set $[\![H_{\lfloor r_1 \rfloor}, H_{\lceil s_1 \rceil}]\!] \cap S$, where r_1 and s_1 are the first coordinates of r and s, respectively (assuming that $r_1 < s_1$). Note that we take the floor and the ceiling, because r_1 and s_1 are not integers. This claim will imply that

$$f_2(n, m) \geq |A(S, m)| \geq \sum_{g=1}^{n/m} \sum_{i=1}^{2g-1} k_g \cdot 2^{k_g-1}.$$

Since $k_g > \log\left(\frac{m}{2g-1}\right) - 1$, we get

$$\begin{aligned} f_2(n, m) &\geq \sum_{g=1}^{n/m} (2g-1)\left(\log\left(\frac{m}{2g-1}\right) - 1\right) \cdot \frac{m}{4(2g-1)} \\ &= \sum_{g=1}^{n/m} \frac{m}{4} \log\left(\frac{m}{2g-1}\right) - \sum_{g=1}^{n/m} \frac{m}{4}. \end{aligned}$$

Since each term in the first summation is larger than the last term, which is larger than $(m/4)\log(m^2/(2n))$, we get

$$f_2(n, m) \geq \frac{n}{m} \cdot \frac{m}{4} \log(m^2/(2n)) - \frac{n}{4} = \frac{n}{4}\left(\log(m^2/(2n)) - 1\right).$$

Since $m > 3\sqrt{n}$,

$$\log(m^2/(2n)) - 1 = \Omega(\log(m^2/n)).$$

We conclude that

$$f_2(n, m) = \Omega(n \log(m^2/n)).$$

It remains to prove the above claim. Let g and i be integers with $1 \leq g \leq n/m$ and $1 \leq i \leq 2g-1$. Consider two binary strings of length k_g that differ in exactly one position; denote this position by t. Let r and s be the two corresponding points of $S_{g,i}$. Note that the vector $v_{g,t}$ is equal to either $r - s$ or $s - r$.

We may assume, without loss of generality, that $r_1 < s_1$. To prove that r and s form the closest pair in the set $[\![H_{\lfloor r_1 \rfloor}, H_{\lceil s_1 \rceil}]\!] \cap S$, we consider an arbitrary pair p and q of points in $[\![H_{\lfloor r_1 \rfloor}, H_{\lceil s_1 \rceil}]\!] \cap S$ such that $\{p, q\} \neq \{r, s\}$. We will show that $||r - s|| < ||p - q||$.

Let g', g'', i', and i'' be such that $p \in S_{g',i'}$ and $q \in S_{g'',i''}$. If $g' \neq g''$ or $i' \neq i''$, then

$$\|p - q\| \geq |p_2 - q_2| \geq \Delta > \operatorname{diam}(Q_g) \geq \|r - s\|.$$

In the rest of the proof, we assume that $g' = g''$ and $i' = i''$. Since both p and q are in $S_{g',i'}$, we can write them as

$$p = \sum_{j=0}^{k_{g'}-1} \beta_{p,j} v_{g',j} + (i' + \varepsilon_{g'}, 2((g'-1)^2 + i' - 1)\Delta) \text{ and}$$

$$q = \sum_{j=0}^{k_{g'}-1} \beta_{q,j} v_{g',j} + (i' + \varepsilon_{g'}, 2((g'-1)^2 + i' - 1)\Delta).$$

Let ℓ be the smallest index such that $0 \le \ell \le k_{g'} - 1$ and $\beta_{p,\ell} \neq \beta_{q,\ell}$. Using the triangle inequality and (3), we have

$$||p - q|| = \left|\left| \sum_{j=\ell}^{k_{g'}-1} (\beta_{p,j} - \beta_{q,j}) v_{g',j} \right|\right| \ge ||v_{g',\ell}|| - \sum_{j=\ell+1}^{k_{g'}-1} ||v_{g',j}|| > \frac{1}{2} \cdot ||v_{g',\ell}||.$$

Thus, it suffices to show that

$$||v_{g',\ell}|| \ge 2 \cdot ||r - s|| = 2 \cdot ||v_{g,t}||.$$

Since both p and q are in $S_{g',i'}$, we have $|p_1 - q_1| \le |r_1 - s_1|$ (where r and s are each at distance ε_g from the left boundary of their corresponding slabs, and p and q are at distance $\varepsilon_{g'}$ from the left boundary of their corresponding slabs). Let ℓ' be the largest index such that $0 \le \ell' \le k_{g'} - 1$ and $\beta_{p,\ell'} \neq \beta_{q,\ell'}$. We have

$$\begin{aligned} x_{g,t} &= |r_1 - s_1| \ge |p_1 - q_1| = \left| \sum_{j=\ell}^{\ell'} (\beta_{p,j} - \beta_{q,j}) x_{g',j} \right| \ge x_{g',\ell'} - \sum_{j=\ell}^{\ell'-1} x_{g',j} \\ &= (2g' - 1) \left(2^{\ell'} - \sum_{j=\ell}^{\ell'-1} 2^j \right) = (2g' - 1) \cdot 2^{\ell} = x_{g',\ell}. \end{aligned}$$

Since $\{p, q\} \neq \{r, s\}$, then $x_{g,t}$ cannot be equal to $x_{g',\ell}$. Therefore, $x_{g,t} > x_{g',\ell}$. Using (3), it then follows that $||v_{g',\ell}|| \ge 2 \cdot ||v_{g,t}||$. ◀

4 The Data Structure

In this section, we will present a data structure for vertical closest pair queries. Our data structure will use the results in the following three lemmas.

▶ **Lemma 10.** *Let S be a set of n points in $\mathbb{R}^d$ and let L be a set of k line segments such that the endpoints of each segment belongs to S. There exists a data structure of size $O(n + k)$, such that for any two real numbers a and b with $a < b$, the shortest segment in L that is completely inside the vertical slab $[\![H_a, H_b]\!]$ can be reported in $O(\log n)$ time.*

Proof. Xue *et al.* [9, Section 3] proved the claim in the case where $d = 2$. A careful analysis of their construction shows that the claim in fact holds for any constant dimension $d \ge 2$. ◀

The next lemma is due to Mehlhorn [4, page 44]; see Smid [6] for a complete analysis of this data structure.

▶ **Lemma 11.** *Let S be a set of n points in $\mathbb{R}^d$ and let $\varepsilon > 0$ be a real constant. There exists a data structure of size $O(n)$, such that for any axis-parallel rectangular box B, all points in $B \cap S$ can be reported in $O(n^{\varepsilon} + |B \cap S|)$ time.*

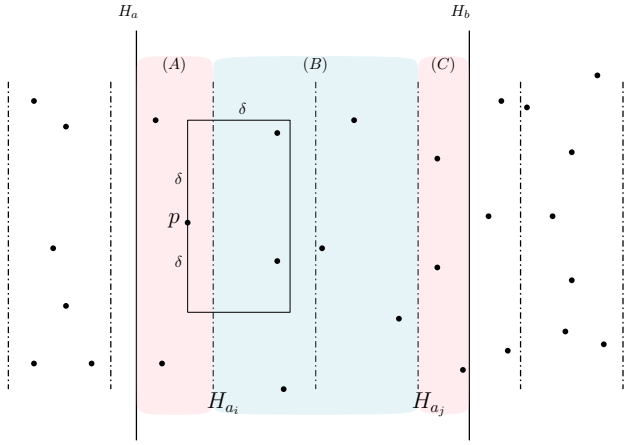


Figure 2 (A), (B), and (C) are the three regions created by a query $[\![H_a, H_b]\!]$. The rectangle R_p is the range query for the point p with respect to the query $[\![H_a, H_b]\!]$.

The last tool that we need is a standard sparsity property.

▶ **Lemma 12.** *Let $r > 0$ be a real number, and let X be a set of points in $\mathbb{R}^d$ that are contained in an $r \times 2r \times 2r \times \cdots \times 2r$ rectangular box B. If the distance of the closest pair of points in X is at least r, then $|X| \leq 2^{d-1} \cdot c^d$, where $c = 1 + \lceil \sqrt{d} \rceil$.*

In the rest of this section, we will prove the following result.

▶ **Theorem 13.** *Let S be a set of n points in $\mathbb{R}^d$, let m be an integer with $1 \leq m \leq n$, and let $\varepsilon > 0$ be a real constant. There exists a data structure of size $O(n + f_d(n, m))$ such that for any two real numbers a and b with $a < b$, the closest pair in the vertical slab $[\![H_a, H_b]\!]$ can be reported in $O(n^{1+\varepsilon}/m)$ time.*

Proof. Let $a_1 < a_2 < \cdots < a_{m+1}$ be real numbers such that for each $i = 1, 2, \ldots, m$, the vertical slab $[\![H_{a_i}, H_{a_{i+1}}]\!]$ contains n/m points of S. Let $k = |A(S, m)|$. Note that $k \leq f_d(n, m)$. Our data structure consists of the following components:

- An array storing the numbers $a_1, a_2, \ldots, a_{m+1}$. For each $i = 1, 2, \ldots, m$, the i-th entry stores, besides the number a_i, a list of all points in $[\![H_{a_i}, H_{a_{i+1}}]\!] \cap S$.
- The data structure of Lemma 10, where L is the set of line segments corresponding to the pairs in $\{CP(S, H_{a_i}, H_{a_j}) : 1 \leq i < j \leq m + 1\}$.
- The data structure of Lemma 11.

The size of the entire data structure is $O(m + n + k)$, which is $O(n + f_d(n, m))$.

We next describe the query algorithm. Let a and b be real numbers with $a < b$. Using binary search, we compute, in $O(\log m) = O(n^{1+\varepsilon}/m)$ time, the indices i and j such that H_a is in the slab $[\![H_{a_{i-1}}, H_{a_i}]\!]$ and H_b is in the slab $[\![H_{a_j}, H_{a_{j+1}}]\!]$.

If $i = j$, then the slab $[\![H_a, H_b]\!]$ contains $O(n/m)$ points of S. In this case, we use the algorithm of Bentley and Shamos [1] to compute the closest pair in $[\![H_a, H_b]\!]$ in $O((n/m) \log(n/m)) = O(n^{1+\varepsilon}/m)$ time.

Assume that $i < j$. The two hyperplanes H_{a_i} and H_{a_j} split the query slab $[\![H_a, H_b]\!]$ into three parts (A), (B), and (C), where (A) is the slab $[\![H_a, H_{a_i}]\!]$, (B) is the slab $[\![H_{a_i}, H_{a_j}]\!]$, and (C) is the slab $[\![H_{a_j}, H_b]\!]$; refer to Figure 2.

Let S_{AC} be the set of points in S that are in the union of (A) and (C), and let S_B be the set of points in S that are in (B). There are three possibilities for the closest pair in $[\![H_a, H_b]\!]$: Both endpoints are in S_{AC}, both endpoints are in S_B, or one endpoint is in S_{AC} and the other endpoint is in S_B.

Using the algorithm of Bentley and Shamos [1], we compute the closest pair distance δ_1 in S_{AC}, in $O((n/m)\log(n/m)) = O(n^{1+\varepsilon}/m)$ time. Using the data structure of Lemma 10, we compute the closest pair distance δ_2 in S_B in $O(\log n) = O(n^{1+\varepsilon}/m)$ time.

Let $\delta = \min(\delta_1, \delta_2)$. In the final part of the query algorithm, we use the data structure of Lemma 11:

- For each point p in the region (A), we compute the set of all points in S that are in the part, say P_p, of the axes-parallel box

$$[p_1, p_1 + \delta] \times [p_2 - \delta, p_2 + \delta] \times \cdots \times [p_d - \delta, p_d + \delta]$$

  that is to the left of H_{a_j}. Then we compute δ_p, which is the minimum distance between p and any point inside $S \cap P_p$.
- For each point p in the region (C), we compute the set of all points in S that are in the part, say P'_p, of the axes-parallel box

$$[p_1 - \delta, p_1] \times [p_2 - \delta, p_2 + \delta] \times \cdots \times [p_d - \delta, p_d + \delta]$$

  that is to the right of H_{a_i}. Then we compute δ_p, which is the minimum distance between p and any point inside $S \cap P'_p$.
- At the end , we return the minimum of δ and $\min\{\delta_p : p \in S_{AC}\}$.

By Lemma 12, the boxes P_p and P'_p each contain $O(1)$ points of S. In total, there are $O(n/m)$ queries to the data structure of Lemma 11, and each one takes $O(n^{\varepsilon})$ time. Thus, this final part of the query algorithm takes $O((n/m) \cdot n^{\varepsilon}) = O(n^{1+\varepsilon}/m)$ time. ◀

The proof of Theorem 2 follows by taking $m = \sqrt{n}$ in Theorem 13 and using Theorem 1.

5 Future Work

The point sets that we constructed for the lower bounds on $f_d(n, m)$ have coordinates that are at least exponential in the number of points. Recall that the *spread* (also known as *aspect ratio*) of a point set is the ratio of the diameter and the closest pair distance. It is well-known that the spread of any set of n points in $\mathbb{R}^d$ is $\Omega(n^{1-1/d})$. It is natural to define $f_d(n, m, \Phi)$ as the quantity analogous to $f_d(n, m)$, where we only consider sets of n points in $\mathbb{R}^d$ having spread at most Φ.

▶ **Problem 14.** *Determine the value of* $f_d(n, m, \Phi)$.

For any set S of n points in $\mathbb{R}^d$, where $d = 2$, Xue *et al.* [9] have presented a data structure of size $O(n \log n)$ that can be used to answer vertical slab closest pair queries in $O(\log n)$ time. Our data structure uses only $O(n)$ space and works in any constant dimension $d \geq 2$. However, its query time is $O(n^{1/2+\varepsilon})$.

▶ **Problem 15.** *Is there a linear space data structure that supports vertical slab closest pair queries in* $o(\sqrt{n})$ *time, or even in* $O(\text{polylog}\,(n))$ *time?*

Another interesting research direction is to design linear space data structures for closest pair queries with other types of ranges, such as axes-parallel 3-sided ranges and general axes-parallel rectangular ranges.

References

1 Jon Louis Bentley and Michael I. Shamos. Divide-and-conquer in multidimensional space. In *Proceedings of the 8th ACM Symposium on the Theory of Computing*, pages 220–230, 1976. doi:10.1145/358841.358850.

2 Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42(1):67–90, 1995. doi:10.1145/200836.200853.

3 Timothy M. Chan, Saladi Rahul, and Jie Xue. Range closest-pair search in higher dimensions. *Comput. Geom.*, 91:101669, 2020. doi:10.1016/J.COMGEO.2020.101669.

4 Kurt Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*. Monographs in Theoretical Computer Science. Springer-Verlag, Berlin, 1984. doi:10.1007/978-3-642-69900-9.

5 R. Sharathkumar and Prosenjit Gupta. Range-aggregate proximity queries. Technical Report IIIT/TR/2007/80, International Institute of Information Technology Hyderabad, 2007.

6 Michiel Smid. Range trees with slack parameter. *Algorithms Review*, 2:77–87, 1991. URL: https://pure.mpg.de/rest/items/item_1834999_3/component/file_2463090/content.

7 Jie Xue. Colored range closest-pair problem under general distance functions. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 373–390, 2019. doi:10.1137/1.9781611975482.24.

8 Jie Xue, Yuan Li, Saladi Rahul, and Ravi Janardan. Searching for the closest-pair in a query translate. *J. Comput. Geom.*, 11(2):26–61, 2020. doi:10.20382/JOCG.V11I2A3.

9 Jie Xue, Yuan Li, Saladi Rahul, and Ravi Janardan. New bounds for range closest-pair problems. *Discret. Comput. Geom.*, 68(1):1–49, 2022. doi:10.1007/S00454-022-00388-7.

Online Routing in Directed Yao_4^∞ Graphs

Prosenjit Bose
School of Computer Science, Carleton University, Ottawa, Canada

Jean-Lou De Carufel
School of Electrical Engineering and Computer Science, University of Ottawa, Canada

John Stuart
School of Electrical Engineering and Computer Science, University of Ottawa, Canada

Abstract

The $\vec{\text{Yao}}_4^\infty$ and $\vec{\text{Yao}}_4$ graphs are two families of directed geometric graphs whose vertices are points in the plane, and each vertex has up to four outgoing edges. Consider a horizontal and a vertical line through each vertex v, defining four quadrants around v. Then v has an outgoing edge to the *closest* vertex in each of its four quadrants. When the distance is measured using the Euclidean norm, the resulting graph is the $\vec{\text{Yao}}_4$ graph, whereas with the L_∞-norm, we obtain the $\vec{\text{Yao}}_4^\infty$ graph, which is a sub-graph of the well-known L_∞-Delaunay graph.

In this paper, we provide a local routing algorithm with routing ratio at most 85.22 for $\vec{\text{Yao}}_4^\infty$ graphs. Prior to this work, no constant spanning or routing ratios for $\vec{\text{Yao}}_4^\infty$ graphs were previously known. Now, $\vec{\text{Yao}}_4^\infty$ graphs are the sparsest family of directed planar graphs supporting a competitive local routing strategy. Furthermore, we show that no local routing algorithm for $\vec{\text{Yao}}_4^\infty$ graphs can have a routing ratio lower than $7 + \sqrt{2} \approx 8.41$. Moreover, we prove that the spanning ratio is at least $5 + \sqrt{2} \approx 6.41$ in the worst case. The techniques we develop in this paper also allow us to prove lower bounds of $7 - \sqrt{3} + \sqrt{5 - 2\sqrt{3}} \approx 6.51$ and $7 + \sqrt{2}$ for the spanning and routing ratios of $\vec{\text{Yao}}_4$, respectively.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases Geometric Spanners, Yao Graphs, Local Routing Algorithms

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.9

Funding This research is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

1 Introduction

A fundamental problem in geometric routing is to construct a directed planar graph on a given set of points that supports competitive local routing, using the fewest number of edges. A routing algorithm is competitive if the length of the path that it finds is at most a constant times the length of the shortest path, however a local algorithm does not have access to the full graph. This problem has applications in wireless sensor networks, robotic navigation, and distributed computing. Routing is inherently directional, and in many real-world settings, signals are sent from a transmitter to an antenna. This directional nature is particularly relevant in wireless communication, where signal transmission can be costly. Wide-angle broadcasting consumes excessive power, and long-distance transmission is expensive, with energy consumption proportional to the distance squared. As a result, nodes should send signals in restricted directions, ensuring that each transmitted signal has a unique receiver. To reduce the effects of interference, we desire planarity, meaning that signals do not cross. However, we allow bidirectional edges, which count as two directed edges, since otherwise, not all point sets admit strongly-connected directed planar graphs. Disallowing bidirectional edges presents different challenges and has led to the study of oriented spanners [12]. Note that every undirected graph can be converted into a directed graph by replacing every

© Prosenjit Bose, Jean-Lou De Carufel, and John Stuart;
licensed under Creative Commons License CC-BY 4.0

undirected edge with two directed edges. This implies that every routing result on undirected graphs can also be viewed as a routing result on directed graphs with twice as many edges. Thus, the challenge is to construct directed planar graphs that support local routing while minimizing the number of directed edges.

Routing on graphs in an online fashion, without relying on precomputed routing tables is a compelling motivation for studying routing on geometric embeddings of graphs. The use of routing tables can be circumvented by exploiting geometric properties of the graph, such as empty proximity regions, to guide path-finding. Routing tables are often inefficient to compute and difficult to maintain in a dynamic setting, compared to maintaining a geometric embedding of a graph. Embeddings like Yao and Theta graphs provide a framework for leveraging these empty proximity regions to facilitate efficient routing. Our focus on the $\vec{\text{Yao}}_4^\infty$ graph stems from its unique combination of properties: it is planar, has a bounded out-degree of 4, and supports a novel competitive local routing algorithm.

For decades, researchers have been studying the distance-preserving properties of geometric graphs, such as the Delaunay triangulation [17]. The vertex set of a geometric graph is a finite set of points in the plane and the edges are weighted by the L_2-norm. The Delaunay triangulation has an edge between two vertices exactly when they lie on the boundary of a disk which contains no other vertex in its interior [15]. When the disk is replaced with a square, then we obtain an L_∞-Delaunay graph. Also a geometric graph of interest is the Yao graph. For any $k > 2$, a Yao_k graph is constructed by considering k equal-sized cones around each vertex and connecting each vertex to the *nearest* point in each cone [18]. In this way, a graph on n points has at most nk edges. Normally, *nearest* is defined using the L_2-norm, however in the case of $k = 4$, replacing L_2 with L_∞ is a natural choice because the resulting graph, denoted Yao_4^∞, is a subgraph of the well-known L_∞-Delaunay graph. In other words, the Yao_4^∞ graph has an edge between two vertices when there exists an empty square with one endpoint on a corner and the other endpoint on an opposing side. If the graph is defined using the L_1-norm, then we obtain the well-studied Θ_4-graph.

Given a geometric graph G, the spanning ratio of a subgraph H is a measure of how well distances of G are preserved. A subgraph H is called a c-spanner of G if for every edge $uv \in G$, the shortest path in H from u to v is at most c times the length of the edge uv in G [17]. The smallest such c is referred to as the spanning ratio of H. Throughout this paper, we consider G to be the complete graph with edges weighted by Euclidean distance. The undirected Yao_4 graph was first proved to be a spanner by Bose et al. [9] with a stretch factor of approximately 662. As part of the proof, the authors established that the undirected Yao_4^∞ graph is an 8-spanner. Then, Bonichon et al. [6] proved that the Yao_4^∞ graph has spanning ratio at most 6.31. Later, Damian and Nelavalli [14] improved both results by showing that the Yao_4 and Yao_4^∞ graphs have spanning ratios of at most 54.62 and 4.93, respectively. We will denote the directed versions of these graphs using an arrow: for example, the directed version of Yao_4^∞ is denoted $\vec{\text{Yao}}_4^\infty$. Despite being a natural construction, the $\vec{\text{Yao}}_4^\infty$ graph previously had no known upper bound on the spanning ratio. The spanning ratio of the L_∞-Delaunay graph was originally shown to be at most $\sqrt{10} \approx 3.16$ by Chew [13]. Then, Bonichon et al. [5] improved the spanning ratio of the L_∞-Delaunay triangulation to $\sqrt{4 + 2\sqrt{2}} \approx 2.61$, which is tight. Their lower bound construction also shows that Yao_4^∞ has a spanning ratio of at least 2.61.

While spanning ratios deal with the existence of short paths, routing ratios are based on finding short directed paths with only local information. Roughly speaking, a path-finding algorithm has routing ratio c if the paths produced have length at most c times the straight-line distance from source to destination. In the context of planar graphs, Chew gave

a local routing algorithm for L_∞-Delaunay graphs in 1986 [13]. An L_∞-Delaunay graph can contain nearly $6|V|$ directed edges in the worst case. Since then, Bonichon et al. [4] defined a number of bounded-degree planar spanners, including G_9 and G_{12}. In [10], local routing algorithms were provided for these graphs. A parametrized construction, denoted $\mathcal{MBDG}$, was used in [2] to attain several lightness properties. In all of these planar graphs, the number of directed edges required is still on the order of $6|V|$. In this paper, we define a local routing algorithm for the $\vec{\text{Yao}}_4^\infty$ graph, which contains at most $4|V|$ directed edges, which is the fewest number of edges among all planar directed graphs that admit a local routing algorithm.

In any bounded out-degree graph, each vertex only needs to store a constant amount of information to fully describe its outgoing neighbourhood. As a result, bounded out-degree directed graphs are a natural setting for local routing. For example, when k is larger than 6, the optimal local routing algorithm for $\vec{\text{Yao}}_k$ graphs is simply *cone routing*, where each decision is to move to the neighbour in the same cone as the destination. For smaller values of k, the $\vec{\text{Yao}}_k$ graph is less dense and cone routing can fail to produce short paths. However, [16] provides a local routing algorithm for the $\vec{\text{Yao}}_6$ graph with routing ratio at most 22.94. The smallest possible k for which the $\vec{\text{Yao}}_k$ graph is strongly connected is $k = 4$, and Bose et al. [11] recently provided a local routing algorithm achieving a routing ratio of at most 23.36 for $\vec{\text{Yao}}_4$ graphs. Furthermore, they show how to modify the path output by their local routing algorithm to upper bound the spanning ratio by 16.54 for undirected Yao_4 graphs. Interestingly, their algorithm is based on the Greedy/Sweep algorithm from [7] which has a routing ratio of 17 in the directed $\vec{\Theta}_4$ graph.

Our local routing algorithm extends the Greedy/Sweep approach from $\vec{\Theta}_4$ graphs to $\vec{\text{Yao}}_4^\infty$ graphs. The original Greedy/Sweep algorithm is quite simple. If the current vertex is u and the target vertex is t, then we consider the right triangle T bounded by vertical and horizontal lines through u and a *diagonal* line through t with slope -1. If no vertices are in T, we say that T is *clean* and we take a *greedy* step in the cone containing t. Otherwise, if T is not *clean*, then we take a *sweeping* step in the cone of T. The authors of [11] adapted the Greedy/Sweep approach to $\vec{\text{Yao}}_4$ graphs by replacing the right triangle with a wedge. However, in the case of $\vec{\text{Yao}}_4^\infty$ graphs, a direct application of the Greedy/Sweep approach fails miserably, resulting in long paths. The main challenge is that *sweeping* steps can cross the diagonal while making negligible progress towards the destination. To overcome this, we modify the definition of *clean* to penalize such diagonal crossings and force provable progress toward the target. Given our redefinition of cleanliness, we refer to our local routing algorithm as the DIRTY Algorithm (**D**irected **I**nfinity **R**outing **T**hrough **Y**ao). Our analysis yields an upper bound of 85.22 on the routing ratio of our local routing algorithm.

Notice that our routing ratio of 85.22 is also the best known upper bound on the spanning ratio for $\vec{\text{Yao}}_4^\infty$ graphs. Next, we give a lower bound of $5 + \sqrt{2} \approx 6.41$ on the spanning ratio of $\vec{\text{Yao}}_4^\infty$ graphs in the worst case, highlighting a clear gap compared to the upper bound of 4.93 for Yao_4^∞ graphs. Finally, we prove that no local routing algorithm for $\vec{\text{Yao}}_4^\infty$ graphs can achieve a routing ratio below $7 + \sqrt{2} \approx 8.41$. Additionally, we use a similar technique to provide lower bounds of $7 - \sqrt{3} + \sqrt{5 - 2\sqrt{3}} \approx 6.51$ and $7 + \sqrt{2}$ for the spanning and routing ratios of $\vec{\text{Yao}}_4$, respectively.

2 Preliminaries

For any point $u \in \mathbb{R}^2$, we let $x(u)$ and $y(u)$ denote the x- and y-coordinates of u, respectively. We denote the line segment between points u, v as uv, and define $\|uv\|_x := |x(u) - x(v)|$ and $\|uv\|_y := |y(u) - y(v)|$. For $p \in [1, \infty)$, the L_p-length of uv is denoted $\|uv\|_p :=$

Table 1 Comparison of upper bounds on the routing ratio of geometric graphs with bounded out-degree. Routing in $\mathcal{MBDG}$ and G_9 requires extra information to be stored at each vertex.

Reference	Graph	Planar	Max out-degree	Routing Ratio
[11]	$\vec{\text{Yao}}_4$	No	4	23.36
[16]	$\vec{\text{Yao}}_6$	No	6	22.94
[3]	$\vec{\text{Yao}}_k$ $(k \geq 7)$	No	k	$\leq 1/(1-2\sin(\pi/k))$
[7]	$\vec{\Theta}_4$	No	4	17
[1]	$\vec{\Theta}_6$	No	6	8
[8]	$\vec{\Theta}_k$ $(k \geq 7)$	No	k	$\leq 1/(1-2\sin(\pi/k))$
[2]	$\mathcal{MBDG}$	Yes	≥ 20	≥ 9.27
[10]	G_{12}	Yes	12	54.85
[10]	G_9	Yes	9	8.66
This paper	$\vec{\text{Yao}}_4^\infty$	Yes	4	85.22

$(\|uv\|_x^p + \|uv\|_y^p)^{\frac{1}{p}}$. We also define $\|uv\|_\infty := \max(\|uv\|_x, \|uv\|_y)$. A geometric graph is a graph whose vertex set contains points in the plane and whose edge weights are the L_2 (Euclidean) lengths of the corresponding line segments. In a geometric graph, each vertex is identified with its coordinates. Throughout the paper, we make the *general position assumption* that no two vertices have the same x- or y-coordinates and all distances between pairs of vertices are unique in the L_∞-norm. On such a point set, the edges of the $\vec{\text{Yao}}_4^\infty$ graph are well-defined. For two vertices u, v in a geometric graph G, the length of the shortest path from u to v in G is denoted $d_G(u, v)$. Then for a constant $c \geq 1$, G is said to be a c-spanner if for all points u, v in G, we have $d_G(u, v) \leq c\|uv\|_2$. The spanning ratio of G is the least c for which G is a c-spanner. The spanning ratio of a class of graphs $\mathcal{G}$ is the least c for which all graphs in $\mathcal{G}$ are c-spanners.

We make the assumption that the graph is embedded on a polynomial-sized grid and therefore the coordinates of the points require $O(\log(n))$ bits. Formally, an m-memory local routing algorithm is a function that takes as input $(s, N(s), t, M)$, and outputs some memory M' and a vertex $p \in N(s)$ where s is the current vertex, $N(s)$ is the outgoing neighbourhood of s, t is the destination, and both M, M' are bit-strings of length m. An algorithm is said to be c-competitive for a family of geometric graphs $\mathcal{G}$ if the path output by the algorithm for any pair of vertices $s, t \in V(G)$ for $G \in \mathcal{G}$ has length at most $c\|st\|_2$. The routing ratio of an algorithm is the least c for which the algorithm is c-competitive for $\mathcal{G}$. Note that the routing ratio is an upper bound on the spanning ratio.

3 A Local Routing Algorithm for $\vec{\text{Yao}}_4^\infty$ Graphs

We will now describe our local routing algorithm for $\vec{\text{Yao}}_4^\infty$ graphs. Without loss of generality, assume the coordinates of t are $(0, 0)$ and $\|st\|_\infty = 1$. Let $0 < \delta < 0.5$, and define the *green square* to be the set of points $p \in \mathbb{R}^2$ such that $\|pt\|_\infty \leq 2\delta$. In addition, we define the diagonal lines ℓ_t^-, ℓ_t^+ with slope ± 1 passing through t. Define the north, east, south and western quadrants as the cones delimited by ℓ_t^+ and ℓ_t^-. Refer to the four cones as , , , , respectively.

Next, for a point $p \in \mathbb{R}^2$, we define its *height*, $h(p)$, as the L_1 distance from p to the nearest diagonal through t. Notice that $h(p) = ||x(p)| - |y(p)||$. The *bands*, denoted by , are the regions close to the diagonals, excluding the green square. More precisely, let

refer to the set of points $p \in \mathbb{R}^2$ such that $p \notin$ and $h(p) \leq \delta$. Refer to Figure 1. Note that $\subset \mathbb{R}^2$ and is composed of four regions, which we label , clockwise from the north-eastern region. Then define the truncated wedges as those points $p \in \mathbb{R}^2$ such that $p \notin$ $\cup$. We have $\subset \mathbb{R}^2$ and since is composed of four regions, we label them , clockwise from the northern region.

Next we describe the terminology used in our local routing algorithm, which we will refer to as DIRTY Algorithm 1 (**D**irected **I**nfinity **R**outing **T**hrough **Y**ao). For any vertex $u \in$, a *band step* from u follows the edge in the cone of u containing t. For any vertex $u \in$, u is *clean* if the quadrant of u facing the closest diagonal $\ell_t \in \{\ell_t^-, \ell_t^+\}$ does not contain any vertex v such that $\|uv\|_\infty + \delta \leq h(u)$. Intuitively, u is clean if there are no points between u and the nearest diagonal. For example, see Figure 1. A sweep step from u follows the edge in the cone towards the closest diagonal to u. We denote the vertex resulting from a sweep step from u as $\mathbf{Sweep}(u, t, N(u), \delta)$. A greedy step from u follows the edge in the cone of u containing t. The vertex resulting from a greedy step from u is denoted $\mathbf{Greedy}(u, t, N(u))$. Note that a band step is technically a greedy step, however we will distinguish band steps from greedy steps in the analysis. Now we will present the local routing algorithm. In short, if there is a neighbour in the green square, then we choose it and rescale δ. On the other hand, we take a greedy edge if the current vertex is in or is clean. When the current vertex is not clean, we take a sweep edge. A partial trace of Algorithm 1 is shown in Figure 1. The constant $0 < c < 1$ in step 1 of Algorithm 1 will be set to 0.08, however the analysis holds for a general c. Furthermore, δ should be set to $c\|st\|_\infty$ for the first call of Algorithm 1.

Algorithm 1 **DIRTY** (**D**irected **I**nfinity **R**outing **T**hrough **Y**ao)
Local routing decision in $\vec{\text{Yao}}_4^\infty$ graph.

Data:
- Current vertex $u \in S$
- Destination vertex $t \in S$
- Neighbourhood $N(u)$ of u
- Parameter $\delta \in \mathbb{R}_{\geq 0}$

Result: A neighbour $v \in N(u)$ and a parameter $\delta' \in \mathbb{R}_{\geq 0}$
1. If $N(u) \cap$ $\neq \emptyset$, select an arbitrary $v \in N(u) \cap$ and set $\delta' := c\|vt\|_\infty$. END
2. Set $\delta' := \delta$. If $u \in$ or u is clean, then $v := \mathbf{Greedy}(u, t, N(u))$. Else, $v := \mathbf{Sweep}(u, t, N(u), \delta)$.

3.1 Analysis of DIRTY Algorithm 1

If edge uv is chosen by Algorithm 1, then notice that $\|ut\|_\infty \geq \|vt\|_\infty$ regardless of whether uv is a sweep, a greedy, or a band edge. In fact, by the general position assumption that all vertices have distinct x- and y-coordinates, we have $\|ut\|_\infty > \|vt\|_\infty$.

▶ Remark 1. If uv is an edge chosen by Algorithm 1, then $\|ut\|_\infty > \|vt\|_\infty$. That is, the vertex v is inside the square centered at t with u on the boundary.

The vertex set being finite means that choosing each successive edge according to Algorithm 1 will output a path from s to t, which we will denote $\mathcal{P}$. For any two vertices u, v on path $\mathcal{P}$ with v occurring after u, we let $\mathcal{P}_{u,v}$ denote the sub-path of $\mathcal{P}$ from u to v.

We define t' to be the first vertex of $\mathcal{P}$ chosen by step 1 of Algorithm 1. In other words, t' is the first vertex of $\mathcal{P}$ such that $\|t't\|_\infty \leq 2c\|st\|_\infty$ because δ is initially set to $c\|st\|_\infty$ and $t' \in$. We will focus on upper bounding the length of $\mathcal{P}_{s,t'}$, which is a path such that

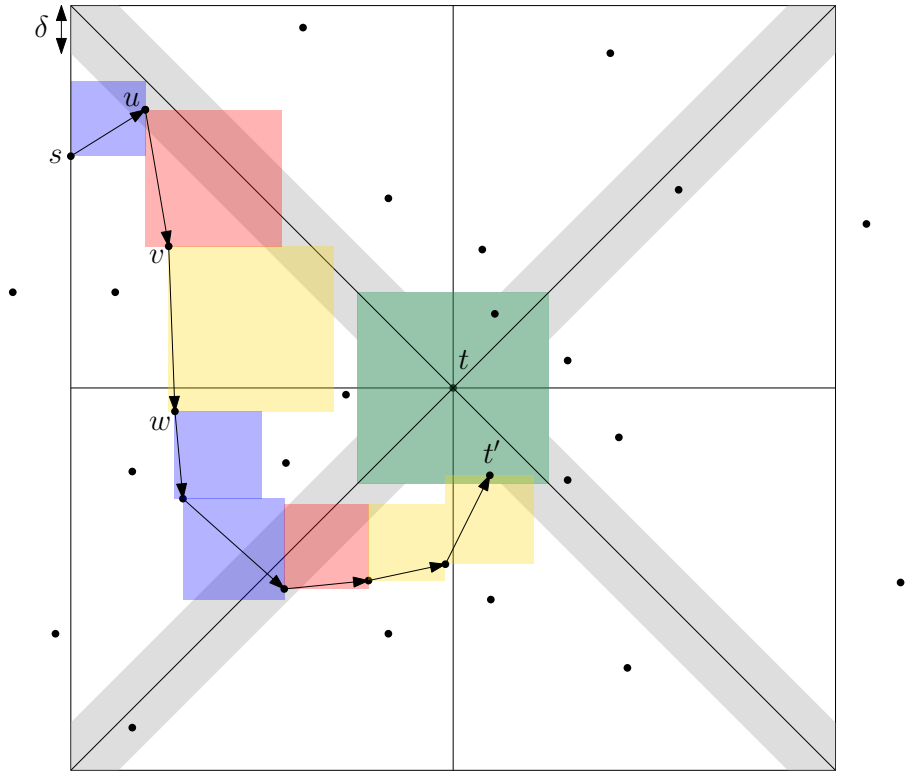


Figure 1 We show the partial trace of DIRTY Algorithm 1 until we reach a vertex in the green square, . The grey regions are the *bands*, denoted as , and the white regions are the truncated wedges, denoted . The sweep edges are blue, the band edges are red, and the greedy edges are gold. For example, su is a sweep step since s is not clean. Next, uv is a band step as $u \in$. Then v is clean, hence vw is a greedy edge.

each edge is chosen according to step 2 except the last edge. As such, we will only refer to edges of $\mathcal{P}_{s,t'}$. In particular, define the set $\mathcal{S}$ as the sweep edges of $\mathcal{P}_{s,t'}$ and define the set $\mathcal{G}$ as the greedy edges uv of $\mathcal{P}_{s,t'}$, where $u \in$. Next we define the set of band edges, $\mathcal{B}$, as the greedy edges uv of $\mathcal{P}_{s,t'}$, where $u \in$. These definitions allow us to partition the edges: $\{uv \mid uv \text{ is an edge on } \mathcal{P}_{s,t'}\} = \mathcal{S} \cup \mathcal{G} \cup \mathcal{B}$.

▶ **Lemma 2.** *If $uv \in \mathcal{S}$ and uv does not cross a diagonal, then $\|uv\|_2 \le h(u) - h(v)$.*

Proof. Since u, v both lie in the same cone among , , , , then $\|uv\|_2 \le \|uv\|_1 = h(u) - h(v)$. ◀

▶ **Lemma 3.** *If $uv \in \mathcal{S}$ and uv crosses a diagonal of t, then $h(u) - h(v) \ge 2\delta$, and $\max(\|uv\|_\infty, \|vt\|_\infty) \le \|ut\|_\infty - \delta$.*

Proof. Assume without loss of generality that $u \in$ and uv crosses ℓ_t^+. This means that $v \in$, as in Figure 2. Since u is not clean, we have $\|uv\|_\infty + \delta \le h(u) \le \|ut\|_\infty$. Also since v is above ℓ_t^+ we have $h(v) + \delta \le \|uv\|_\infty$. Combining these two inequalities yields $h(u) - h(v) \ge 2\delta$. Next, $u \in$ and $v \in$ together imply $h(u) = x(u) - y(u)$, so we have $\|ut\|_\infty - \|vt\|_\infty = x(u) - y(v) = h(u) + y(u) - y(v) \ge \|uv\|_\infty + \delta + y(u) - y(v) \ge \delta$. ◀

▶ **Lemma 4.** *If $uv \in \mathcal{G}$, then $h(v) - h(u) \le \|uv\|_\infty$.*

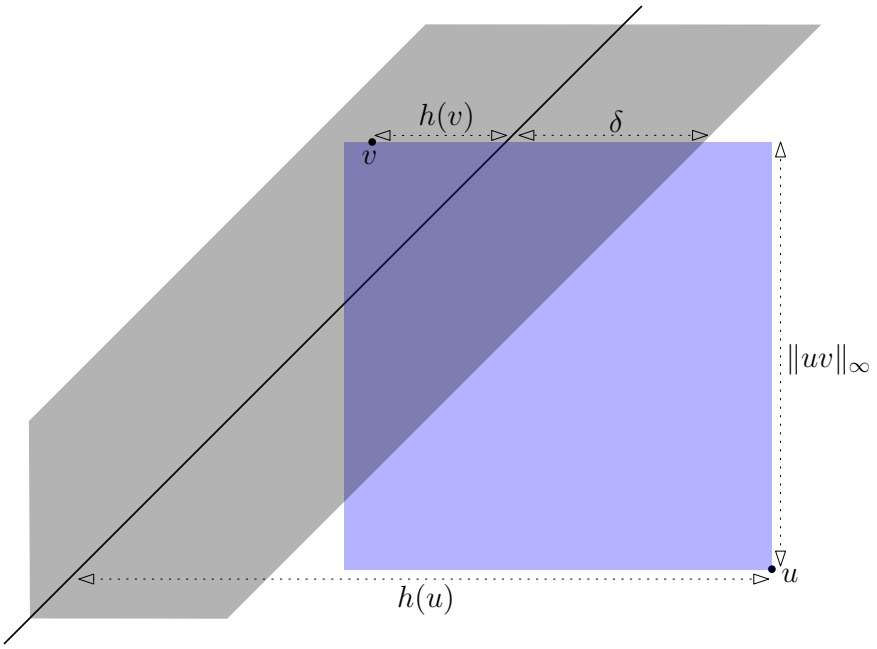


Figure 2 Lemma 3: An example of a sweep edge uv that crosses the diagonal ℓ_t^+ of t.

Proof. Suppose without loss of generality that $u \in$ ◀ with $y(u) > 0$. Then since uv is greedy, we have $x(v) < x(u) \leq x(v) + \|uv\|_\infty$ and $y(v) < y(u) \leq y(v) + \|uv\|_\infty$. Combining these inequalities yields

$$x(v) - y(v) \leq x(u) - y(u) + \|uv\|_\infty \text{ and } y(v) - x(v) \leq y(u) - x(u) + \|uv\|_\infty.$$

Since $0 \leq h(u) = |x(u) - y(u)|$, then we have shown $|x(v) - y(v)| \leq \|uv\|_\infty + h(u)$. The result follows from the reverse triangle inequality $h(v) = ||x(v)| - |y(v)|| \leq |x(v) - y(v)|$. ◀

Next we will characterize the behaviour of band edges.

▶ **Lemma 5.** *If $uv \in \mathcal{B}$, then* $\max_{\psi \in \{x,y\}} (\|uv\|_\psi + \|vt\|_\psi - \|ut\|_\psi) \leq 2\delta$.

Proof. Assume without loss of generality that $u \in$ ◤. Notice that $\|uv\|_y + \|vt\|_y - \|ut\|_y = (y(u) - y(v)) + |y(v)| - y(u) = |y(v)| - y(v)$, which is at most 2δ when $y(v) \geq -\delta$. Assume towards a contradiction that $y(v) < -\delta$. When u is above ℓ_t^+, then $\|uv\|_\infty \geq y(u) - y(v) > y(u) + \delta > y(u) = \|ut\|_\infty$. Otherwise when u is below the diagonal ℓ_t^+, we have $\|uv\|_\infty \geq y(u) - y(v) > y(u) + \delta \geq x(u) = \|ut\|_\infty$. Either way is a contradiction since t and v are in the same cone of u, so $\|uv\|_\infty \leq \|ut\|_\infty$. A symmetric argument completes the proof when $\psi = x$. ◀

Lemma 5 tells us that when uv is a band edge with $u \in$ ◤, then $v \in$ ◤∪◀∪◥∪■. Indeed, the proof describes why $y(v) \geq -\delta$ and $x(v) \geq -\delta$, and the region $\{p \in \mathbb{R}^2 \mid \min(x(p), y(p)) \geq -\delta\}$ is a subset of ◤∪◀∪◥∪■. In other words, a sequence of consecutive band edges remains in the same region of ◤◢ until it ends in a neighbouring truncated wedge, or in the green square. To formalize this, we will define a set $\mathcal{B}'$ containing pairs of vertices that correspond to maximal sub-paths of band edges.

▶ **Definition 6** (Maximal sYaoub-paths of band edges). *Define the set $\mathcal{B}'$ as follows. Let u, v be vertices of $\mathcal{P}_{s,t'}$ such that $\mathcal{P}_{u,v}$ is a sequence of band edges. We let the pair $(u, v) \in \mathcal{B}'$ if and only if $u \in$, $v \in$, and u is not the tip of a band edge.*

Notice that if $(u, v) \in \mathcal{B}'$, then $\mathcal{P}_{u,v}$ must be x- and y-monotone since all edges between u and v are band edges in the same cone.

▶ **Lemma 7.** *For all $(u, v) \in \mathcal{B}'$, we have $h(v) - h(u) \leq \|uv\|_\infty$.*

Proof. Suppose without loss of generality that $u \in$ and $v \in$. Then we have $x(v) < x(u) \leq x(v) + \|uv\|_\infty$ and $y(v) < y(u) \leq y(v) + \|uv\|_\infty$. The rest of the proof is identical to that of Lemma 4. ◀

To better understand which band edges are accounted for by paths in $\mathcal{B}'$, we provide the following definition.

▶ **Definition 8** (Vertices v' and u'). *Let v' be the last vertex of $\mathcal{P}_{s,t'}$ in . If no such v' exists, $v' := s$. Then, let u' be the vertex directly after v' on $\mathcal{P}_{s,t'}$.*

Notice that all band edges of $\mathcal{P}_{s,v'}$ are contained in a sub-path $\mathcal{P}_{u,v}$ for some $(u, v) \in \mathcal{B}'$. In addition, the path $\mathcal{P}_{u',t'}$ consists entirely of band edges.

▶ **Lemma 9.** *We have* $\sum_{uv \in \mathcal{G}} (\|ut\|_\infty - \|vt\|_\infty) + \sum_{(u,v) \in \mathcal{B}'} (\|ut\|_\infty - \|vt\|_\infty) \leq 1.$

Proof. Each difference $\|ut\|_\infty - \|vt\|_\infty$ in the following telescoping sum corresponds to a sub-path of $\mathcal{P}_{s,u'}$ and is therefore positive by Remark 1.

$$\sum_{uv \in \mathcal{G} \cup \mathcal{S}} (\|ut\|_\infty - \|vt\|_\infty) + \sum_{(u,v) \in \mathcal{B}'} (\|ut\|_\infty - \|vt\|_\infty) = \|st\|_\infty - \|u't\|_\infty \leq 1.$$

Dropping the terms corresponding to sweep edges yields the desired inequality. ◀

Now we can establish a bound on the sweep edges that depends on the greedy edges and $\mathcal{B}'$ paths.

▶ **Lemma 10.** *Let $k := \lfloor \frac{1-2\delta}{\delta} \rfloor + 1$. We have*

$$\sum_{uv \in \mathcal{S}} \|uv\|_2 \leq 1 + \sqrt{2}(k - \delta \frac{k(k+1)}{2}) - 2\delta k + \sum_{uv \in \mathcal{G}} \|uv\|_\infty + \sum_{(u,v) \in \mathcal{B}'} \|uv\|_\infty.$$

Proof. First, we split the sum based on whether the sweeping edge crosses a diagonal. Let $\mathcal{X} \subseteq \mathcal{S}$ denote the set of sweep edges that cross a diagonal ℓ_t^+ or ℓ_t^-.

$$\begin{aligned}
\sum_{uv \in \mathcal{S}} \|uv\|_2 &= \sum_{uv \in \mathcal{S} \setminus \mathcal{X}} \|uv\|_2 + \sum_{uv \in \mathcal{X}} \|uv\|_2 \\
&\leq \sum_{uv \in \mathcal{S} \setminus \mathcal{X}} (h(u) - h(v)) + \sum_{uv \in \mathcal{X}} \|uv\|_2 && \text{By Lemma 2} \\
&= \sum_{uv \in \mathcal{S}} (h(u) - h(v)) + \sum_{uv \in \mathcal{X}} (\|uv\|_2 - h(u) + h(v))
\end{aligned}$$

Next, by Definition 8, the path $\mathcal{P}_{s,u'}$ is precisely all the greedy edges, sweep edges, and sub-paths of band edges $\{\mathcal{P}_{u,v} \mid (u, v) \in \mathcal{B}'\}$. This means that

$$\sum_{uv \in \mathcal{S} \cup \mathcal{G}} (h(u) - h(v)) + \sum_{(u,v) \in \mathcal{B}'} (h(u) - h(v)) = h(s) - h(u') \tag{1}$$

Next we rearrange (1) and use Lemmas 4 and 7:

$$\sum_{uv\in\mathcal{S}}(h(u)-h(v)) = h(s)-h(u')+\sum_{uv\in\mathcal{G}}(h(v)-h(u))+\sum_{(u,v)\in\mathcal{B}'}(h(v)-h(u))$$
$$\leq 1+\sum_{uv\in\mathcal{G}}\|uv\|_\infty+\sum_{(u,v)\in\mathcal{B}'}\|uv\|_\infty$$

Now, let $m := |\mathcal{X}|-1$, and label the edges of $\mathcal{X}$ as $u_0v_0, ..., u_mv_m$ in the order they appear on the path $\mathcal{P}_{s,t'}$. By Lemma 3, we have $\|u_it\|_\infty - \|v_it\|_\infty \geq \delta$ for all $i \in \{0, ..., m\}$. Additionally, u_{i+1} appears after v_i on the path $\mathcal{P}_{s,t'}$, then by Remark 1 we have $\|u_{i+1}t\|_\infty \leq \|v_it\|_\infty$ for $i \in \{0, ..., m-1\}$. Since $\|u_0t\|_\infty \leq \|st\|_\infty = 1$, the above inequalities imply that $\|u_it\|_\infty \leq 1-i\delta$ for $i \in \{0, ..., m\}$. Next, vertex t' was defined to be the first vertex on the path $\mathcal{P}$ such that $\|t't\|_\infty \leq 2\delta$, therefore we must have $\|u_mt\|_\infty > 2\delta$ since u_mv_m is an edge in $\mathcal{P}_{s,t'}$. Combining inequalities yields $1-m\delta > 2\delta$, and isolating m yields $m \leq \lfloor\frac{1-2\delta}{\delta}\rfloor = k-1$. Also, Lemma 3 gives us $\|u_iv_i\|_\infty + \delta \leq \|u_it\|_\infty \leq 1-i\delta$ and $h(u_i)-h(v_i) \geq 2\delta$. The result follows:

$$\sum_{uv\in\mathcal{X}}(\|uv\|_2 - h(u)+h(v)) \leq \sum_{i=0}^{m}(\sqrt{2}\|u_iv_i\|_\infty - h(u_i)+h(v_i)) \quad \text{By } \|uv\|_2 \leq \sqrt{2}\|uv\|_\infty$$
$$\leq \sum_{i=0}^{k-1}(\sqrt{2}(1-i\delta-\delta)-2\delta)$$
$$= \sqrt{2}(k-\delta\frac{k(k+1)}{2})-2\delta k. \qquad \blacktriangleleft$$

From Lemma 10, we can see that the total L_2 distance of sweep edges can be upper bounded if we upper bound the total L_∞ distance of greedy edges and pairs in $\mathcal{B}'$.

▶ **Lemma 11.** *We have* $\sum_{uv\in\mathcal{G}}\|uv\|_\infty \leq 16-8\delta$.

Proof. Consider the set $\mathcal{G}' \subseteq \mathcal{G}$ of greedy edges uv where $u \in$ ◖ and $y(u) > 0$. We will show that $\sum_{uv\in\mathcal{G}'}\|uv\|_\infty \leq 2-\delta$ using a disjoint projection argument illustrated in Figure 3.

Firstly, label the edges of $\mathcal{G}'$ as $u_1v_1, ..., u_mv_m$. Note that the edges are labelled in the order they appear, and not necessarily consecutive. For $i \in \{1, .., m\}$, define the point $p_i := u_i - (0, \|u_iv_i\|_\infty)$. Note that $\|u_iv_i\|_\infty = \|u_ip_i\|_y$. Next, define the projection $F : \mathbb{R}^2 \to \mathbb{R}^2$ as $F(x, y) := (0, x+y)$. Notice that for a point $p \in \mathbb{R}^2$, the point $F(P)$ lies on the vertical line through t with the segment $pF(p)$ having slope -1. We will show that the segments $\{F(u_i)F(p_i) \mid 1 \leq i \leq m\}$ are disjoint.

Let $i \in \{1, ..., m-1\}$. Firstly, since u_iv_i is greedy, we have $y(v_i) < y(u_i)$, hence $y(F(p_i)) < y(F(u_i))$. Next, we have $x(u_{i+1}) < x(u_i)$ by Remark 1 and $y(u_{i+1}) < y(u_i)$ since u_i is clean. Combining these coordinate restrictions with the fact that p_i lies on the southeast corner of the empty square corresponding to u_iv_i, we obtain $y(F(u_{i+1})) < y(F(p_i)) < y(F(u_i))$. This means that the projections are disjoint and their y-coordinates range from $y(F(p_m))$ to $y(F(u_1))$. Since $p_m \in$ ◭, then $y(F(p_m)) \geq 0$. Also, we have $y(F(u_1)) \leq 2-\delta$ given that $u_1 \in$ ◖.

The proof is completed after multiplication by 8 to account for the greedy edges of $\mathcal{G}$ originating in the seven other regions symmetric to $\mathcal{G}'$. ◀

At this stage, we could multiply the bound from Lemma 11 by $\sqrt{2}$ in order to bound the total L_2 length of the greedy edges. However, we can do better with a more careful analysis described in Lemma 12.

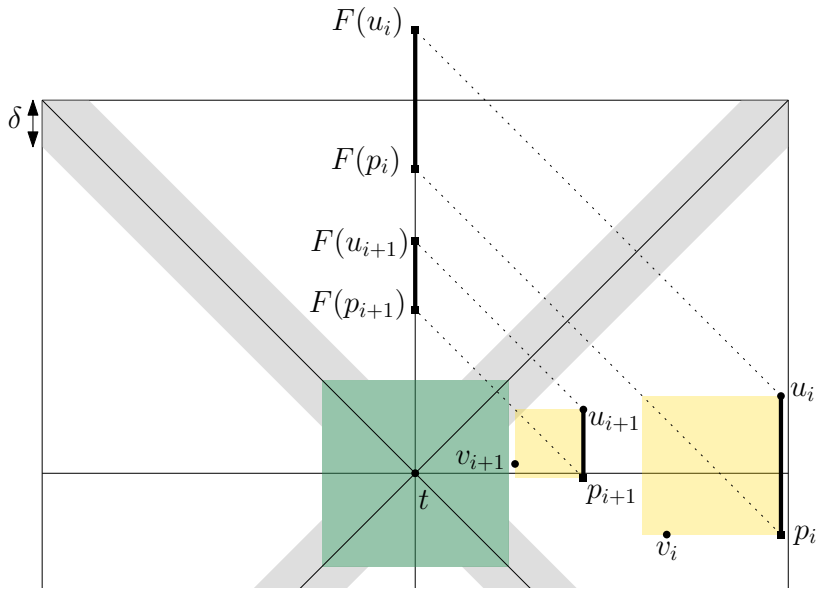


Figure 3 Lemma 11: The vertical components of the edges u_iv_i and $u_{i+1}v_{i+1}$ can be projected onto the y-axis using the projection F. Furthermore, the projections are disjoint.

▶ **Lemma 12.** *We have* $\sum_{uv\in\mathcal{G}} \|uv\|_2 \le 12 + 4\sqrt{2} - 8\delta + \sum_{uv\in\mathcal{G}} (\|ut\|_\infty - \|vt\|_\infty)$.

Proof. We start by partitioning $\mathcal{G}$ into three parts that correspond to the following three types of greedy edges, illustrated in Figure 4. Suppose without loss of generality that $uv \in \mathcal{G}$ and $u \in$ with $y(u) > 0$. Then the edge uv is Type A if $v \in$, Type B if $v \in$, and Type C if $v \in$. Now we extend the definition of Types A, B, and C to all edges in $\mathcal{G}$ by symmetry. For example, an edge $uv \in \mathcal{G}$ with $u \in$, $0 < x(u)$ and $v \in$ would be Type C. Define the sets $\mathbf{A}, \mathbf{B}, \mathbf{C}$ of Type A, B, and C greedy edges, respectively. This means that we have partitioned the greedy edges as follows: $\mathcal{G} = \mathbf{A} \cup \mathbf{B} \cup \mathbf{C}$. We proceed with the following claim for Type A and B edges.

▷ Claim. If $uv \in \mathbf{A} \cup \mathbf{B}$ then $\|uv\|_1 \le \|uv\|_\infty + \|ut\|_\infty - \|vt\|_\infty$.

Proof. Suppose without loss of generality that $u \in$ and $y(u) > 0$. If $uv \in \mathbf{A}$ then $v \in$, so $\|uv\|_x \le \|ut\|_\infty - \|vt\|_\infty$. Combining this with $\|uv\|_y \le \|uv\|_\infty$ yields the result. If instead $uv \in \mathbf{B}$, then $v \in$, giving us $\|uv\|_y \le \|ut\|_\infty - \|vt\|_\infty$. Adding this to $\|uv\|_x \le \|uv\|_\infty$ completes the claim. ◁

Next, we prove that for Type C edges, $\sum_{uv\in\mathbf{C}} \|uv\|_\infty \le 4$. Consider the greedy edges $u_1v_1, ..., u_mv_m \in \mathbf{C}$ with $u_i \in$ labelled in the order they appear in $\mathcal{P}_{s,t'}$. Then by the two empty squares (one from u_i being clean and one from u_iv_i), we have $\|u_{i+1}t\|_\infty \le \|u_it\|_\infty - \|u_iv_i\|_\infty$ for $i \in \{1, ..., m-1\}$ and $\|u_mv_m\|_\infty \le \|u_mt\|_\infty$. Then we have $\sum_{i=1}^{m} \|u_iv_i\|_\infty \le \|u_mt\|_\infty + \sum_{i=1}^{m-1} \|u_it\|_\infty - \|u_{i+1}t\|_\infty = \|u_1t\|_\infty \le 1$. The final step is multiplication by 4 to account for the four connected components of .

Now we combine the above claim with Lemma 11 to complete the proof.

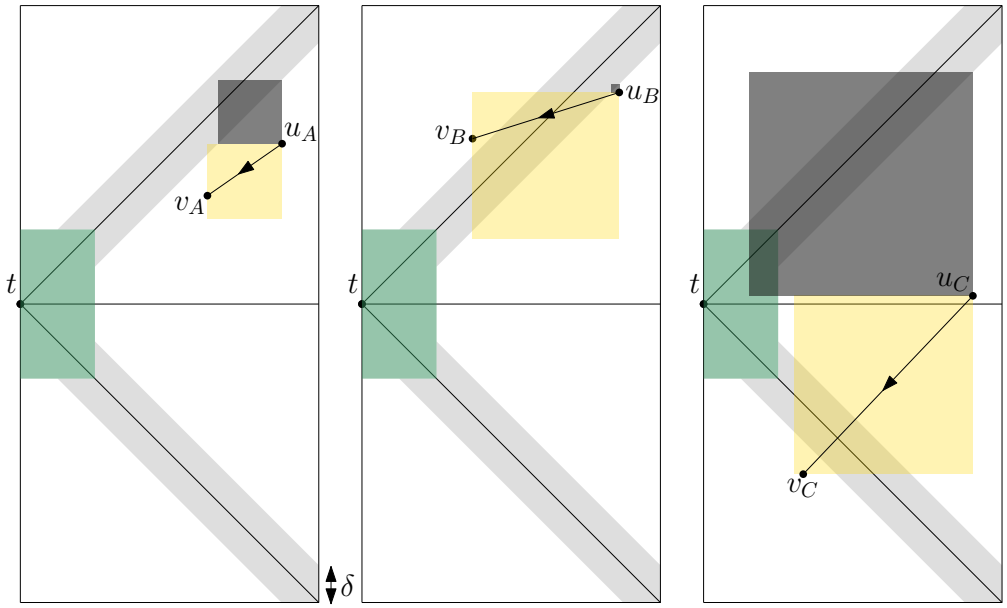


Figure 4 Lemma 12: Edges u_Av_A, u_Bv_B, u_Cv_C are examples of Type A, B and C greedy edges, respectively. Notice that u_A, u_B, u_C are all clean with respect to ℓ_t^+, meaning that the dark grey regions are empty.

$$\begin{aligned}
\sum_{uv\in\mathcal{G}} \|uv\|_2 &\le \sum_{uv\in\mathbf{A}\cup\mathbf{B}} \|uv\|_1 + \sum_{uv\in\mathbf{C}} \|uv\|_2 \\
&\le \sum_{uv\in\mathbf{A}\cup\mathbf{B}} (\|uv\|_\infty + \|ut\|_\infty - \|vt\|_\infty) + \sum_{uv\in\mathbf{C}} (\|uv\|_\infty + (\sqrt{2}-1)\|uv\|_\infty) \\
&\le \sum_{uv\in\mathcal{G}} (\|uv\|_\infty + \|ut\|_\infty - \|vt\|_\infty) + \sum_{uv\in\mathbf{C}} (\sqrt{2}-1)\|uv\|_\infty \\
&\le \sum_{uv\in\mathcal{G}} (\|ut\|_\infty - \|vt\|_\infty) + 16 - 8\delta + 4(\sqrt{2}-1) \qquad \blacktriangleleft
\end{aligned}$$

Next, a similar projection argument can be used to bound the total the length of band edges. First, we provide the following definition.

▶ **Definition 13.** *For $2\delta \le r \le 1$, define the eastern hexagon $\blacktriangleleft_r$ to be the set of points $p \in \blacktriangleleft$ such that $2\delta < x(p) < r$ and $|y(p)| < (r-\delta)$. Define the north, west and south hexagons of radius r similarly.*

Next, we present a lemma that formalizes the idea that when the path $\mathcal{P}$ exits $\blacktriangleleft_r$, either $\mathcal{P}$ becomes significantly closer to t, or $\mathcal{P}$ never revisits the region $\blacktriangleleft$.

▶ **Lemma 14.** *Suppose vertex $w \in \blacktriangleleft$, and let v be the next vertex of $\mathcal{P}$ outside of $\blacktriangleleft_{\|wt\|_\infty}$. Then at least one of the following must be true:*

1. $v \in \blacksquare$,
2. $\|vt\|_\infty \le \|wt\|_\infty - \delta$,
3. $\mathcal{P}_{v,t'} \cap \blacktriangleleft = \emptyset$

Proof. Throughout the proof, let u denote the vertex before v in $\mathcal{P}$, meaning $u \in \blacktriangleleft_{\|wt\|_\infty}$. Since $u \in \blacktriangleleft_{\|wt\|_\infty} \subset \blacktriangleleft$, then we must have $v \in \blacktriangledown \cup \blacksquare \cup \blacktriangle$. Without loss of generality, assume $v \in \blacktriangledown \setminus \blacksquare$, implying that $y(v) = \|vt\|_\infty$. Since we have assumed Item 1 to be false, we will show that in each of the following three cases, Item 2 or Item 3 is true. The cases are shown in Figure 5.

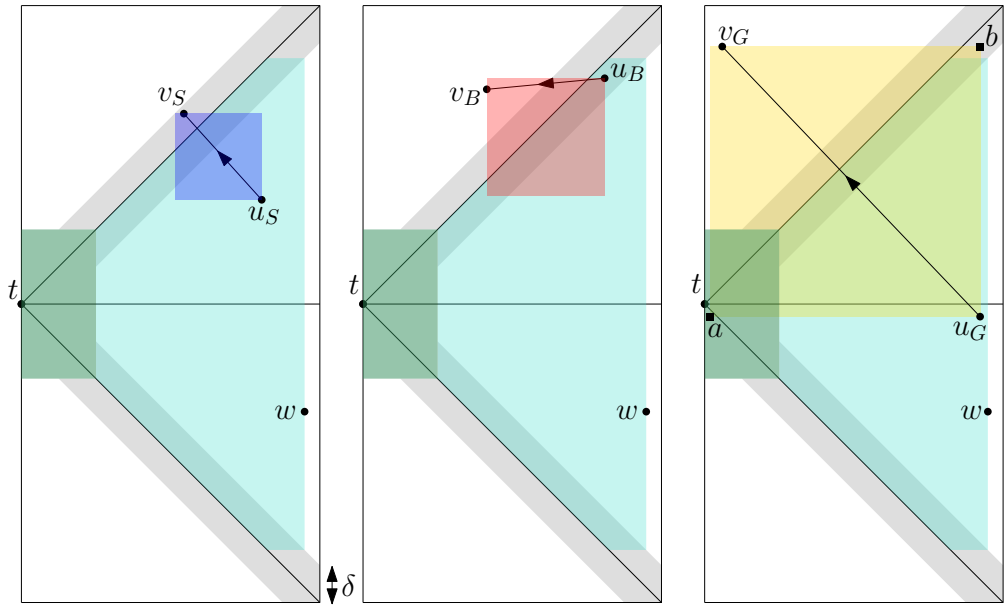


Figure 5 Lemma 14: Edges u_Sv_S, u_Bv_B, u_Gv_G are examples of sweep, band, and greedy edges, respectively. The turquoise region is $\blacktriangleleft_{\|wt\|_\infty}$. The labels a and b refer to the southwest and northeast corners of the gold square corresponding to edge u_Gv_G.

Case: uv is a sweep edge. Then we must have $u \in \blacktriangleleft$ and $y(u) > 0$, therefore $y(v) - y(u) \leq \|uv\|_\infty \leq h(u) - \delta = x(u) - y(u) - \delta$. Item (2) follows from $x(u) = \|ut\|_\infty \leq \|wt\|_\infty$.

Case: uv is a band edge. Since $v \in \blacktriangledown \setminus \blacksquare$, we must have $u \in \blacktriangleleft\blacktriangleright$, meaning $y(v) < y(u)$. By definition of $u \in \blacktriangleleft_{\|wt\|_\infty}$, we have $y(u) < \|wt\|_\infty - \delta$, which again satisfies Item (2).

Case: uv is a greedy edge. Assume $\|vt\|_\infty > \|wt\|_\infty - \delta$. We will show that the region $\{p \in \blacktriangleleft \mid \|pt\|_\infty < \|ut\|_\infty\}$ contains no vertices, implying Item (3). Since $v \in \blacktriangledown$, then the closest diagonal to u must be ℓ_t^-. Let points a, b be the southwest and northeast corners of the empty square corresponding to uv. Then $y(a) = y(u) < 0$ and $0 < x(a)$ since the square bounding a, b is empty. Furthermore, since $x(u) - \delta \leq \|wt\|_\infty - \delta \leq y(v) \leq y(b)$, we must have $b \in \blacktriangleleft\blacktriangleright$. This leads to $d_1(a, \ell_t^+) < \delta$, so $a \in \blacksquare$. Then since u is clean with respect to ℓ_t^-, and also $a \in \blacksquare$ and $b \in \blacktriangleleft\blacktriangleright$ then the region $\{p \in \blacktriangleleft \mid \|pt\|_\infty < \|ut\|_\infty\}$ contains no vertices. Item (3) follows since $\|vt\|_\infty < \|ut\|_\infty$ by Remark 1. ◀

We will also make use of the following remark:

▶ Remark 15. If $(u, v) \in \mathcal{B}'$, then v is clean with respect to the closest diagonal to u. Indeed, v is the tip of a band edge from the connected component of $\blacktriangleleft\blacktriangleright$ containing u by Lemma 5.

Now we have the tools to prove Lemma 16.

▶ **Lemma 16.** *We have* $\sum_{(u,v)\in\mathcal{B}'} \|uv\|_\infty \leq 16 - 20\delta$.

Proof. We will first focus on paths $\mathcal{P}_{u,v}$ where $(u, v) \in \mathcal{B}'$ and $v \in \blacktriangleleft$. Let $(u_1, v_1), ..., (u_m, v_m) \in \mathcal{B}'$ be labelled according to their order on $\mathcal{P}$, where $v_i \in \blacktriangleleft$. Recall that all edges between u_i and v_i are band edges, so by Lemma 5, we have $u_i \in \blacktriangleleft\blacktriangleright \cup \blacktriangleleft\blacktriangleright$. Then $\|u_iv_i\|_\infty = \|u_iv_i\|_y$. Consider the point $p_i := (x(u_i), y(v_i))$ and the resulting segment

u_ip_i with length $\|u_iv_i\|_y$. We will show how to project u_ip_i onto the line $x = 2\delta$ such that when $i \neq j$, the projections of u_ip_i and u_jp_j are disjoint, illustrated in Figure 6. Furthermore, we will show that all projections lie on a sub-segment of length $4 - 5\delta$.

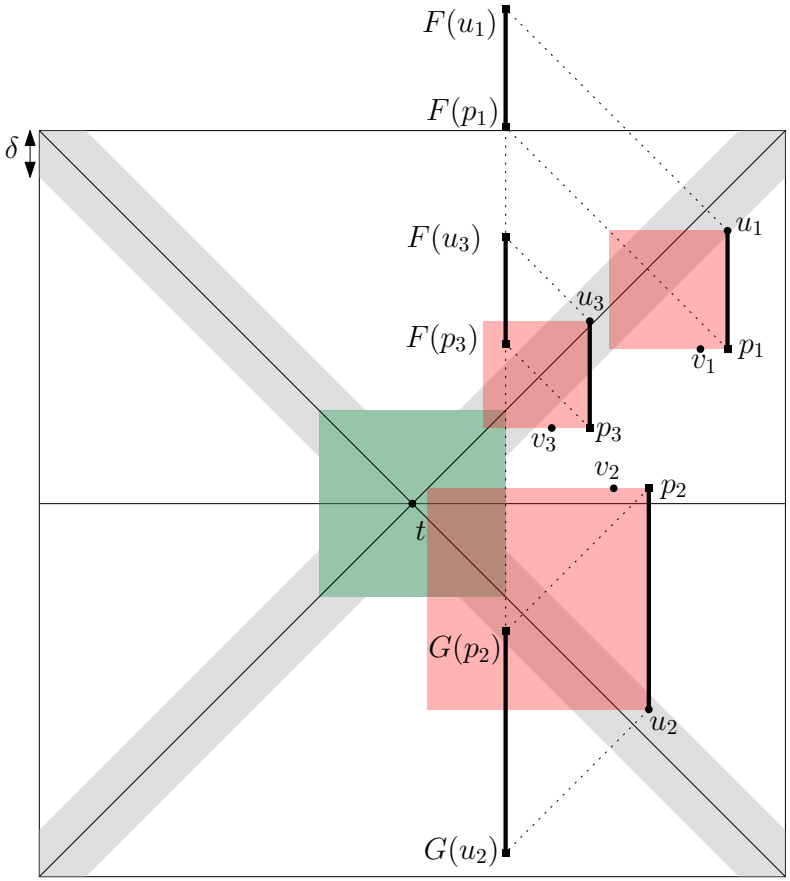


Figure 6 Lemma 16: We project the vertical components of banded sequences onto the dotted line, and prove that the projections are disjoint.

Define the projections $F, G : \mathbb{R}^2 \to \mathbb{R}^2$ as $F(x, y) := (2\delta, x + y - 2\delta)$ and $G(x, y) := (2\delta, y - x + 2\delta)$. Note that for $p \in \mathbb{R}^2$, both F and G project p onto the vertical line $x = 2\delta$, and the slope of the segments $pF(p)$ and $pG(p)$ are -1 and 1, respectively. For $1 \leq i \leq m$, we define the projection

$$\text{proj}(i) := \begin{cases} F(u_i)F(p_i) & \text{if } u_i \in \\ G(u_i)G(p_i) & \text{if } u_i \in \end{cases}$$

Let $1 \leq i < j \leq m$. Now we show that that proj(i) and proj(j) are disjoint by cases. Assume without loss of generality that $u_i \in$, so by Remark 15, v_i is clean with respect to ℓ_t^+. We will show that proj(j) is entirely below proj(i) in both cases.

Case: $u_j \in$. By Lemma 14, we have $y(u_j) < x(v_i) - \delta$. Indeed, $v_i \in$, therefore either $\mathcal{P}_{v_i,u_j} \subset \|_{v_it\|_\infty}$, or $\|u_jt\|_\infty \leq \|v_it\|_\infty - \delta$ since $u_j \notin$ and $v_j \in$. Furthermore, $x(u_j) \leq \|u_jt\|_\infty < \|v_it\|_\infty = x(v_i)$. Combining $x(u_j) < x(v_i)$ with $y(u_j) < x(v_i) - \delta$ and the fact that v_i is clean with respect to ℓ_t^+, we have that $y(F(u_j)) < y(F(v_i))$. Lastly, since $\mathcal{P}_{u_i,v_i}$ is x-monotone, then $x(u_i) > x(v_i)$, so $y(F(v_i)) < y(F(p_i))$. By transitivity, we have $y(F(u_j)) < y(F(p_i))$ completing this case.

Case: $u_j \in$. Assume towards a contradiction that $y(F(p_i)) \leq y(G(p_j))$. Then since $2\delta < x(p_j) < x(p_i)$ we must have $y(p_i) < y(p_j)$. However, this implies that v_i is not clean with respect to ℓ_t^+ since $v_j \in$ with $y(v_i) = y(p_i) < y(p_j) = y(v_j)$ and $x(v_j) < x(v_i)$.

Now that we have proven that the projections proj(i) are disjoint on the vertical line $x = 2\delta$, we can proceed to showing that the projections lie on a sub-segment of the line with length at most $4 - 5\delta$. Assume without loss of generality that $u_1 \in$. Then all projections lie below $y(F(u_1)) \leq y(F(1,1)) = 2 - 2\delta$. Furthermore, we will show that all projections lie above $y(G(1, \delta - 1)) = 3\delta - 2$. Indeed, if $u_i \in$ for all $1 \leq i \leq m$, then the projections are all above $y(F(p_m)) \geq y(F(1, \delta - 1)) = -\delta$. Otherwise, let j be the least index in $\{2, .., m\}$ such that $u_j \in$. Then all projections are above $y(G(u_j))$, and by Lemma 14 we have $|y(u_j)| \leq x(v_1) - \delta \leq 1 - \delta$. This means that $y(G(u_j)) \geq y(G(1, \delta - 1)) = 3\delta - 2$. Therefore all projections proj(i) have y-coordinates between $3\delta - 2$ and $2 - 2\delta$. Summarizing, we have shown

$$\sum_{\substack{(u,v)\in\mathcal{B}' \\ v\in}} \|uv\|_\infty = \sum_{\substack{(u,v)\in\mathcal{B}' \\ v\in}} \|uv\|_y \leq (2 - 2\delta) - (3\delta - 2) = 4 - 5\delta$$

Multiplying by 4 for the four connected components of concludes the proof. ◀

Now we can use Lemma 16 to bound the length of all band edges.

▶ **Lemma 17.** *We have* $\sum_{uv\in\mathcal{B}} \|uv\|_2 \leq 18 - 19\delta + \sum_{(u,v)\in\mathcal{B}'} (\|ut\|_\infty - \|vt\|_\infty)$.

Proof. By Definition 8, we can rewrite $\sum_{uv\in\mathcal{B}} \|uv\|_2 = \|\mathcal{P}_{u',t'}\|_2 + \sum_{(u,v)\in\mathcal{B}'} \|\mathcal{P}_{u,v}\|_2$. For now we focus on $\|\mathcal{P}_{u',t'}\|_2$. If $u' = t'$, then $\|\mathcal{P}_{u',t'}\|_2 = 0$. Otherwise, assume without loss of generality that $u' \in$ and $t' \in \cap$. Then $0 \leq x(t') < x(u')$ and by Lemma 5, $-\delta < y(t')$ since t' is the tip of a band edge. Since $\mathcal{P}_{u',t'}$ is x- and y-monotone, we have

$$\|\mathcal{P}_{u',t'}\|_2 \leq \|u't'\|_x + \|u't'\|_y \leq 1 + (1 + \delta)$$

Next, for $(u, v) \in \mathcal{B}'$, we have $\|\mathcal{P}_{u,v}\|_2 \leq \|uv\|_x + \|uv\|_y$ since $\mathcal{P}_{u,v}$ is x- and y- monotone. Also, $\min(\|uv\|_x, \|uv\|_y) = \|ut\|_\infty - \|vt\|_\infty$ since uv crosses a diagonal boundary of by Lemma 5, yielding

$$\begin{aligned}
\sum_{(u,v)\in\mathcal{B}'} \|\mathcal{P}_{u,v}\|_2 &\leq \sum_{(u,v)\in\mathcal{B}'} \big(\min(\|uv\|_x, \|uv\|_y) + \max(\|uv\|_x, \|uv\|_y)\big) \\
&\leq \sum_{(u,v)\in\mathcal{B}'} (\|ut\|_\infty - \|vt\|_\infty + \|uv\|_\infty) \\
&\leq 16 - 20\delta + \sum_{(u,v)\in\mathcal{B}'} (\|ut\|_\infty - \|vt\|_\infty) \qquad \text{by Lemma 16.}
\end{aligned}$$

◀

The bounds we have proved for sweep, greedy and band edges can now be combined to prove Theorem 18.

▶ **Theorem 18.** *DIRTY Algorithm 1 is an $O(\log(n))$-memory local routing algorithm with routing ratio of at most 85.22 for $\vec{Yao}_4^\infty$ graphs.*

Proof. We bound the total L_2 distance of sweep edges by combining Lemmas 10, 11 and 16:

$$\sum_{uv\in\mathcal{S}} \|uv\|_2 \leq 33 - 28\delta + \sqrt{2}(k - \delta\frac{k(k+1)}{2}) - 2\delta k \tag{2}$$

Finally, we can bound the length of the path from s to t'.

$$\begin{aligned}
\|\mathcal{P}_{s,t'}\|_2 &= \sum_{uv\in\mathcal{S}} \|uv\|_2 + \sum_{uv\in\mathcal{G}} \|uv\|_2 + \sum_{uv\in\mathcal{B}} \|uv\|_2 \\
&\leq (33 - 28\delta + \sqrt{2}(k - \delta\frac{k(k+1)}{2}) - 2\delta k) && \text{by (2)} \\
&+ (12 + 4\sqrt{2} - 8\delta + \sum_{uv\in\mathcal{G}} (\|ut\|_\infty - \|vt\|_\infty)) && \text{by Lemma 12} \\
&+ (18 - 19\delta + \sum_{(u,v)\in\mathcal{B}'} (\|ut\|_\infty - \|vt\|_\infty)) && \text{by Lemma 17} \\
&\leq 64 - 55\delta + \sqrt{2}(k - \delta\frac{k(k+1)}{2} + 4) - 2\delta k =: f(\delta) && \text{by Lemma 9.}
\end{aligned}$$

Notice that by the definition of step 1 of Algorithm 1, the path $\mathcal{P}_{t',t}$ is identical to the path that Algorithm 1 would output if the first vertex s was chosen to be t'. This means that the entire path from s to t has length at most $\frac{f(c)}{1-2c}$ because δ is initially set to $c\|st\|_\infty$ until an edge is chosen in step 1. The ratio evaluates to 85.22 when $c = 0.08$. Furthermore, $\|st\|_2 \geq \|st\|_\infty = 1$ by assumption. The only memory required to construct the path locally is to store δ for each successive iteration. Hence we have given a local routing algorithm with routing ratio at most 85.22 for $\vec{\text{Yao}}_4^\infty$ using at most $O(\log(n))$ bits of memory. ◀

4 Lower Bounds

▶ **Lemma 19.** *Let $\epsilon > 0$. There exists a $\vec{Yao}_4^\infty$ graph with spanning ratio at least $5 + \sqrt{2} - \epsilon$.*

Proof. Let $\epsilon > 0$ be arbitrarily small. We will create a graph with spanning ratio at least $5 + \sqrt{2} - \epsilon$, shown in Figure 7. We let $\delta > 0$ and choose its exact value later in the proof. Let $s := (0, -1) + \delta(3, 1)$, $a := (-1, -1) + \delta(6, 3)$, $b := (-1, 0) + \delta(7, -1)$, $c := (-1, 1) + \delta(8, -10)$, $d := (0, 1) + \delta(-1, -11)$, $e := (1, 1) + \delta(-13, -12)$, $f := (2, 1) + \delta(-13, -13)$, $g := (2, -1) + \delta(-12, +2)$, and $t = (0, 0)$. Let G be the $\vec{\text{Yao}}_4^\infty$ graph with vertices $\{s, a, b, c, d, e, f, g, t\}$. Then excluding the outgoing edges from t, G contains the following edges: $sa, sg, as, ab, ba, bs, bc, cb, cd, db, dc, de, ed, et, ef, fe, fs, fg, gf, gs$. Since the only incoming edge to t is et, then the shortest path from s to t must pass by e. The shortest path from s to e is (s, a, b, c, d, e) since

$$\begin{aligned}
&\|sg\|_2 + \|gf\|_2 + \|fe\|_2 \\
&\geq \|sg\|_\infty + \|gf\|_\infty + \|fe\|_\infty \\
&= (2 - 15\delta) + (2 - 15\delta) + 1 \\
&\geq (1 - 1\delta) + (1 - 3\delta) + (1 - 8\delta) + (1 - 8\delta) + (1 - 11\delta) \\
&= \|sa\|_1 + \|ab\|_1 + \|bc\|_1 + \|cd\|_1 + \|de\|_1 \\
&\geq \|sa\|_2 + \|ab\|_2 + \|bc\|_2 + \|cd\|_2 + \|de\|_2 \\
&\geq \|sa\|_\infty + \|ab\|_\infty + \|bc\|_\infty + \|cd\|_\infty + \|de\|_\infty \\
&= (1 - 3\delta) + (1 - 4\delta) + (1 - 9\delta) + (1 - 9\delta) + (1 - 12\delta) = 5 - 37\delta
\end{aligned}$$

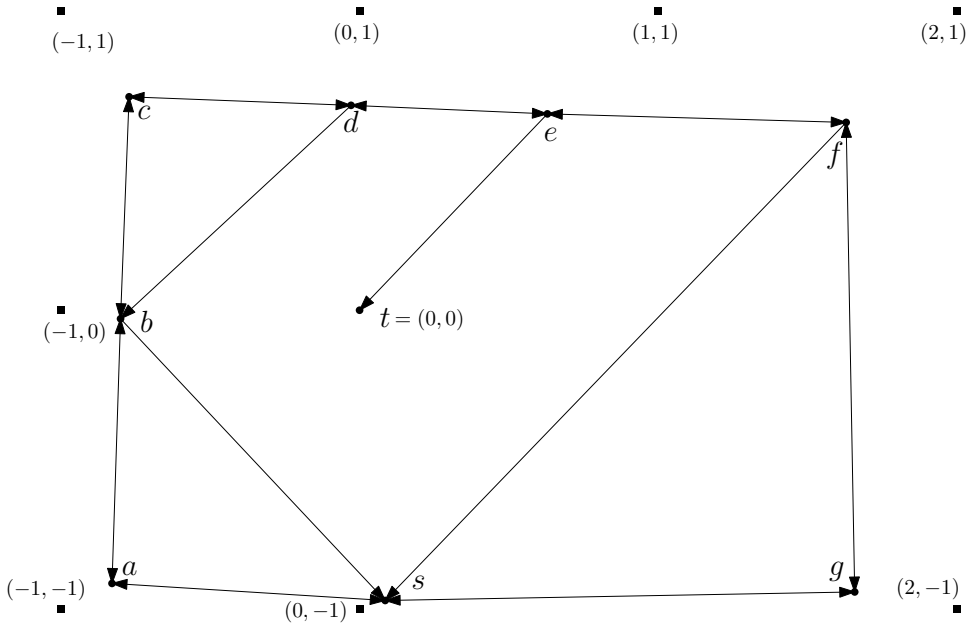


Figure 7 Pictured is the $\vec{\text{Yao}}_4^\infty$ graph with vertices $\{s, a, b, c, d, e, f, g, t\}$. The shortest path from s to t is (s, a, b, c, d, e, t) and its length can be made arbitrarily close to $5 + \sqrt{2}$. The outgoing edges from t are omitted.

Next, we have $\|et\|_2 \geq \sqrt{2}\min(\|et\|_x, \|et\|_y) = \sqrt{2}(1 - 13\delta)$, therefore the shortest path from s to t in G has length at least $5 + \sqrt{2} - \delta(37 + 13\sqrt{2})$. Finally, $\|st\|_2 \leq \sqrt{(1-\delta)^2 + (3\delta)^2} \leq 1$ when $\delta \leq \frac{1}{5}$. Therefore choosing $\delta < \frac{\epsilon}{37+13\sqrt{2}}$ means that the spanning ratio of G is at least $5 + \sqrt{2} - \epsilon$. ◀

▶ **Lemma 20.** *Let $\epsilon > 0$. Then any local routing algorithm for $\vec{Yao}_4^\infty$ graphs has routing ratio at least $7 + \sqrt{2} - \epsilon$ in the worst case.*

Proof. Let $\epsilon > 0$ be arbitrarily small and let A be a local routing algorithm. We will construct three graphs G_W, G_N, G_E such that A has a routing ratio of at least $7 + \sqrt{2} - \epsilon$ in at least one of $\{G_W, G_N, G_E\}$ when routing from s to $t := (0, 0)$. The three graphs are shown in Figure 9. Let $\delta > 0$ be arbitrarily small.

We will start by defining the *zip-zag paths*, illustrated in Figure 8, which will act as subgraphs. Define the two segments $s_1 := (15\delta - 1, \delta)(0, \delta - \delta^2)$ and $s_2 := (15\delta - 1, -\delta)(0, \delta^2 - \delta)$. Let $p_1 := (15\delta - 1, \delta)$, and $p_2 \in s_1$ such that $x(p_2) = 13\delta - 1$. Let $q_1 \in s_2$ such that p_2q_1 has slope 1. Then for $i \geq 3$ define $p_i \in s_1$ and $q_{i-1} \in s_2$ such that triangle $p_{i-1}p_iq_{i-1}$ is similar to triangle $p_{i-2}p_{i-1}q_{i-2}$. Let k be the greatest index such that $x(p_k) < 0$. Then define WestZag $:= \{p_1, p_2, ..., p_k, q_1, q_2, ..., q_{k-1}, t\}$.

The shortest path from p_1 to t in the $\vec{\text{Yao}}_4^\infty$ graph of WestZag is $p_1, q_1, p_2, q_2, ..., q_{k-1}, p_k, t$, and it is x-monotone. We will now express the length of this path in terms of δ. Firstly, we have $\|p_1q_1\|_2 \geq \|p_1q_1\|_y \geq 2(\delta - \delta^2)$ and $\|q_1p_2\|_2 = \sqrt{2}\|q_1p_2\|_y \geq 2\sqrt{2}(\delta - \delta^2)$. By similar triangles, for all $1 < i < k$ we have

$$\frac{\|p_1q_1\|_2 + \|q_1p_2\|_2}{x(p_2) - x(p_1)} = \frac{\|p_iq_i\|_2 + \|q_ip_{i+1}\|_2}{x(p_{i+1}) - x(p_i)}.$$

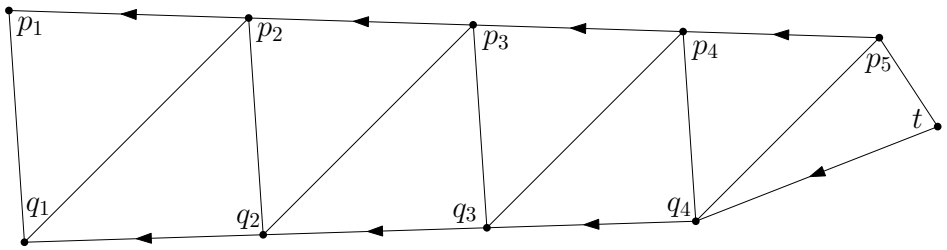


Figure 8 The shortest path from p_1 to t in the $\vec{\text{Yao}}_4^\infty$ graph of WestZag has length arbitrarily close to $1+\sqrt{2}$. Note, all $p_1, p_2, ..., p_k$ lie on segment s_1, and all $q_1, q_2, ..., q_{k-1}$ lie on segment s_2.

Combining this with the observation that $-2\delta < x(p_k)$, then the shortest path from p_1 to t in WestZag has length at least

$$\begin{aligned}
\sum_{i=1}^{k-1}(\|p_iq_i\|_2 + \|q_ip_{i+1}\|_2) &= \frac{\|p_1q_1\|_2 + \|q_1p_2\|_2}{x(p_2)-x(p_1)} \sum_{i=1}^{k-1}(x(p_{i+1}) - x(p_i)) \\
&\geq \frac{2(\delta-\delta^2)(1+\sqrt{2})}{2\delta}(x(p_k) - x(p_1)) \\
&\geq \frac{2(\delta-\delta^2)(1+\sqrt{2})}{2\delta}(-2\delta - (15\delta - 1)) \\
&= (1+\sqrt{2})(1-\delta)(1-17\delta) \\
&\geq 1+\sqrt{2}-44\delta
\end{aligned}$$

Additionally, the shortest path from q_1 to t in WestZag has length at least $1+\sqrt{2}-47\delta$ since $\|p_1q_1\|_2 \leq \sqrt{2}\|p_1q_1\|_\infty \leq 3\delta$. Analogously, we define NorthZag to be WestZag reflected about the line $y=-x$. Similarly, define EastZag to be WestZag reflected about the vertical line through t.

Now, define the points $s := (0,-1)+\delta(1,1)$, $a := (-1,-1)+\delta(5,2)$, $b := (-1,0)+\delta(6,-2)$, $c := (-1,1)+\delta(7,-10)$, $d := (0,1)+\delta(-1,-11)$, $e := (1,1)+\delta(-4,-6)$, $f := (1,0)+\delta(-3,-1)$, $g := (1,-1)+\delta(-2,3)$, $c' := (-1,1)+\delta(10,-8)$, $d' := (0,1)+\delta(1,-7)$, $e' := (1,1)+\delta(-14,-12)$. Note that $\|st\|_2 = \sqrt{(1-\delta)^2+\delta^2} \leq 1$ since $0<\delta<1$. We define vertex sets $V(G_W) := \text{WestZag} \cup \{s,a,b,c',d',e,f,g\}$, $V(G_N) := \text{NorthZag} \cup \{s,a,b,c,d',e,f,g\}$, $V(G_E) := \text{EastZag} \cup \{s,a,b,c,d,e',f,g\}$.

This construction guarantees that the shortest path from any vertex not in a *zip-zag path* to t must pass through a *zip-zag path*. The decision tree for which graph to provide to A is illustrated in Figure 10. One can verify that the decision tree represents a valid adversarial strategy since the collections of graphs at each node contain identical neighbourhoods at all previously visited vertices. Finally, if we exclude the zig-zag paths, then each vertex is within a distance of 20δ from its nearest integer coordinate. Since the shortest path to t in each case uses at least six edges with length at least $1-2(20\delta)$ before passing through q_1, then the total path length in any case is at least $6(1-40\delta)+1+\sqrt{2}-47\delta = 7+\sqrt{2}-287\delta$. Setting $\delta < \frac{\epsilon}{287}$ completes the proof. ◀

▶ **Lemma 21.** *Let $\epsilon > 0$. There exists a $\vec{\text{Yao}}_4$ graph with spanning ratio at least $7-\sqrt{3}+\sqrt{5-2\sqrt{3}}-\epsilon$.*

Proof. Let $\epsilon > 0$ be arbitrarily small. We will create a graph with spanning ratio at least $7-\sqrt{3}+\sqrt{5-2\sqrt{3}}-\epsilon$, shown in Figure 11.

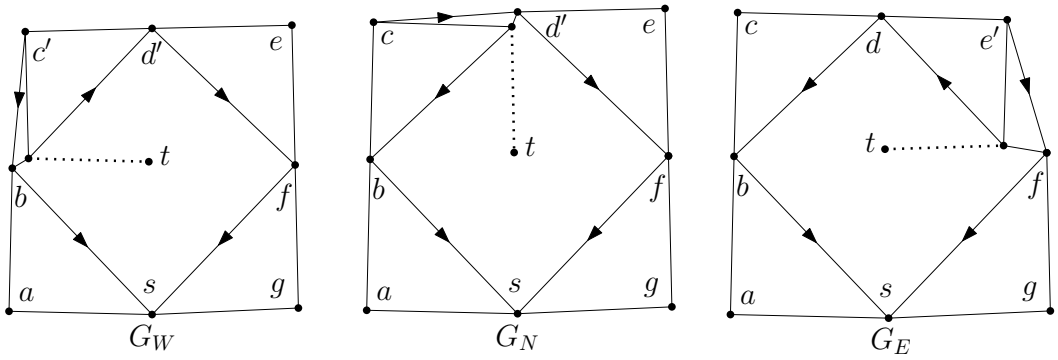


Figure 9 The graphs are G_W, G_N, G_E from left to right. When routing from s to t, the routing ratio of A is at least $7 + \sqrt{2} - \epsilon$ in one of the graphs. The dotted lines represent the zig-zag paths to t. All outgoing edges from the zig-zag paths are omitted.

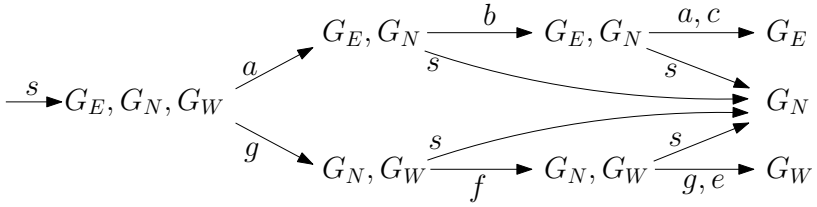


Figure 10 The decision tree describing the adversarial strategy for which graph to provide to A. For example, suppose A first visits the vertices s, a, b. Notice that the neighbourhoods around vertices s, a, b are all identical between graphs G_E, G_N. Then if A chooses c for the next vertex, then A would have a routing ratio of at least $7 + \sqrt{2} - \epsilon$ in G_E.

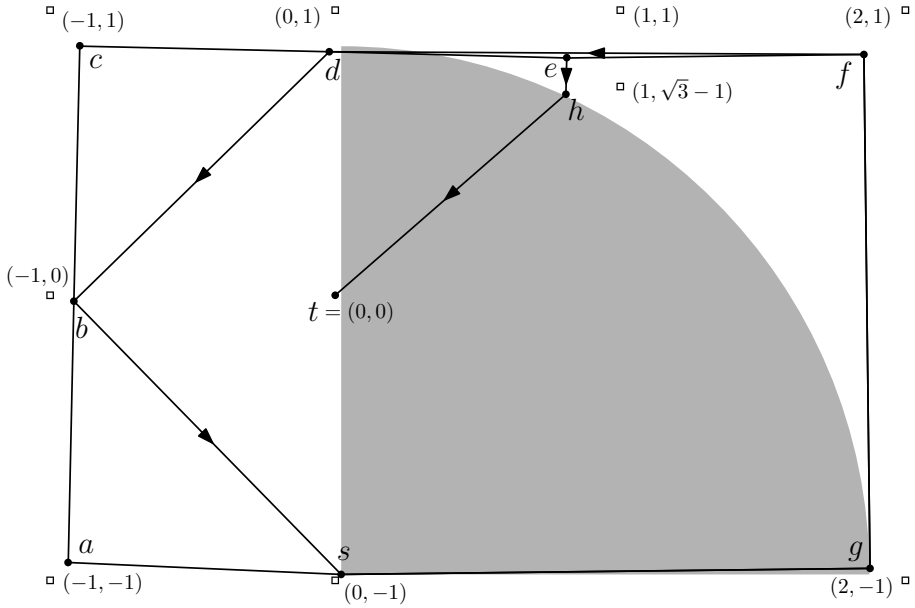


Figure 11 Pictured is the $\vec{\text{Yao}}_4$ graph with vertices $\{s, a, b, c, d, e, f, g, h, t\}$. The shortest path from s to t must pass by h, and its length can be made arbitrarily close to $7 - \sqrt{3} + \sqrt{5 - 2\sqrt{3}} \approx 6.51$. Several edges are omitted for clarity.

Let $\delta > 0$ be arbitrarily small. Define $s := (0,-1) + \delta(1,1)$, $a := (-1,-1) + \delta(3,3)$, $b := (-1,0) + \delta(4,-1)$, $c := (-1,1) + \delta(5,-6)$, $d := (0,1) + \delta(-1,-7)$, $e := (1,1) + \delta(-9,-8)$, $f := (2,1) + \delta(-7,-7.5)$, $g := (2,-1) + \delta(-6,2)$, and $t = (0,0)$.

Next, we define a point in the region below and to the left of e. More precisely, start by defining p_1 to be the north-most point satisfying $x(p_1) = x(e)$ and $\|sp_1\|_2 = \|sg\|_2$. Next, let p_2 be the east-most point such that $\|dp_2\|_2 = \|de\|_2$ and p_1p_2 is orthogonal to sp_1. Finally let $p_3 := (x(p_1), y(p_2))$. Define the convex combination $h := (1-2\delta)p_1 + \delta p_2 + \delta p_3$. Note that this construction guarantees that h is in the triangle $p_1p_2p_3$. Let G be the $\vec{\text{Yao}}_4$ graph with vertices $\{s, a, b, c, d, e, f, g, h, t\}$. By construction, ignoring outgoing edges from t, the edges of G are $sa, sg, as, ab, bs, ba, bc, cb, cd, db, dc, de, ed, eh, ef, eg, fe, fd, fg, gs, gf, ht, hd, he, hg$. Note that the only incoming edge to t is ht, and that the only incoming edge to h is eh. Also, for any distinct $u, v \in \{s, a, b, c, d, e, f, g\}$, the coordinates of u and v differ by at most 9δ from their nearest integer grid points, so the edge uv has length at least $1 - 18\delta$. Furthermore, sg and gf both have length at least $2 - 18\delta$. It remains to bound the lengths $\|eh\|_2$ and $\|ht\|_2$. We have

$$\begin{aligned}
\|sp_1\|_2 = \|sg\|_2 &\implies (1-10\delta)^2 + (y(p_1) - \delta + 1)^2 = (2-7\delta)^2 + \delta^2 \\
&\implies y(p_1) = \sqrt{3 + 6\delta - 50\delta^2} - 1 + \delta \\
&\implies y(p_1) \le \sqrt{3} - 1 + 4\sqrt{\delta} \text{ and } y(p_1) \ge \sqrt{3} - 1 - 8\delta \\
&\implies |y(p_1) - (\sqrt{3} - 1)| \le 8\sqrt{\delta}
\end{aligned}$$

Next, the $x-$ and $y-$ coordinates of p_1, p_2 and p_3 are in $[0,1]$, so by definition, we have

$$\|h - p_1\|_\infty = \| -2\delta y(p_1) + \delta y(p_2) + \delta y(p_3)\|_\infty \le 2\delta$$

This allows us to bound the distance from h to $(1, \sqrt{3}-1)$:

$$\|h - (1,\sqrt{3}-1)\|_\infty \le \|h - p_1\|_\infty + \|p_1 - (1,\sqrt{3}-1)\|_\infty \le 2\delta + \max(9\delta, 8\sqrt{\delta}) \le 11\sqrt{\delta}$$

Now we can lower bound the edge length $\|ht\|_2$ and $\|eh\|_\infty$ as follows:

$$\begin{aligned}
\|ht\|_2 &\ge \|(1,\sqrt{3}-1)t\|_2 - \|h(1,\sqrt{3}-1)\|_2 \ge \sqrt{5 - 2\sqrt{3}} - 11\sqrt{2\delta} \\
\|eh\|_\infty &\ge \|(1,1) - (1,\sqrt{3}-1)\|_\infty - \|e - (1,1)\|_\infty - \|h - (1,\sqrt{3}-1)\|_\infty \ge 2 - \sqrt{3} - 9\delta - 11\sqrt{\delta}
\end{aligned}$$

The shortest path from s to t in G will follow either path $P_1 := (s, a, b, c, d, e)$ or $P_2 = (s, g, f, e)$, followed by the edges eh and ht. We have

$$\begin{aligned}
d_G(s,t) &\ge \min(\|P_1\|_2, \|P_1\|_2) + \|eh\|_2 + \|ht\|_2 \\
&\ge 5(1-18\delta) + (2 - \sqrt{3} - 9\delta - 11\sqrt{\delta}) + (\sqrt{5 - 2\sqrt{3}} - 11\sqrt{2\delta}) \\
&\ge 7 - \sqrt{3} + \sqrt{5 - 2\sqrt{3}} - 126\sqrt{\delta}.
\end{aligned}$$

Setting $\delta < (\epsilon/126)^2$ completes the proof since $\|st\|_2 = \sqrt{(1-\delta)^2 + \delta^2} \le 1$. ◀

▶ **Lemma 22.** *Let $\epsilon > 0$. Then any local routing algorithm for $\vec{\text{Yao}}_4$ graphs has routing ratio at least $7 + \sqrt{2} - \epsilon$ in the worst case.*

Proof. Let $\epsilon > 0$ be arbitrarily small. We will construct five $\vec{\text{Yao}}_4$ graphs G_1, G_2, G_3, G_4, G_5 such that any local routing algorithm has a routing ratio of at least $7 + \sqrt{2} - \epsilon$ in at least one of the five graphs. The five graphs are shown in Figure 12. Let $\delta < 0$ be arbitrarily small. Define

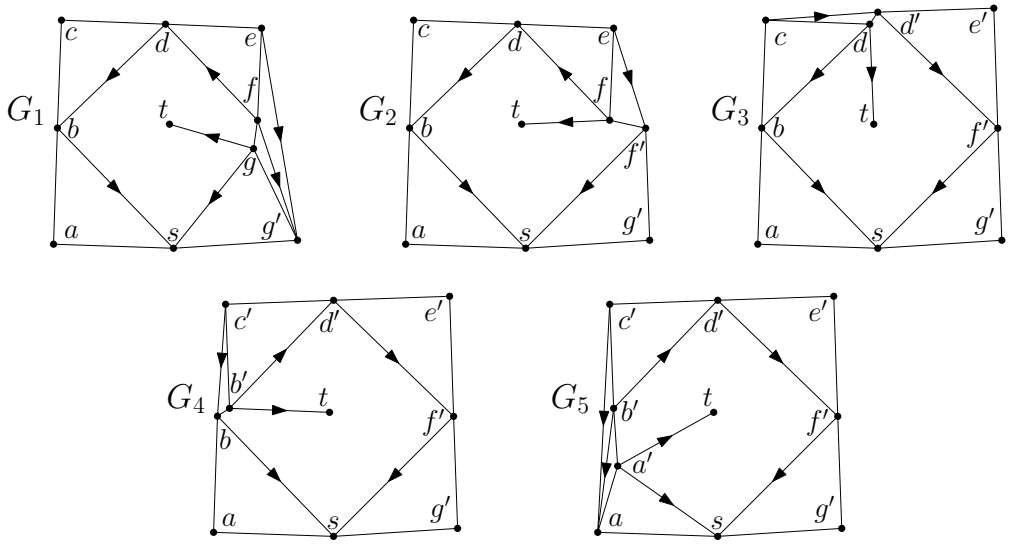


Figure 12 Any local routing algorithm for $\vec{\text{Yao}}_4$ has a routing ratio of at least $7 + \sqrt{2} - \epsilon$ in one of the five graphs. The outgoing edges from t are omitted.

the points $t := (0, 0)$, $s := (0, -1) + \delta(1, 1)$, $a := (-1, -1) + \delta(3, 2)$, $b := (-1, 0) + \delta(4, -1)$, $c := (-1, 1) + \delta(5, -6)$, $d := (0, 1) + \delta(-1, -7)$, $e := (1, 1) + \delta(-9, -8)$, $f := (1, 0) + \delta(-10, 1)$, $g := (1 - 11\delta, \sqrt{22\delta} + 2\delta - 1)$, $a' := (8\delta - 1, \sqrt{10\delta} + 2\delta - 1)$, $b' := (-1, 0) + \delta(7, 1)$, $c' := (-1, 1) + \delta(6, -5)$, $d' := (0, 1) + \delta(1, -4)$, $e' := (1, 1) + \delta(-2, -3)$, $f' := (1, 0) + \delta(-1, -1)$, $g' := (1, -1) + \delta(0, 3)$. We have $\|st\|_2 = \sqrt{(1 - \delta)^2 + \delta^2} \leq 1$ since $0 < \delta < 1$. Then define the graphs by their point sets: $V(G_1) := \{s, t, a, b, c, d, e, f, g, g'\}$, $V(G_2) := \{s, t, a, b, c, d, e, f, f', g'\}$, $V(G_3) := \{s, t, a, b, c, d, d', e', f', g'\}$, $V(G_4) := \{s, t, a, b, b', c', d', e', f', g'\}$, $V(G_5) := \{s, t, a, a', b', c', d', e', f', g'\}$. Notice that the only incoming edges to t in the five graphs are $gt, ft, dt, b't, a't$, respectively. The decision tree for which graph to provide to any algorithm is illustrated in Figure 13.

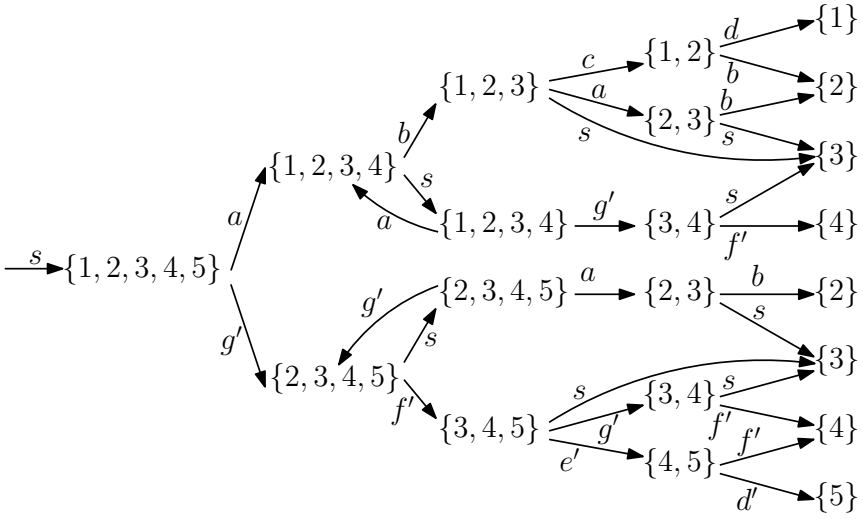


Figure 13 The adversarial decision tree for which graph to provide to any algorithm.

By construction, each coordinate of each vertex differs by at most $11\sqrt{\delta}$ from its nearest integer grid point. This implies that any edge connecting two different letters must have length of at least $1-22\sqrt{\delta}$. Moreover, eg and $c'a$ have length at least $2-22\sqrt{\delta}$. Furthermore, the nearly diagonal edges, $bs, f's, a't, gt$, have length at least $\sqrt{2}(1-22\sqrt{\delta})$. In any case, we append the shortest path to t from a leaf of the decision tree to get a routing ratio of at least $7(1-22\sqrt{\delta})+\sqrt{2}(1-22\sqrt{\delta}) \geq 7+\sqrt{2}-186\sqrt{\delta}$ in the worst case. Setting $\delta < (\epsilon/186)^2$ guarantees that the routing ratio is at least $7+\sqrt{2}-\epsilon$. ◀

References

1 Hugo A. Akitaya, Ahmad Biniaz, and Prosenjit Bose. On the spanning and routing ratios of the directed Θ_6-graph. *Comput. Geom.*, 105-106:101881, 2022. doi:10.1016/j.comgeo.2022.101881.

2 Vikrant Ashvinkumar, Joachim Gudmundsson, Christos Levcopoulos, Bengt J. Nilsson, and André van Renssen. Local routing in sparse and lightweight geometric graphs. *Algorithmica*, 84(5):1316–1340, 2022. doi:10.1007/s00453-022-00930-2.

3 Luis Barba, Prosenjit Bose, Mirela Damian, Rolf Fagerberg, Wah Loon Keng, Joseph O'Rourke, André van Renssen, Perouz Taslakian, Sander Verdonschot, and Ge Xia. New and improved spanning ratios for Yao graphs. *J. Comput. Geom.*, 6(2):19–53, 2015. doi:10.20382/jocg.v6i2a3.

4 Nicolas Bonichon, Cyril Gavoille, Nicolas Hanusse, and Ljubomir Perkovic. Plane spanners of maximum degree six. In *ICALP (1)*, volume 6198 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 2010. doi:10.1007/978-3-642-14165-2_3.

5 Nicolas Bonichon, Cyril Gavoille, Nicolas Hanusse, and Ljubomir Perkovic. Tight stretch factors for L_1- and L_∞-Delaunay triangulations. *Comput. Geom.*, 48(3):237–250, 2015. doi:10.1016/j.comgeo.2014.10.005.

6 Nicolas Bonichon, Iyad A. Kanj, Ljubomir Perkovic, and Ge Xia. There are plane spanners of degree 4 and moderate stretch factor. *Discret. Comput. Geom.*, 53(3):514–546, 2015. doi:10.1007/s00454-015-9676-z.

7 Prosenjit Bose, Jean-Lou De Carufel, Darryl Hill, and Michiel Smid. On the spanning and routing ratio of the directed theta-four graph. *Discret. Comput. Geom.*, 71(3):872–892, 2024. doi:10.1007/s00454-023-00597-8.

8 Prosenjit Bose, Jean-Lou De Carufel, Pat Morin, André van Renssen, and Sander Verdonschot. Towards tight bounds on theta-graphs: More is not always better. *Theor. Comput. Sci.*, 616:70–93, 2016. doi:10.1016/j.tcs.2015.12.017.

9 Prosenjit Bose, Mirela Damian, Karim Douïeb, Joseph O'Rourke, Ben Seamone, Michiel H. M. Smid, and Stefanie Wuhrer. $\pi/2$-angle Yao graphs are spanners. *Int. J. Comput. Geom. Appl.*, 22(1):61–82, 2012. doi:10.1142/S0218195912600047.

10 Prosenjit Bose, Rolf Fagerberg, André van Renssen, and Sander Verdonschot. Optimal local routing on Delaunay triangulations defined by empty equilateral triangles. *SIAM J. Comput.*, 44(6):1626–1649, 2015. doi:10.1137/140988103.

11 Prosenjit Bose, Darryl Hill, Michiel Smid, and Tyler Tuttle. On the spanning and routing ratios of the Yao-four graph. In *ISAAC*, volume 322 of *LIPIcs*, pages 15:1–15:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ISAAC.2024.15.

12 Kevin Buchin, Joachim Gudmundsson, Antonia Kalb, Aleksandr Popov, Carolin Rehs, André van Renssen, and Sampson Wong. Oriented spanners. In *ESA*, volume 274 of *LIPIcs*, pages 26:1–26:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.26.

13 Paul Chew. There is a planar graph almost as good as the complete graph. In *SCG*, pages 169–177. ACM, 1986. doi:10.1145/10515.10534.

14 Mirela Damian and Naresh Nelavalli. Improved bounds on the stretch factor of y_4. *Comput. Geom.*, 62:14–24, 2017. doi:10.1016/j.comgeo.2016.12.001.

15 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. doi:10.1007/978-3-540-77974-2.

16 William Michael Zoltan Kalnay. Routing ratio of the directed Yao-6 graphs. Master's thesis, Carleton University, Ottawa, 2023.

17 Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007. doi:10.1017/CBO9780511546884.

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

On Geodesic Disks Enclosing Many Points*

Prosenjit Bose
School of Computer Science, Carleton University, Ottawa, Canada

Guillermo Esteban
Departamento de Física y Matemáticas, Universidad de Alcalá, Alcalá de Henares, Spain

David Orden
Departamento de Física y Matemáticas, Universidad de Alcalá, Alcalá de Henares, Spain

Rodrigo I. Silveira
Departament de Matemàtiques, Universitat Politècnica de Catalunya, Barcelona, Spain

Tyler Tuttle
School of Computer Science, Carleton University, Ottawa, Canada

Abstract

Let $\Pi(n)$ be the largest number such that for every set S of n points in a polygon P, there always exist two points $x, y \in S$, where every geodesic disk containing x and y contains $\Pi(n)$ points of S. We establish upper and lower bounds for $\Pi(n)$, and show that $\lceil \frac{n}{5} \rceil + 1 \leq \Pi(n) \leq \lceil \frac{n}{4} \rceil + 1$. We also show that there always exist two points $x, y \in S$ such that every geodesic disk with x and y on its boundary contains at least $\frac{16}{665}(n-2) \approx \lceil \frac{n-2}{41.6} \rceil$ points both inside and outside the disk. For the special case where the points of S are restricted to be the vertices of a geodesically convex polygon we give a tight bound of $\lceil \frac{n}{3} \rceil + 1$. We provide the same tight bound when we only consider geodesic disks having x and y as diametral endpoints. Finally, we give a lower bound of $\lceil \frac{n-2}{36} \rceil + 2$ for the two-colored version of the problem.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Enclosing disks, Geodesic disks, Bichromatic

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.10

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2506.06477 [11]

Funding G. E., D. O. and R. S. are partially supported by grants PID2019-104129GB-I00 and PID2023-150725NB-I00 funded by MICIU/AEI/10.13039/501100011033. P. B. and T. T. are supported in part by NSERC.

1 Introduction

Given a set S of n points in the plane in general position – no three of them are collinear and no four of them are cocircular – there always exist two points $u, v \in S$ such that any disk that contains u and v also contains a constant fraction of S. Neumann-Lara and Urrutia [19] proved that this constant fraction is at least $\lceil \frac{n-2}{60} \rceil$. This bound was improved in a series of papers [14, 15] culminating in the best known lower bound of about $\frac{n}{4.7}$ by Edelsbrunner et al. [13]. Hayward et al. [15] gave an upper bound by constructing a set of n points such that for every pair of points u, v, there exists a disk with u, v on its boundary containing less than $\lceil \frac{n}{4} \rceil$ points, thereby showing that $\lceil \frac{n}{4} \rceil + 1$ is an upper bound for the problem. In addition, they studied the problem for point sets in convex position, giving a tight bound of $\lceil \frac{n}{3} \rceil + 1$.

* Since the acceptance of this paper by the WADS Program Committee, the authors have improved some bounds that can be found in the full version [11].

© Prosenjit Bose, Guillermo Esteban, David Orden, Rodrigo I. Silveira, and Tyler Tuttle;
licensed under Creative Commons License CC-BY 4.0

Table 1 Summary of our results, comparing the Euclidean and geodesic settings. Note that non-integer values in the table are approximations, and refer to the corresponding reference for the exact value.

Variant (see Def. 1)	Euclidean setting		Geodesic setting	
	Lower bound	Upper bound	Lower bound	Upper bound
$\Pi(n)$	$\lfloor \frac{n}{4.7} \rfloor$ [13]	$\lceil \frac{n}{4} \rceil + 1$ [15]	$\lceil \frac{n}{5} \rceil + 1$ (Thm. 10)	$\lceil \frac{n}{4} \rceil + 1$
$\overline{\Pi}(n)$	$\lceil \frac{n}{3} \rceil + 1$ [15]	$\lceil \frac{n}{3} \rceil + 1$ [15]	$\lceil \frac{n}{3} \rceil + 1$ (Thm. 14)	$\lceil \frac{n}{3} \rceil + 1$
$\Pi^{in\text{-}out}(n)$	$\lfloor \frac{n}{4.7} \rfloor$ [23]	$\lceil \frac{n}{4} \rceil + 1$ [15]	$\lceil \frac{n-2}{41.6} \rceil + 1$ (Thm. 12)	$\lceil \frac{n}{4} \rceil + 1$
$\Pi^{diam}(n)$	$\lceil \frac{n}{3} \rceil + 1$ [2]	$\lceil \frac{n}{3} \rceil + 1$ [2]	$\lceil \frac{n}{3} \rceil + 1$ (Thm. 16)	$\lceil \frac{n}{3} \rceil + 1$
$\Pi^{bichrom}(n)$	$\lfloor \frac{n}{6.8} \rfloor$ [12]	$\lceil \frac{n}{4} \rceil + 1$ [12]	$\lceil \frac{n-2}{36} \rceil + 2$ (Thm. 18)	$\lceil \frac{n}{4} \rceil + 1$

Another version of the problem is to consider disks in the plane having the points u and v as diametral endpoints [2]. In this case, a tight bound of $\lceil \frac{n}{3} \rceil + 1$ for both the convex and non-convex cases was shown.

The best known lower bound of $\left(\frac{1}{2} - \frac{1}{\sqrt{12}}\right) n + O(1) \approx \frac{n}{4.7}$ was obtained in multiple ways, see [12, 13, 23]. Edelsbrunner et al. [13] were the first to show this using techniques related to the order-k Voronoi diagram. Ramos and Viaña [23] used known results about j-facets of point sets in $\mathbb{R}^3$. In fact, these results allowed Ramos and Viaña to prove that there is always a pair of points such that any disk through them has at least $\frac{n}{4.7}$ points both inside and outside the disk.

Generalizations of the original problem to higher dimensions have been studied by Bárány et al. [6] who proved that any set $S \subset \mathbb{R}^d$ of n points in general position contains a subset A of size $m = \lfloor \frac{1}{2}(d+3) \rfloor$ such that, any ball with A on its boundary, contains at least $\frac{m!(d-m-1)!}{d!} n$ other points of S. Moreover, Bárány and Larman extended the results in [19] from Euclidean balls to ellipses in $\mathbb{R}^2$ and, more generally, to quadrics in $\mathbb{R}^d$ [5]. In particular, they showed that any set $S \subset \mathbb{R}^d$ of n points contains a subset A, of size $\lfloor \frac{1}{4} d(d+3) \rfloor + 1$, with the property that any quadric through A, contains at least $\frac{n}{s 2^{s+1}}$ points of S, where $s = \frac{(d+1)(d+2)}{2}$.

A colored version of the problem has also been studied in the literature: Given a set S of $\frac{n}{2}$ red and $\frac{n}{2}$ blue points in the plane, there always exists a bichromatic pair of points $u, v \in S$ such that any disk containing u and v contains at least a constant fraction of the n points. Prodromou [22] proved[1] that any set $S \subset \mathbb{R}^d$ colored with $k = \lfloor \frac{d+3}{2} \rfloor$ colors contains a subset $A \subset S$ of k points, one of each color, such that any ball through all points in A contains at least $\frac{|S|}{2k3^k}$ points of S, which gives a fraction $\frac{n}{36}$ for points in the plane. This fraction for the two-dimensional case was later improved in [12] to $\left(\frac{1}{2} - \frac{1}{\sqrt{8}}\right) n - o(n) \approx \frac{n}{6.8}$.

We note that all the upper bounds in the Euclidean setting translate directly to the geodesic setting by enclosing the constructions in a large enough triangle. Thus, in this paper, we focus mainly on the lower bounds and address all variants of the problem in the case where S is a set of n points inside a simple polygon P. In addition, throughout the paper, we consider that $n \geq 21$, so that all our results hold.

We summarize all our results and compare them with the Euclidean setting in Table 1. In Section 3, we present an overview of key properties of shortest paths in P (which we refer to as the *geodesic setting*), emphasizing both their similarities to and differences from

[1] Using techniques from Edelsbrunner et al. [13], a PhD thesis written in Spanish and not published elsewhere had claimed before a lower bound of $\frac{23n}{250} \approx \frac{n}{11}$ [25].

the Euclidean case. It is these differences that pose challenges we address to extend the results to the geodesic setting. Using these properties, in Section 4, we extend the lower bounds from the Euclidean case [14, 19] to the geodesic setting. While the upper bounds carry over directly, the lower bounds require new arguments due to the differences outlined in Section 3. For instance, unlike the Euclidean metric, which has bounded doubling dimension, the doubling dimension of the geodesic metric is not necessarily bounded since it depends on the number of reflex vertices of P. Additionally, we establish a tight bound for the case where the point set is in geodesically convex position. In Section 5, we show that the techniques in [2], where only diametral disks are considered, can be generalized to the geodesic setting. Finally, in Section 6 we extend the results in [22, 25] for two-colored points.

The problem of studying the largest number of points contained in any disk through two points inside a polygon is distinct from the planar case. As illustrated in Figure 1, a geodesic disk of radius r in a polygon P can be strictly contained in a Euclidean disk of radius r. As such, in this setting, it is unclear whether there always exists a pair of points such that any geodesic disk through them can contain the same number of points as in the Euclidean setting.

2 Preliminaries and notation

A sequence of points in the plane, $p_1, \ldots, p_n$, forms a *polygonal chain* where the edges of the chain are the segments $p_i p_{i+1}$, for $i \in \{1, \ldots, n-1\}$. A polygonal chain is *simple* if consecutive edges intersect at their common point and non-consecutive segments do not intersect. In a graph theoretic sense, this is a path. A simple polygon is a simple closed polygonal chain, i.e. a cycle. Informally, a polygon P is a *weakly-simple* polygon provided that a slight perturbation of the vertices of the polygon results in a simple polygon. See Akitaya et al. [1] for a formal definition of weakly-simple as well as an algorithm to quickly recognize weakly-simple polygons. Given a pair of points u, v in a polygon P, the *geodesic* (or *shortest*) *path* from u to v in P is denoted by $g(u, v)$. It is well-known that $g(u, v)$ is a polygonal chain. The length of this path is the sum of the lengths of its edges and is denoted by $|g(u, v)|$.

A subset P' of P is called *geodesically convex* if, for all pairs of points $u, v \in P'$, the geodesic path $g(u, v)$ is contained in P'. The *geodesic convex hull* of a set S of points in P is the geodesically convex set of minimum area that contains S. We say that S is in *geodesically convex position* if every point of S is on the boundary of the geodesic convex hull of S. A *geodesic triangle* on three points $a, b, c \in P$, denoted $\triangle(a, b, c)$, is a weakly-simple polygon whose boundary consists of $g(a, b)$, $g(b, c)$, and $g(c, a)$. The geodesic paths $g(a, b)$ and $g(a, c)$ follow a common route from a until they diverge at a point a' (which may be a itself). Similarly, let b' be the point where $g(b, a)$ and $g(b, c)$ diverge, and c' be the point where $g(c, a)$ and $g(c, b)$ diverge. The geodesic triangle $\triangle(a', b', c')$ is a simple polygon that has a', b' and c' as its convex vertices. The geodesic triangle $\triangle(a', b', c')$ is referred to as the *geodesic core* of $\triangle(a, b, c)$ with the vertices a', b', c' referred to as the *core* vertices corresponding to the vertices a, b, c, respectively. The geodesic core of $\triangle(a, b, c)$ is denoted as $\triangledown(a, b, c)$ [9, 21]. In addition, $\angle abc$ denotes the convex angle formed by the two edges of the geodesic core adjacent to the core vertex b' in $\triangledown(a, b, c)$. See Figure 2.

A *geodesic disk* centered at $c \in P$ with radius $r \geq 0$ is the set $D(c, r) = \{x \in P : |g(c, x)| \leq r\}$. In the plane, there always exists a disk through three given points, assuming they are not collinear. When we refer to a disk *through* a set of points, we mean that the points are on the boundary of the disk. In a simple polygon, it is no longer true that there always exists a geodesic disk through three given points. This essentially means that given

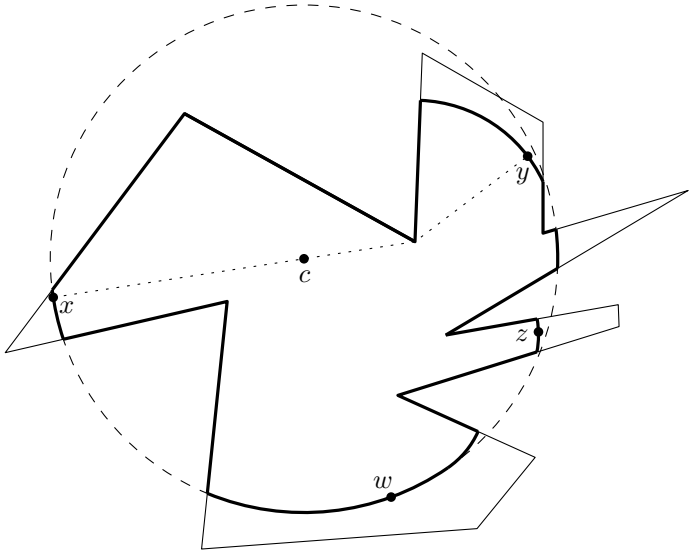


Figure 1 Geodesic disk centered at c, with radius $|g(c,z)|$, shown solid, and the equivalent Euclidean disk, shown dashed.

three points in a simple polygon, there does not always exists a point in the polygon that is geodesically equidistant to all three. However, if there is a geodesic disk through three points in a polygon, then it is unique [4]. Given two points u and v, there always exists a geodesic disk through u and v centered at c_{uv}, the midpoint of $g(u,v)$, called the *diametral geodesic disk*.

A set S' of at least three points in P is *geodesically collinear* if $\exists x, y \in S'$ such that $S' \subset g(x,y)$, see set $\{x,c,y\}$ in Figure 1. Similarly, a set S'' of at least four points is *geodesically cocircular* if there exists a geodesic disk with S'' on its boundary, see set $\{w,x,y,z\}$ in Figure 1. A set S of n points in a polygon P is said to be in *general position* if no three points in S are geodesically collinear and no four points in S or the boundary of P are geodesically cocircular. In this paper we only consider sets S in general position in polygons P with the extra condition that there are no two points $x, y \in S$ and a reflex vertex p on the boundary of P such that $y \subset g(x,p)$.

Given two points u and v, the *extension path* $\ell(u,v)$ is defined as follows: extend the first and last segment of $g(u,v)$ until they properly intersect with P. Denote these extension points on the boundary as $\hat{u}$ and $\hat{v}$, respectively. See the red path in Figure 2. Note that if u (or v) is on the boundary of P then $\hat{u} = u$ (or $\hat{v} = v$). Also note that, because of the extra condition added to the general position, the extensions cannot touch a reflex vertex p on the boundary of the polygon P. The extension path $\ell(u,v)$ naturally partitions the polygon into two weakly-simple polygons. We say that a point w is to the left (resp., right) of $g(u,v)$ if w belongs to the weakly-simple polygon to the left (resp., right) of $\ell(u,v)$, when considering $\ell(u,v)$ oriented from u to v. In addition, we define the left (resp., right) half-polygon of the path $g(u,v)$ as the set of points that are to the left (resp., right) of its extension path $\ell(u,v)$ and denote it as $P^+(u,v)$ (resp., $P^-(u,v)$).

Given two points $u, v \in S$, the bisector $b(u,v)$ is the set of all points that are equidistant to u and v. In this paper, as in [3], we assume that no bisector $b(u,v)$ passes through a vertex of P. Hence, $b(u,v)$ is a continuous curve connecting two points on the boundary of P. Moreover, $b(u,v)$ can be decomposed into $O(|P|)$ pieces [3], each of which is a subarc of a hyperbola (that could degenerate to a line segment).

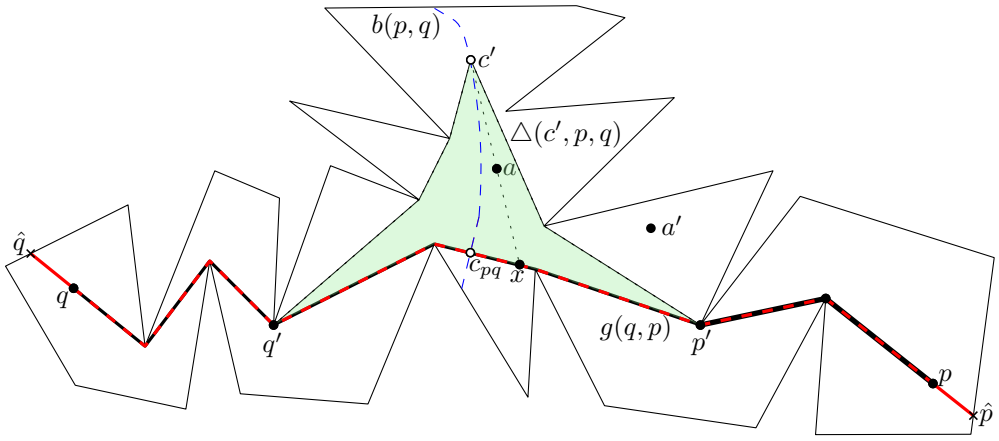


Figure 2 The point a belongs to the interior of $\triangle(c', p, q)$.

▶ **Definition 1.** *Let S be a set of n points in a polygon P.*

- $\Pi(n)$ *(resp., $\overline{\Pi}(n)$) is the largest integer such that, for every set (resp., geodesically convex set) S, there exist points $u, v \in S$ such that every geodesic disk through them contains $\Pi(n)$ (resp., $\overline{\Pi}(n)$) points of S.*
- $\Pi^{in\text{-}out}(n)$ *is the largest integer such that, for every set S, there exist points $u, v \in S$ such that every geodesic disk through them has $\Pi^{in\text{-}out}(n)$ points of S, both inside and outside.*
- $\Pi^{diam}(n)$ *(resp., $\overline{\Pi}^{diam}(n)$) is the largest integer such that, for every set (resp., geodesically convex set) S, there exist points $u, v \in S$ such that $D\left(c_{uv}, \frac{|g(u,v)|}{2}\right)$ contains $\Pi^{diam}(n)$ (resp., $\overline{\Pi}^{diam}(n)$) points of S.*

Let B be a set of $\frac{n}{2}$ blue points, and R be a set of $\frac{n}{2}$ red points in a polygon P, and let $S = B \cup R$.

- $\Pi^{bichrom}(n)$ *is the largest integer such that, for every set S, there exist points $u \in B$, $v \in R$ such that every geodesic disk through them contains $\Pi^{bichrom}(n)$ points of S.*

3 Euclidean and geodesic disks: some similarities and differences

In this section we describe some of the similarities and differences concerning Euclidean and geodesic disks that arise when proving the main results in this paper.

3.1 Some similarities

Two bounded curves g_1 and g_2 *intersect* if by traversing g_1 from one of its endpoints to the other endpoint, it crosses g_2 and switches from one side of g_2 to the other side [7, 24]. We say that g_1 and g_2 are *non-intersecting* if they do not intersect. Two non-intersecting curves can share an endpoint or can "touch" each other. Note that if g_1 and g_2 are geodesics they can intersect at most once [7].

It is well known that given two segments ab and pq that intersect transversely in the plane, any disk through a and b contains at least one endpoint of pq, or any disk through p and q contains at least one endpoint of ab, see, e.g., [19]. This result translates to the geodesic setting as shown below. We begin with a property of geodesic disks.

▶ **Lemma 2.** *Let p, q be two points in P and D, D' be two geodesic disks through p and q, respectively with centers c and c'. If c' is left of $\ell(p, c)$ then $D \cap P^+(p, q) \subset D'$.*

Proof. Let a be an arbitrary point in $D \cap P^+(p,q)$. By definition of $P^+(p,q)$, we have that a is to the left of $\ell(p,q)$. We consider two cases: $a \in \triangle(c',p,q)$ and $a \notin \triangle(c',p,q)$.

If a is in $\triangle(c',p,q)$, see Figure 2, then we use the fact that, for every point $x \in g(p,q)$, the length of $g(c',x)$ as x moves from p to q is a convex function with the maximum occurring at p or q [21]. In particular, we have that $|g(c',x)| \leq |g(c',q)| = |g(c',p)|$, for $x \in g(p,q)$. If we move x from p to q along $g(p,q)$, at some point $g(c',x)$ will go through the point a, so $|g(c',a)| \leq |g(c',x)|$. Hence, $|g(c',a)| \leq |g(c',q)| = |g(c',p)|$, which means that $a \in D'$.

Now consider the case where the point a is outside $\triangle(c',p,q)$, like a' in Figure 2. Without loss of generality, assume a is to the right of $g(p,c')$. (Otherwise, we can consider a and q, and the proof is analogous). In this case, $g(c',p)$ must intersect $g(c,a)$. Let x be a point in this intersection. By definition, we have $|g(c',x)| + |g(x,p)| = |g(c',p)|$. Since $a \in D$, we have that $|g(c,a)| = |g(c,x)| + |g(x,a)| \leq |g(c,p)|$ and by the triangle inequality, we have $|g(c,p)| \leq |g(c,x)| + |g(x,p)|$. This means that $|g(x,a)| \leq |g(x,p)|$. Therefore, $|g(c',a)| \leq |g(c',x)| + |g(x,a)| \leq |g(c',x)| + |g(x,p)| = |g(c',p)|$, implying $a \in D'$ in this case. ◀

With this property of geodesic disks in hand, we are now ready to prove the corresponding property of intersecting segments in the geodesic setting.

▶ **Lemma 3.** *Let a,b,p,q be four distinct points in general position in P such that $g(p,q)$ and $g(a,b)$ intersect. Then every geodesic disk with p and q on its boundary contains at least one endpoint of $g(a,b)$, or every geodesic disk with a,b on its boundary contains at least one endpoint of $g(p,q)$.*

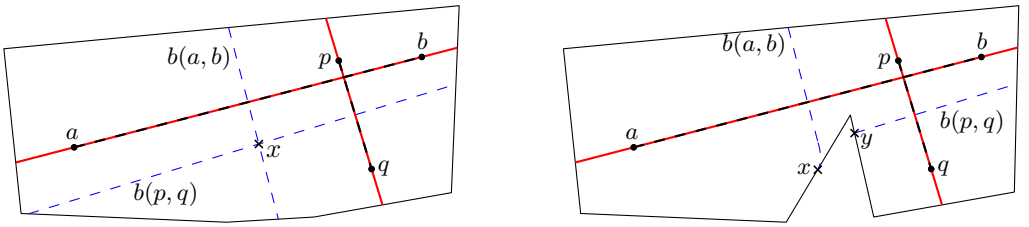
Proof. First, suppose that the bisectors $b(a,b)$ and $b(p,q)$ intersect, see Figure 3a, and let x be a point in their intersection. Then we have that $|g(x,a)| = |g(x,b)|$ and $|g(x,p)| = |g(x,q)|$. We can assume, without loss of generality, that $|g(x,p)| < |g(x,a)|$ since the points in S are in general position, and a,b,p,q are not co-circular. The case where $|g(x,p)| > |g(x,a)|$ is symmetric.

When $|g(x,p)| < |g(x,a)|$, the geodesic disk through a and b with center x contains p and q. Since $g(a,b)$ and $g(p,q)$ intersect, p and q are on different sides of the extension path $\ell(a,b)$. Hence, by Lemma 2, if we move the center of this disk along the bisector $b(a,b)$ in one direction while keeping a,b on its boundary, the geodesic disk will always contain p in its interior, and if we move the center in the other direction, the geodesic disk will always contain q.

Now, suppose that the bisectors $b(a,b)$ and $b(p,q)$ do not intersect, see Figure 3b. Hence, all points of the bisector $b(a,b)$ are closer to one endpoint of $g(p,q)$ than to the other one. Assume, without loss of generality, that this point is p, i.e., $|g(x',p)| \leq |g(x',q)|$, $\forall x' \in b(a,b)$. Similarly, all points of the bisector $b(p,q)$ are closer to one endpoint of $g(a,b)$ than to the other one. Assume, without loss of generality, that this point is b.

The bisector $b(a,b)$ intersects the boundary of the polygon. Also, the extension path $\ell(a,b)$ divides the polygon P into two half-polygons, one containing p and the other containing q. Let x be the intersection of $b(a,b)$ with the polygon boundary that is contained in the same half-polygon as q. Similarly, the extension path $\ell(p,q)$ divides the polygon into two half-polygons, and let y be the intersection of $b(p,q)$ with the polygon contained in the same half-polygon as a.

By definition, we have that $|g(x,a)| = |g(x,b)|$, and $|g(x,p)| < |g(x,q)|$. If $|g(x,p)| \leq |g(x,b)|$, the geodesic disk with center at x and radius $|g(x,b)|$ contains a,b and p. By Lemma 2, this means that every geodesic disk through a,b contains p.



(a) Bisectors $b(a, b)$ and $b(p, q)$ intersect. **(b)** Bisectors $b(a, b)$ and $b(p, q)$ do not intersect.

Figure 3 Any geodesic disk through a and b contains at least one endpoint of $g(p, q)$.

The only remaining case to consider is if $|g(x, p)| > |g(x, b)|$. Since the bisectors do not intersect, we have that $g(x, b)$ intersects $g(y, p)$. Let z be a point in this intersection. By definition, we have $|g(x, b)| = |g(x, z)| + |g(z, b)|$ and $|g(y, p)| = |g(y, z)| + |g(z, p)|$. By the triangle inequality, we have $|g(x, p)| \leq |g(x, z)| + |g(z, p)|$, which with our assumption that $|g(x, p)| > |g(x, b)|$ implies that $|g(z, b)| < |g(z, p)|$. However, this means that $|g(y, b)| \leq |g(y, z)| + |g(z, b)| < |g(y, z)| + |g(z, p)| = |g(y, p)|$. Therefore, when $|g(x, p)| > |g(x, b)|$, we have $|g(y, b)| < |g(y, p)|$. By Lemma 2, this means that every geodesic disk through p, q contains b. ◀

We now prove a complementary lemma about points that always remain outside disks through endpoints of geodesic paths that intersect. The proof is similar to that of Lemma 3, and we include it here for completeness.

▶ **Lemma 4.** *Let a, b, p, q be four distinct points in general position in P such that $g(p, q)$ and $g(a, b)$ intersect. Then every geodesic disk with p and q on its boundary does* not *contain at least one endpoint of $g(a, b)$, or every geodesic disk with a, b on its boundary does* not *contain at least one endpoint of $g(p, q)$.*

Proof. Suppose that the bisectors $b(a, b)$ and $b(p, q)$ intersect, see Figure 3a, and let x be a point in their intersection. Then we have that $|g(x, a)| = |g(x, b)|$ and $|g(x, p)| = |g(x, q)|$. We can assume, without loss of generality, that $|g(x, a)| < |g(x, p)|$ since the points in S are in general position, and a, b, p, q are not co-circular. The case where $|g(x, a)| > |g(x, p)|$ is symmetric.

When $|g(x, a)| < |g(x, p)|$, the geodesic disk through a and b with center x contains neither p nor q. Since $g(a, b)$ and $g(p, q)$ intersect, p and q are on different sides of the extension path $\ell(a, b)$. Note that if we exchange the roles of p by q, and D by D' in Lemma 2, we have that $D' \cap P^-(p, q) \subset D$. Hence, by Lemma 2, if we move the center of this disk along the bisector $b(a, b)$ in one direction while keeping a, b on its boundary, the geodesic disk will never contain p, and if we move the center in the other direction, the geodesic disk will never contain q.

Now, suppose that the bisectors $b(a, b)$ and $b(p, q)$ do not intersect, see Figure 3b. Hence, all points of the bisector $b(a, b)$ are closer to one endpoint of $g(p, q)$ than to the other one. Assume, without loss of generality, that this point is p, i.e., $|g(x', p)| \leq |g(x', q)|$, $\forall x' \in b(a, b)$. Similarly, all points of the bisector $b(p, q)$ are closer to one endpoint of $g(a, b)$ than to the other one. Assume, without loss of generality, that this point is b.

The bisector $b(a, b)$ intersects the boundary of the polygon. Also, the extension path $\ell(a, b)$ divides the polygon P into two half-polygons, one containing p and the other containing q. Let x be the intersection of $b(a, b)$ with the polygon boundary that is contained in the

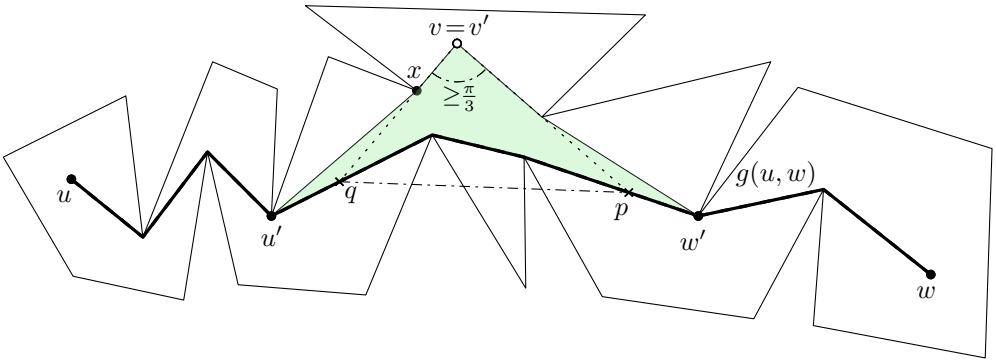


Figure 4 The angle $\angle uvw$ is greater than $\frac{\pi}{3}$, and $|vq| < |vp|$.

same half-polygon as q. Similarly, the extension path $\ell(p,q)$ divides the polygon into two half-polygons, and let y be the intersection of $b(p,q)$ with the polygon contained in the same half-polygon as a.

By definition, we have that $|g(x,a)| = |g(x,b)|$, and $|g(x,p)| < |g(x,q)|$. If $|g(x,b)| < |g(x,q)|$, the geodesic disk with center at x through a, b does not contain q. By Lemma 2, this means that every geodesic disk through a, b does not contain q.

The only remaining case to consider is when $|g(x,b)| \geq |g(x,q)|$. By the fact that a, b, p, q are not co-circular, we have that in fact $|g(x,b)| > |g(x,q)|$. Since the bisectors do not intersect, we have that $g(x,q)$ intersects $g(y,b)$. Let z be a point in this intersection. By definition, we have $|g(x,q)| = |g(x,z)| + |g(z,q)|$ and $|g(y,b)| = |g(y,z)| + |g(z,b)|$. By the triangle inequality, we have $|g(x,b)| \leq |g(x,z)| + |g(z,b)|$, which with our assumption that $|g(x,b)| > |g(x,q)|$ implies that $|g(z,b)| > |g(z,p)|$. However, this means that $|g(y,q)| < |g(y,z)| + |g(z,q)| \leq |g(y,z)| + |g(z,b)| = |g(y,b)|$. Therefore, when $|g(x,b)| \geq |g(x,q)|$, we have $|g(y,p)| < |g(y,b)|$. By Lemma 2, this means that every geodesic disk through p, q does not contain b. ◀

We now show two results about the length of the sides of a triangle. These results are trivial in the plane, but in the geodesic setting have not been proven before. In particular, they will be useful in Section 5 to prove the lower bounds on the diametral case.

▶ **Lemma 5.** *Let $\triangle(u,v,w)$ be a geodesic triangle, such that v is a convex vertex of $\nabla(u,v,w)$ and $\angle uvw \geq \frac{\pi}{3}$. Then $|g(u,w)| \geq \min\{|g(u,v)|, |g(v,w)|\}$.*

Proof. Consider the geodesic core $\nabla(u,v,w) = \triangle(u',v,w')$ of $\triangle(u,v,w)$. Recall that $\angle uvw$ is the angle at vertex v in $\nabla(u,v,w)$. Let q be the intersection of the geodesic $g(u',w')$ with the line containing the segment of $g(v,u)$ adjacent to v. Symmetrically, let p be the intersection of the geodesic $g(u',w')$ with the line containing the segment of $g(v,w)$ adjacent to v, see Figure 4.

Since $\triangle(v,p,q)$ is a Euclidean triangle, we know that $|pq| \geq \min\{|vp|, |vq|\}$. We can assume, without loss of generality, that $|pq| \geq |vq|$. Let x be the vertex that is the endpoint of the first edge of $g(v,u)$. By construction $|vq| = |vx| + |xq|$. Note that $|g(p,q)| \geq |pq| \geq |vq| = |vx| + |xq|$ (1). By the triangle inequality, $|g(x,u')| \leq |xq| + |g(q,u')|$. This implies that $|g(x,u')| - |xq| \leq |g(q,u')|$ (2). Putting it all together, we have

$$\begin{aligned} |g(v,u)| &= |g(u,u')| + |g(v,u')| \text{ by definition} \\ &= |g(u,u')| + |vx| + |g(x,u')| \text{ by definition} \\ &= |g(u,u')| + |vx| + |g(x,u')| - |xq| + |xq| \end{aligned}$$

$$\leq |g(u,u')| + |vx| + |g(q,u')| + |xq| \text{ by (2)}$$
$$\leq |g(q,u)| + |g(p,q)| \text{ by (1)}$$
$$\leq |g(p,u)|$$
$$\leq |g(w,u)|$$

We conclude that $|g(u,w)| \geq \min\{|g(u,v)|, |g(v,w)|\}$. ◀

▶ **Lemma 6.** *Let $\triangle(u,v,w)$ be a geodesic triangle, such that $\angle uvw \leq \frac{\pi}{3}$. Then we have that $|uw| \leq \max\{|g(u,v)|, |g(v,w)|\}$.*

Proof. Since the three points u, v, w form a Euclidean triangle, we know that $|uw| \leq \max\{|uv|, |vw|\}$. We can assume, without loss of generality, that $|uw| \leq |uv|$. Since $|uv| \leq |g(u,v)|$, we thus have that $|uw| \leq |g(u,v)| \leq \max\{|g(u,v)|, |g(v,w)|\}$. ◀

3.2 Some differences

In contrast to the similarities in the previous section, there are properties that cannot be generalized from the Euclidean to the geodesic setting. We highlight a few differences. One difference which we alluded to in Section 1 is that the geodesic metric, unlike the Euclidean metric, does not necessarily have bounded doubling dimension. Another difference is that the geodesic bisector of a pair of points inside a simple polygon can intersect a line segment more than once [10].

The Voronoi diagrams of order k were one of the tools used to prove the lower bounds in the Euclidean setting, see, e.g., [12, 13, 23]. However, some important properties of order-k Voronoi diagrams do not extend to the geodesic setting. For instance, in the Euclidean setting, the bisector of a pair of points is a line, whereas in the geodesic setting, the bisector is a finite curve. Moreover, this means that in the Euclidean case, three points in general position uniquely define a disk with those points on the boundary. On the other hand, given three points inside a polygon P, there may not exist a point in P equidistant to all three. Thus, there are cases where three points do not define a geodesic disk with the three points on the boundary [4].

Another difference between the two settings is that the well-known lifting transformation that maps a point $(x,y) \in \mathbb{R}^2$ to the point $(x,y,x^2+y^2) \in \mathbb{R}^3$ used in [23] to obtain the lower bound of $\frac{n}{4.7}$ is no longer applicable in the geodesic setting. The exact number of edges of the order-k Voronoi diagram is known in the Euclidean case [18]. The formula for this exact number of edges was fundamental to Edelsbrunner et al.'s lower bound proof [13]. However, only upper bounds are known in the geodesic case [8]. In fact, this is one of the reasons why the lower bounds we obtain in the geodesic setting are not as strong as in the Euclidean setting.

4 The general case

In this section we provide a generalization to the geodesic setting of lower bounds that are known for the original problem in the plane. In particular, we generalize results from [14,15,19]. We note that upper bounds for the Euclidean setting are in general also valid for the geodesic setting, since one can take the constructions provided in [2, 12, 14], and enclose the relevant set of points in a bounding polygon, big enough so that the geodesic disks through every three points coincide with the corresponding Euclidean disks.

We begin by adapting some definitions and results from [19] to our setting. The *intersection number* $I(S)$ of a set of points S in a polygon is defined as the number of different geodesic paths $g(u,v)$, $g(x,y)$ for $u,v,x,y \in S$ such that $g(u,v) \cap g(x,y) \neq \emptyset$. Then, the *intersection graph* $G(S) = (V(G(S)), E(G(S)))$ is defined with $V(G(S)) = \{g(u,v) | u,v \in S, u \neq v\}$, and two vertices $g(u,v)$, $g(x,y)$ being joined by an edge in $G(S)$ if $g(u,v) \cap g(x,y) \neq \emptyset$. Finally, the oriented graph $\overrightarrow{G}(S)$ of $G(S)$ is defined orienting an edge $g(u,v) - g(x,y)$ as $g(u,v) \to g(x,y)$ if any geodesic disk through u and v contains x or y, or otherwise orienting $g(x,y) \to g(u,v)$. This orientation is consistent because of Lemma 3.

▶ **Theorem 7.** *For any set S of $n \geq 5$ points in a polygon, $\Pi(n) \geq \left\lceil \frac{n-2}{60} \right\rceil \approx 0.0166n$.*

Proof. By Kuratowski's Theorem [17], each subset $S' \subset S$, with $|S'| = 5$, contains four points w,x,y,z such that $g(w,x) \cap g(y,z) \neq \emptyset$. Moreover, the subset $\{w,x,y,z\}$ appears in exactly $n-4$ subsets of S with five elements. Thus, $I(S) \geq \frac{\binom{n}{5}}{n-4}$. By the definition of the intersection graph $G(S)$, $|E(G(S))| = I(S)$, thus $|E(G(S))| \geq \frac{\binom{n}{5}}{n-4}$.

Let $d^+(g(x,y))$ be the out-degree of a vertex $g(x,y)$ in $\overrightarrow{G}(S)$. Then,

$$\sum_{g(x,y) \in V(\overrightarrow{G}(S))} d^+(g(x,y)) = |E(G(S))| \geq \frac{\binom{n}{5}}{n-4}.$$

Since we have exactly $\binom{n}{2}$ vertices in $\overrightarrow{G}(S)$, there is a vertex $g(u_0,v_0)$ with

$$d^+(g(u_0,v_0)) \geq \frac{\frac{\binom{n}{5}}{n-4}}{\binom{n}{2}} = \frac{(n-2)(n-3)}{60}.$$

This means that any geodesic disk through u_0 and v_0 contains at least one endpoint of $\frac{(n-2)(n-3)}{60}$ geodesic paths $g(x,y)$, $x,y \in S$. Since each point could appear in at most $n-3$ of these geodesic paths, any geodesic disk through u_0 and v_0 contains at least $\left\lceil \frac{\frac{(n-2)(n-3)}{60}}{n-3} \right\rceil = \left\lceil \frac{n-2}{60} \right\rceil$ points of S different from u_0 and v_0. ◀

We now establish an improved lower bound on the number of points contained in any disk with x and y on its boundary, by generalizing the results by Hayward in [14] to the geodesic setting. We begin with a preliminary combinatorial result.

▶ **Lemma 8.** *Among any set S' of k points in a polygon P, there are at least $\binom{k-3}{2}$ pairs of points $x,y \in S'$ such that every geodesic disk through x and y contains at least one of the other $k-2$ points of S'.*

Proof. Let $G = (V,E)$ be the graph where $V = S'$, and two vertices are joined by an edge if at least one of the geodesic disks through them does not contain any of the other $k-2$ points of S'. The edge between u and v is drawn as the geodesic path $g(u,v)$. We claim that G is planar, and the edges can be drawn without crossings inside P. Suppose there are two edges $g(u,v), g(x,y)$ in G, for $u,v,x,y \in S'$, that intersect. By Lemma 3, one of the edges has the property that every geodesic disk through both of its endpoints contains at least one of the endpoints of the other diagonal, which contradicts the definition of the edges of G. Thus, G is a planar graph with k vertices. By Euler's formula, $|E| \leq 3k-6$, so S' contains at least $\binom{k}{2} - (3k-6) = \binom{k-3}{2}$ pairs that satisfy the property. ◀

Now, using Lemma 8, we obtain a tighter lower bound for the number of points contained in any geodesic disk with two points u and v on its boundary.

▶ **Theorem 9.** *For any set S of $n \geq 8$ points in a polygon,* $\Pi(n) \geq \left\lceil \frac{5}{84}(n-2) \right\rceil = \left\lceil \frac{n-2}{16.8} \right\rceil \approx 0.0595n$.

Proof. Two points in a subset $S' \subset S$ of k points *dominate* S' if every geodesic disk through them contains at least one of the other $k-2$ points of S'. There are $\binom{n}{k}$ subsets of size k. By Lemma 8, each of these sets has $\binom{k-3}{2}$ dominating pairs. The average number of subsets dominated by a pair is $\frac{\binom{k-3}{2}\binom{n}{k}}{\binom{n}{2}}$. Thus, there is a pair of points $\{u, v\} \in S'$ that dominates at least $\frac{\binom{k-3}{2}\binom{n}{k}}{\binom{n}{2}}$ subsets of S.

For each of these subsets, every geodesic disk through u and v contains at least one of the other $k-2$ points of S'. Since a point of $S \setminus \{u, v\}$ can be in at most $\binom{n-3}{k-3}$ subsets that include u and v, every geodesic disk through u and v contains at least

$$\frac{\binom{k-3}{2}\binom{n}{k}}{\binom{n}{2}\binom{n-3}{k-3}} = (n-2)\frac{(k-3)(k-4)}{k(k-1)(k-2)}$$

other points of S. The largest coefficient of $n-2$ is $\frac{5}{84}$, obtained for $k=8$ and for $k=9$. ◀

We now present our strongest lower bound for $\Pi(n)$ by generalizing the approach by Edelsbrunner et al. [13] to the geodesic setting.

▶ **Theorem 10.** *For any set S of n points in a polygon,* $\Pi(n) \geq \left\lceil \frac{n}{5} \right\rceil + 1$.

Proof. Unless specified otherwise, in this proof, order-k diagrams refer to order-k geodesic Voronoi diagrams. Recall that a geodesic disk through two points containing $k-1$ points in its interior has its center on an edge of the order-k diagram. Consider the set of all disks that pass through two points $p, q \in S$ in the polygon. These disks are all centered on the bisector $b(p, q)$, and there are at most $n-2$ points on the bisector where a third point of S is on the boundary of the disk. This partitions the bisector into *at most* $n-1$ pieces such that any circle through p and q centered on any point belonging to a fixed piece s contains in its interior the same number of points $\rho(s)$. The pieces s such that $\rho(s) = k-1$ are precisely the edges of the order-k diagram.

Now consider some fixed bisector $b(p, q)$. The edges of the order-1 through order-k diagrams on $b(p, q)$ form a number of connected components (which is strictly less than the total number of edges of all the diagrams). Let λ_k be the number of connected components formed by *all* the bisectors of the order-1 through order-k diagrams. The goal is to find the largest value of k such that $\lambda_k < \binom{n}{2}$. Every bisector is completely covered by a single connected component when $k = n-2$. When $\lambda_k < \binom{n}{2}$, by the pigeonhole principle there exists a bisector $b(p, q)$ that does not have any edges of the order-1 through order-k diagrams. Therefore, every geodesic disk with p and q on its boundary must contain in its interior at least k points.

We derive a formula for λ_k in terms of the number of edges in the order-1 to the order-k diagram by considering how the pieces must "pair up", in the sense that a piece s with $\rho(s) = i$ can only be adjacent to pieces s' with $\rho(s') = i \pm 1$. If a piece s with $\rho(s) = k$ is adjacent to two pieces s_1 and s_2 with $\rho(s_1) = \rho(s_2) = k-1$, the number of connected components is decreased by one. If s is adjacent to only one piece s_1 with $\rho(s_1) = k-1$, it extends an existing component. Otherwise, if it is not adjacent to any piece s_1 with $\rho(s_1) = k-1$, it creates a new connected component. Let ϵ_a, ϵ_b and ϵ_c be, respectively, the number of pieces s with $\rho(s) = k$ of the first, second, and third type. Then we have

$$\lambda_k = \lambda_{k-1} + \epsilon_c - \epsilon_a \tag{1}$$

Let e_k be the number of edges in an order-k diagram and let S_k be the number of edges of the order-k diagram that intersect the boundary. The total number of endpoints of all order-k edges is given by $2e_k - S_k$ since the edges that intersect the boundary of the polygon only contribute one endpoint. The edges intersecting the boundary are analogous to the unbounded edges in the Euclidean setting. Moreover, we note that $e_k = \epsilon_a + \epsilon_b + \epsilon_c$.

Let ϕ_i be the number of unpaired endpoints in λ_i. Consider all the unpaired endpoints of the connected components of λ_{k-1}. These endpoints can only come from edges in the order-$k-1$ diagram since all edges of the order-i diagrams for $i \in \{1, \ldots, k-2\}$ are all paired. Therefore, these unpaired endpoints can only pair up with edges from the order-k diagram, which implies that $\phi_{k-1} = 2\epsilon_a + \epsilon_b$. Thus, from Equation (1) and the fact that $e_k = \epsilon_a + \epsilon_b + \epsilon_c$, we have that $\lambda_k = \lambda_{k-1} + e_k - \phi_{k-1}$. If we iterate over k, we get that $\lambda_k = (e_k - \phi_{k-1}) + (e_{k-1} - \phi_{k-2}) + \ldots + (e_2 - \phi_1) + e_1 = \sum_{i=1}^{k} e_i - \sum_{i=1}^{k-1} \phi_i$. Note that ϕ_{i-1} is the number of unpaired endpoints in λ_{i-1}, which are the paired endpoints of λ_i. Therefore, $\phi_i + \phi_{i-1} = 2e_i - S_i$–the number of endpoints in order-i diagrams. We use this to rewrite the equation for λ_k in terms of e_i and S_i.

$$\begin{aligned}
\lambda_k &= \sum_{i=1}^{k} e_i - [(\phi_{k-1} + \phi_{k-2}) + (\phi_{k-3} + \phi_{k-4}) + \ldots] \text{ (end of sum depends on parity of } k) \\
&= \sum_{i=1}^{k} e_i - [(2e_{k-1} - S_{k-1}) + (2e_{k-3} - S_{k-3}) + \ldots + (2e_{2\text{-}or\text{-}1} - S_{2\text{-}or\text{-}1})] \text{ (depending on parity of } k) \\
&= \sum_{i=1}^{k} (-1)^{k-i} e_i + (S_{k-1} + S_{k-3} + \ldots + S_{2\text{-}or\text{-}1}) \text{ } (S_2 \text{ or } S_1 \text{ depending on parity of } k)
\end{aligned}$$

To complete the bound on λ_k, we need to bound the size of the order-j diagrams. Exact bounds on the size of order-j Euclidean Voronoi diagrams are known but do not apply since unlike the Euclidean case, the geodesic order-j Voronoi diagram has no unbounded edges. We can, however, establish upper and lower bounds using abstract Voronoi diagrams [16]: Geodesic bisectors fulfill all the axioms for an abstract Voronoi diagram except for the property of being unbounded, which in the geodesic setting corresponds to hitting the boundary of the polygon. We can crop this diagram with a closed Jordan curve that encloses all the intersections between bisectors (as is the case for the polygon and the geodesic bisectors) to have an abstract Voronoi diagram [20]. It is this cropping that no longer allows us to have an exact formula. Since an order-j geodesic Voronoi diagram is an order-j abstract Voronoi diagram, we have that the number of edges $e_j = 3(F_j - 1) - S_j$, where $F_j = 2jn - j^2 - n + 1 - \sum_{i=1}^{j-1} S_i$ is the number of faces of an order-j geodesic Voronoi diagram, see [8, Lemmas 15 and 16]. Hence, $e_j = 6jn - 3j^2 - 3n - 3\sum_{i=1}^{j-1} S_i - S_j$. Then we get

$$e_j - e_{j-1} = 6n - 6j + 3 - 2S_{j-1} - S_j. \tag{2}$$

In the final calculation, as in the proof of the lower bound in [13], we distinguish the case that k is even from k odd. However, in both cases we have that

$$\lambda_k = 3kn - \frac{3}{2}k^2 - \frac{3}{2}k - \sum_{i=1}^{k} S_i.$$

We use the fact that $\sum_{i=1}^{j} S_i \geq j(j+1)$, see [8, Lemma 11], which implies that

$$\lambda_k \leq 3kn - \frac{3}{2}k^2 - \frac{3}{2}k - k(k+1) = 3kn - \frac{5}{2}k^2 - \frac{5}{2}k,$$

which is a quadratic polynomial in k. We note that $\lambda_k < \binom{n}{2}$ whenever $k < \frac{n}{5}$, i.e., $k \leq \lceil \frac{n}{5} \rceil - 1$. Then, as mentioned above, the pigeonhole principle implies that every geodesic disk with p and q on its boundary must contain in its interior at least $k \leq \lceil \frac{n}{5} \rceil - 1$ points. Thus, we get that $\Pi(n) \geq \lceil \frac{n}{5} \rceil + 1$, with the plus one accounting for p and q. ◀

4.1 A Constant Fraction of Points Inside and Outside Geodesic Disks

In this section we show that for a point set S, there is a pair of points $x, y \in S$ such that any disk through them contains a constant fraction of points of S, both inside and outside the disks. We obtain this bound by modifying the previous proof of Theorem 9 by using Lemma 11. For completeness, we add here the proof of the bound.

▶ **Lemma 11.** *Among any set S' of k points in a polygon, there are at least $\binom{k-3}{2}$ pairs of points $x, y \in S'$ such that every geodesic disk through x and y leaves outside at least one of the other $k-2$ points of S'.*

Proof. Let $G = (V, E)$ be the graph where $V = S'$, and two vertices are joined by an edge if at least one of the geodesic disks through them contains one of the other $k-2$ points of S'. The edge between u and v is drawn as the geodesic path $g(u, v)$. We claim that G is planar, and the edges can be drawn without crossings inside P. Suppose there are two edges $g(u, v), g(x, y)$ in G, for $u, v, x, y \in S'$, that intersect. By Lemma 3, one of the edges has the property that every geodesic disk through both of its endpoints leaves outside at least one of the endpoints of the other diagonal, which contradicts the definition of the edges of G. Thus, G is a planar graph with k vertices. By Euler's formula, $|E| \leq 3k - 6$, so S' contains at least $\binom{k}{2} - (3k-6) = \binom{k-3}{2}$ pairs that satisfy the property. ◀

▶ **Theorem 12.** *For any set S of $n \geq 21$ points in a polygon, $\Pi^{in\text{-}out}(n) \geq \lceil \frac{16}{665}(n-2) \rceil \approx \lceil \frac{n-2}{41.6} \rceil \approx 0.024n$.*

Proof. In the same spirit as in the proof Theorem 9, we define two points in a subset $S' \subset S$ of k points to be *dominated by* S' if every geodesic disk through them leaves outside at least one of the other $k-2$ points of S'.

Among any set of k points in P there are at most $\binom{k}{2} - \binom{k-3}{2} = 3k-6$ pairs of points that do not satisfy the property of Lemma 8 and at most $3k-6$ pairs of points (possibly different than the previous pairs) that do not satisfy the property of Lemma 11. For $k \geq 11$, it holds that $2(3k-6) < \binom{k}{2}$, so that there is at least one pair of points $\{u, v\} \in S'$ that satisfies both properties, hence dominates and is dominated by at least $\frac{\left[2\binom{k-3}{2} - \binom{k}{2}\right]\binom{n}{k}}{\binom{n}{2}}$ subsets of S. For each of these subsets, every geodesic disk through u and v has at least one of the other $k-2$ points of S', both inside and outside. Since a point of $S \setminus \{u, v\}$ could be in at most $\binom{n-3}{k-3}$ subsets containing u and v, every geodesic disk through u and v has at least

$$\frac{\left[2\binom{k-3}{2} - \binom{k}{2}\right]\binom{n}{k}}{\binom{n}{2}\binom{n-3}{k-3}} = (n-2)\frac{2(k-3)(k-4) - k(k-1)}{k(k-1)(k-2)}$$

other points of S, both inside and outside. The largest coefficient of $n-2$ is obtained for $k = 21$, thus $k = 21$ leads to the largest possible coefficient, the one in the statement. ◀

4.2 Sets in geodesically convex position

In the Euclidean setting, Hayward et al. [15] proved that when points are in convex position, for every disk containing two points on the boundary, there always exists a pair of points such that any disk containing that pair contains at least $\lceil \frac{n}{3} \rceil + 1$. We generalize that result to the geodesic setting by showing that $\overline{\Pi}(n) \geq \lceil \frac{n}{3} \rceil + 1$. The geodesic case poses its unique challenges, for example, it is not always possible to find an enclosing geodesic disk with three points on the boundary. However, when no such enclosing geodesic disk exists, we show that there exists a pair of points such that every geodesic disk through them is an enclosing geodesic disk.

▶ **Lemma 13.** *Let S be a set of points in a polygon P. If there is no enclosing geodesic disk through three points of S, then there exist two points of S such that any geodesic disk through them is an enclosing geodesic disk.*

Proof. Define the notation $D(c)$ to be the disk $D(c, \max_{p \in S} |g(c,p)|)$ which is the geodesic disk centered at an arbitrary point $c \in P$, and having as radius the maximum geodesic distance from c to any other point $p \in S$. By construction, this is an enclosing geodesic disk. Consider an arbitrary disk $D(x)$ centered on a point $x \in S$ with a point $y \in S$ on the boundary of $D(x)$. Either $D(x)$ has a second point z on its boundary or only has y on its boundary. If the latter is true, then we show how to obtain an enclosing disk with a second point on its boundary. Consider the disk $D(x)$ as the center x moves along $g(x,y)$ towards y. Since the radius is shrinking, eventually, a second point $z \in S$ will reach the boundary of $D(x)$. Note that $D(x)$ always remains enclosing.

Thus, $D(x)$ is an enclosing disk with y and z on its boundary. We now show that every geodesic disk through y and z is enclosing. Let w be any point on $b(y,z)$. $D(w)$ has y and z on its boundary. Moreover, $D(w)$ must be enclosing. If $D(w)$ is not enclosing, then we contradict the fact that there is no enclosing geodesic disk through three points. The lemma follows. ◀

We now generalize the result from Hayward et al. [15] for point sets in convex position to the geodesic setting. In particular, when S is a set of points in P that is in geodesically convex position, then there is a pair of points such that any geodesic disk through them contains at least $\lceil \frac{n}{3} \rceil + 1$ points of S.

▶ **Theorem 14.** *For any set S of n points in geodesically convex position in a polygon, $\overline{\Pi}(n) \geq \lceil \frac{n}{3} \rceil + 1$.*

Proof. By Lemma 13, if there is no enclosing geodesic disk through three points, there exist two points of S such that any geodesic disk through them is an enclosing geodesic disk. Then all such disks contain all points of S, and the inequality holds.

Otherwise, there is an enclosing disk D that has three points on its boundary, namely x, y, z in clockwise order along the boundary of D. Let $f(\triangle(x,y,z))$ be the maximum among the number of points from S contained in each of the three regions $D \cap P^+(x,y), D \cap P^+(y,z)$ and $D \cap P^+(z,x)$. Among all enclosing disks with 3 points on the boundary, consider disk D_1 with points $u, v, w \in S$ on its boundary which minimizes $f(\triangle(u,v,w))$. There must be at least one region among $D_1 \cap P^+(u,v), D_1 \cap P^+(v,w)$ and $D_1 \cap P^+(w,u)$ whose interior has at least $\lceil \frac{n-3}{3} \rceil$ points. Assume, without loss of generality, that $D_1 \cap P^+(u,v)$ does.

If the interior of $D_1 \cap P^-(u,v)$ contains fewer than $\lceil \frac{n-3}{3} \rceil$ points, then we continuously change D_1 by the disks through u, v obtained when moving the center c of D_1 along the part of the bisector $b(u,v)$ to the right of $\ell(u,c)$, as in Figure 5. While doing this, all those

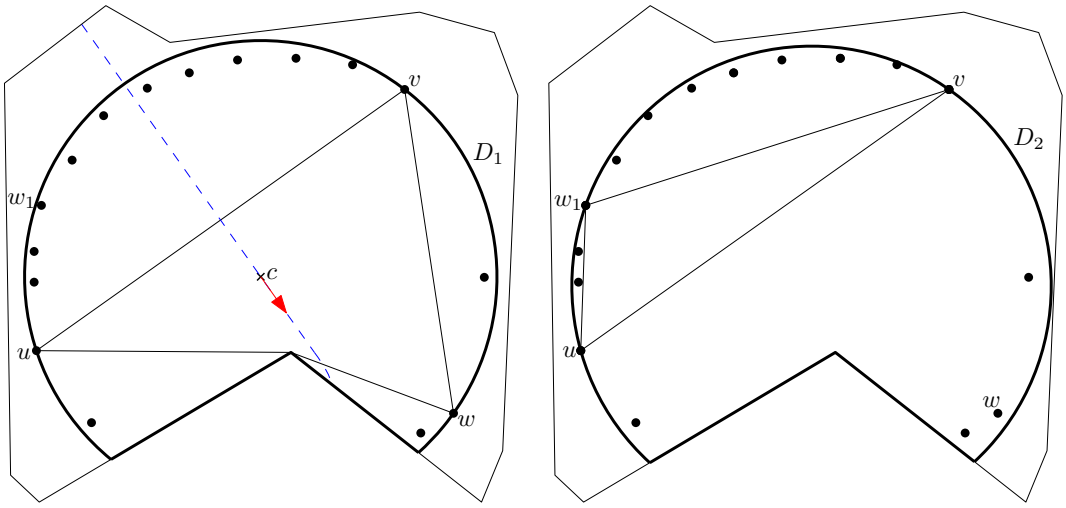


Figure 5 The geodesic disk D_2 is obtained by moving the center c of D_1 along the part of the bisector $b(u, v)$ (shown dashed) to the right of $\ell(u, c)$ until hitting point w_1.

disks will keep being enclosing disks, by Lemma 2. As we move c, we may arrive at one of two situations: Either the center c hits the boundary of the polygon, or the boundary of the current disk, D_2, hits a point $w_1 \in D_1 \cap P^+(u, v)$. We show, by contradiction, that the latter case cannot occur. If it occurred, the geodesic triangle $\triangle(u, w_1, v)$ divides D_2 into three regions $D_2 \cap P^+(u, w_1)$, $D_2 \cap P^+(w_1, v)$ and $D_2 \cap P^+(v, u)$. Note that $D_2 \cap P^+(v, u) = D_2 \cap P^-(u, v)$ contains fewer than $\lceil \frac{n-3}{3} \rceil$ points of S, since it is contained in $D_1 \cap P^-(u, v)$. Also, $D_2 \cap P^+(u, w_1)$ and $D_2 \cap P^+(w_1, v)$ both contain fewer points than $D_1 \cap P^+(u, v)$. Thus, $f(\triangle(u, w_1, v)) < f(\triangle(u, v, w))$, and contradicting the minimality of $f(\triangle(u, v, w))$.

Therefore, we will eventually hit the boundary. Then, every geodesic disk through u and v with center in the part of $b(u, v)$ to the right of $\ell(u, c)$ contains all the points in S, since the current D_1 is an enclosing disk. Also, by Lemma 2, every geodesic disk through u and v with center in the part of $b(u, v)$ to the left of $\ell(u, c)$ contains $D_1 \cap P^+(u, v)$, thus at least $\lceil \frac{n-3}{3} \rceil$ points of S.

It follows that there exist two points u, v such that any geodesic disk through u and v contains at least $\lceil \frac{n-3}{3} \rceil$ points of S in its interior. By accounting for u and v, on the disk boundary, we conclude that $\overline{\Pi}(n) \geq \lceil \frac{n}{3} \rceil + 1$. ◀

5 Diametral geodesic disk

In the Euclidean setting, Akiyama et al. [2] showed that for points in the plane (both in convex position and not) there always exists a diametral disk that contains $\lceil \frac{n}{3} \rceil + 1$ points of S and that this is a tight bound. In this section, we consider the problem of proving lower bounds on the values of $\Pi^{diam}(n)$ and $\overline{\Pi}^{diam}(n)$, generalizing the results in [2]. We prove that, among a set S of n points, there is a pair of points such that the diametral geodesic disk through them contains at least $\lceil \frac{n}{3} \rceil + 1$ points of S.

In order to obtain these lower bounds, we are going to use an enclosing geodesic disk with its center inside the geodesic convex hull of the points on its boundary.

▶ **Lemma 15.** *Let S be a set of points in a polygon. There exists an enclosing geodesic disk D with (i) three points x, y, z on the boundary, and the center of D inside $\triangle(x, y, z)$, or (ii) two points x, y on the boundary, and the center of D at the midpoint of $g(x, y)$.*

Proof. Let D be an enclosing geodesic disk with three points on its boundary. If such a disk does not exist, by Lemma 13, there exist two points $x, y \in S$ such that every geodesic disk through them is a geodesic enclosing disk. In particular, the disk $D\left(c_{xy}, \frac{|g(x,y)|}{2}\right)$ is an enclosing diametral disk with x and y as its diametral endpoints.

Otherwise, let $x, y, z \in S$ be the three points on the boundary of D, in clockwise order. If the center of the disk is inside the geodesic triangle $\triangle(x, y, z)$, we are done. Otherwise, the center is inside one of the regions $D \cap P^+(x, y), D \cap P^+(y, z)$ or $D \cap P^+(z, x)$. Assume, without loss of generality, it is in $D \cap P^+(x, y)$. Then we move the center c of D along $b(x, y)$ towards c_{xy}. If the center c reaches c_{xy}, then $D\left(c_{xy}, \frac{|g(x,y)|}{2}\right)$ is a diametral enclosing geodesic disk with x and y as its diametral endpoints. If the disk hits a third point $w \in S$, and the center of the disk is inside $\triangle(x, y, w)$, we are done. Otherwise, we repeat the process with the points x, y, w as the new points x, y, z until the center of the disk through the three points on the boundary of the disk lies inside the geodesic triangle formed by the three points on the boundary. This process ends because the distance to the center from the closest shortest path is decreasing at every iteration. ◀

▶ **Theorem 16.** *For any set S of n points in a polygon, $\Pi^{diam}(n) = \overline{\Pi}^{diam}(n) \geq \lceil \frac{n}{3} \rceil + 1$.*

Proof. Consider an enclosing disk D as guaranteed by Lemma 15. Either (i) D has exactly three points of S on the boundary, and the center of the geodesic disk is inside the geodesic triangle formed by the three points, or (ii) D has two points of S as diametral endpoints. For case (ii), D having two points on the boundary, the claimed result holds, since D contains all the points.

For case (i), let a, b and c be the three points on the boundary of the geodesic disk. We will prove that any point $x \in S$ is contained in at least one of the geodesic disks $D\left(c_{ab}, \frac{|g(a,b)|}{2}\right)$, $D\left(c_{bc}, \frac{|g(b,c)|}{2}\right)$ and $D\left(c_{ca}, \frac{|g(c,a)|}{2}\right)$.

First, consider the case where x is outside the geodesic triangle $\triangle(a, b, c)$. Then, x belongs to one of the regions $D \cap P^+(a, b), D \cap P^+(b, c)$, and $D \cap P^+(a, c)$. Without loss of generality, assume $x \in D \cap P^+(a, b)$. By Lemma 15, the center of the geodesic disk is inside $\triangle(a, b, c)$, so it does not belong to $D \cap P^+(a, b)$. Then, by Lemma 2, x is contained in the diametral geodesic disk $D\left(c_{ab}, \frac{|g(a,b)|}{2}\right)$.

Now, we prove the case where $x \in \triangle(a, b, c)$. Consider the geodesic paths $g(a, x), g(b, x)$ and $g(c, x)$. One of the angles $\angle axb, \angle bxc$ or $\angle axc$ is greater than or equal to $\frac{2\pi}{3}$, and the other two angles are smaller than or equal to $\frac{\pi}{3}$. Assume, without loss of generality, that $\angle bxc \geq \frac{2\pi}{3}$, see blue angle in Figure 6. Let o be the midpoint of the shortest path $g(b, c)$. We also know that one of the angles $\angle bxo$ or $\angle oxc$ is greater than or equal to $\frac{\pi}{3}$. Assume, without loss of generality, that it is the angle $\angle oxc$. When o is not visible from x, the angle $\angle oxc$ is equal to the angle $\angle bxc \geq \frac{2\pi}{3} > \frac{\pi}{2}$. Hence, by [21, Corollary 2], $|g(x, o)| < |g(o, c)| = |g(o, b)|$, and x is contained in the geodesic disk $D\left(c_{bc}, \frac{|g(b,c)|}{2}\right)$. In the case of o being visible from x, by Lemma 5, $|g(o, c)| \geq \min\{|g(c, x)|, |g(x, o)|\}$ and, by Lemma 6, $|g(x, o)| \leq \max\{|g(c, x)|, |g(o, c)|\}$, hence $|g(x, o)| \leq |g(o, c)| = |g(o, b)|$, and x is contained in the geodesic disk $D\left(c_{bc}, \frac{|g(b,c)|}{2}\right)$.

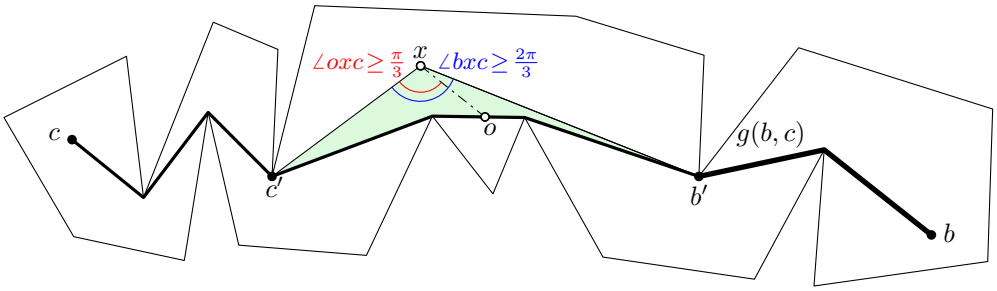


Figure 6 The point o is visible from x.

Then, the union of the geodesic disks $D\left(c_{ab}, \frac{|g(a,b)|}{2}\right)$, $D\left(c_{bc}, \frac{|g(b,c)|}{2}\right)$ and $D\left(c_{ca}, \frac{|g(c,a)|}{2}\right)$ covers all the points in P. Since the points a, b and c are counted twice,

$$\left|D\left(c_{ab}, \frac{|g(a,b)|}{2}\right) \cap P\right| + \left|D\left(c_{bc}, \frac{|g(b,c)|}{2}\right) \cap P\right| + \left|D\left(c_{ca}, \frac{|g(c,a)|}{2}\right) \cap P\right| > n + 3.$$

By the pigeonhole principle, the result follows. ◀

6 Bichromatic geodesic disks

In this section, we consider the bichromatic version of the problem, which has also received attention in the Euclidean setting. Let B be a set of $\frac{n}{2}$ blue points and R be a set of $\frac{n}{2}$ red points inside a polygon P, and let $S = B \cup R$.

Following the procedure in [25], we obtain a similar result to Theorem 7, and we prove that it is possible to find a bichromatic pair such that any geodesic disk containing them contains at least $\left\lceil \frac{|S|-2}{72} \right\rceil$ points of S.

▶ **Theorem 17.** *For any set S of $\frac{n}{2} \geq 3$ red points and $\frac{n}{2} \geq 3$ blue points in a polygon* $\Pi^{bichrom}(n) \geq \left\lceil \frac{n-2}{72} \right\rceil \approx 0.0139n$.

Proof. Let $G(S) = (V(G(S)), E(G(S)))$ be the intersection graph defined as follows: $V(S) = \{g(u,v) | u, v \in S, u \text{ is red, and } v \text{ is blue}\}$, and two vertices $g(u,v), g(x,y)$ are joined by an edge in $G(S)$ if $g(u,v) \cap g(x,y) \neq \emptyset$. The oriented graph $\overrightarrow{G}(S)$ of $G(S)$ is defined orienting an edge $g(u,v) - g(x,y)$ as $g(u,v) \to g(x,y)$ if any geodesic disk through u and v contains x or y, or otherwise orienting $g(x,y) \to g(u,v)$. By Lemma 3, if two of these paths intersect, then one of the two bichromatic pairs fulfills the property that any geodesic disk through it will contain one of the other two endpoints, so this orientation is consistent.

We now obtain the minimum number of intersections between all pairs of geodesic paths. Let, as in Theorem 7, $I(S)$ be the number of pairs of geodesic paths that intersect. For each $S' \subset S$ with exactly 3 blue and 3 red points, by Kuratowski's Theorem [17], there are two blue points $w, x \in S'$ and two red points $y, z \in S'$ such that $g(w,y) \cap g(x,z) \neq \emptyset$. There are $\binom{\frac{n}{2}}{3}^2$ subsets of this type. Furthermore, every intersection belongs to exactly $\left(\frac{n}{2} - 2\right)^2$ subsets of this type. Dividing by this number, we remove repetitions, and we obtain that $I(S) \geq \left(\frac{\binom{\frac{n}{2}}{3}}{\frac{n}{2}-2}\right)^2$.

By definition of the intersection graph $G(S)$, $|E(G(S))| = I(S)$ and $|V(G(S))| = \left(\frac{n}{2}\right)^2$. Hence, there exists one vertex $g(u_0, v_0) \in \overrightarrow{G}$ with out-degree at least $\frac{I(S)}{\left(\frac{n}{2}\right)^2}$.

Then, we have that $\frac{I(S)}{\binom{\frac{n}{2}}{2}} \geq \left(\frac{\frac{n}{2}-1}{6}\right)^2$. This means that there exist two points u_0 and v_0, one of each color, such that the geodesic path between them intersects at least $\left(\frac{\frac{n}{2}-1}{6}\right)^2$ other geodesic paths. Since every point is the endpoint of at most $\frac{n}{2}-1$ geodesic paths, every geodesic disk through this pair of points contains at least $\left(\frac{\frac{n}{2}-1}{6}\right)^2 \frac{1}{\frac{n}{2}-1} = \frac{\frac{n}{2}-1}{36} = \frac{n-2}{72}$ points of S. ◀

It was proved in [22] that any set of bichromatic points S, contains a pair $\{p, q\} \subset S$, one point of each color, such that any disk through p and q contains a positive fraction of the points of S, in particular $\frac{1}{36}$. We present the analogous colored result in the geodesic setting.

▶ **Theorem 18.** *For any set S of $\frac{n}{2} \geq 3$ blue points and $\frac{n}{2} \geq 3$ red points in a polygon, $\Pi^{bichrom}(n) \geq \lceil \frac{n-2}{36} \rceil + 2 \approx 0.0277n$.*

Proof. Let B be the set of blue points, and let R be the set of red points. By Kuratowski's Theorem and Lemma 3, for every set $Z \subset S$ of six points, three being blue and three being red, there is a pair $\{u, v\}$ of bichromatic points such that any geodesic disk with u and v on its boundary contains at least one point of $Z \setminus \{u, v\}$. Hence, there is a family $\mathcal{Z}$ and a pair $\{u, v\} \subset S$ of bichromatic points such that $\{u, v\}$ belongs to every $Z \in \mathcal{Z}$ and

$$|\mathcal{Z}| \geq \frac{\binom{\frac{n}{2}}{3}^2}{\binom{\frac{n}{2}}{1}^2} = \frac{\left(\frac{n}{2}-2\right)^2\left(\frac{n}{2}-1\right)^2}{6^2}. \tag{3}$$

Now we will upper-bound the value $|\mathcal{Z}|$. Let D be a geodesic disk with u and v on its boundary, and $m-2$ points in its interior. Each $Z \in \mathcal{Z}$ contains a point of $D \setminus \{u, v\}$. The set $\{u, v\}$ can be extended to Z, given that $D \cap (Z \setminus \{u, v\}) \neq \emptyset$, by (i) choosing a point of Z from the remaining $m-2$ points in D, (ii) choosing a point of the same color class from the remaining $\frac{n}{2}-2$ points in S, and (iii) choosing two points in S from the remaining color class. Thus, we get that

$$|\mathcal{Z}| \leq \binom{m-2}{1}\binom{\frac{n}{2}-2}{1}\binom{\frac{n}{2}-1}{2} = (m-2)\frac{\left(\frac{n}{2}-2\right)^2\left(\frac{n}{2}-1\right)}{2}. \tag{4}$$

Finally, from equations (3) and (4) we get $m-2 \geq \frac{\frac{n}{2}-1}{18} \Longrightarrow m \geq \frac{\frac{n}{2}+35}{18} = \frac{n+70}{36}$. ◀

7 Conclusion

We study different variants of the following question: Given a set S of n points in a polygon P, does there always exist two points $x, y \in S$, such that every geodesic disk containing x and y contains a constant fraction of the points of S? This question has been studied intensely in the Euclidean setting [2, 12–15, 19, 22, 23, 25]. We focus on the geodesic versions of this problem. We note that all the solutions to these problems lend themselves to polynomial time algorithms since the characterizations are based on standard structures in Computational Geometry such as order-k Voronoi diagrams. An obvious open problem is the following: Can our lower bounds be improved? Basically, some of our geodesic lower bounds are not as strong as in the Euclidean case. It would be interesting to determine whether there is a separation between the bounds in the Euclidean and geodesic case. Since the acceptance of

this paper by the WADS Program Committee, we have improved the bounds in Theorems 12 and 18 by modifying the technique using order-k geodesic Voronoi diagrams as was done in the proof of Theorem 10. The proof of these improved results can be found in [11]. Finally, the main open problem which has been open since 1988 is whether we can find a tight bound for the original question?

References

1 Hugo A. Akitaya, Greg Aloupis, Jeff Erickson, and Csaba D. Tóth. Recognizing weakly simple polygons. *Discrete & Computational Geometry*, 58(4):785–821, 2017. doi:10.1007/S00454-017-9918-3.

2 Jin Akiyama, Yoshiyasu Ishigami, Masatsugu Urabe, and Jorge Urrutia. On circles containing the maximum number of points. *Discrete Mathematics*, 151(1-3):15–18, 1996. doi:10.1016/0012-365X(94)00076-U.

3 Boris Aronov. On the geodesic Voronoi diagram of point sites in a simple polygon. *Algorithmica*, 4(1-4):109–140, 1989. doi:10.1007/BF01553882.

4 Boris Aronov, Steven Fortune, and Gordon Wilfong. The furthest-site geodesic Voronoi diagram. *Discrete & Computational Geometry*, 9:217–255, 1993. doi:10.1007/BF02189321.

5 Imre Bárány and David G. Larman. A combinatorial property of points and ellipsoids. *Discrete & Computational Geometry*, 5:375–382, 1990. doi:10.1007/BF02187798.

6 Imre Bárány, James H. Schmerl, Stuart J. Sidney, and Jorge Urrutia. A combinatorial result about points and balls in Euclidean space. *Discrete & Computational Geometry*, 4:259–262, 1989. doi:10.1007/BF02187727.

7 Ahmad Biniaz, Prosenjit Bose, Anil Maheshwari, and Michiel Smid. Plane geodesic spanning trees, Hamiltonian cycles, and perfect matchings in a simple polygon. *Computational Geometry*, 57:27–39, 2016. doi:10.1016/J.COMGEO.2016.05.004.

8 Cecilia Bohler, Panagiotis Cheilaris, Rolf Klein, Chih-Hung Liu, Evanthia Papadopoulou, and Maksym Zavershynskyi. On the complexity of higher order abstract Voronoi diagrams. *Computational Geometry*, 48(8):539–551, 2015. doi:10.1016/J.COMGEO.2015.04.008.

9 Prosenjit Bose, Paz Carmi, and Thomas C. Shermer. Piercing pairwise intersecting geodesic disks. *Computational Geometry*, 98:101774, 2021. doi:10.1016/J.COMGEO.2021.101774.

10 Prosenjit Bose, Anthony D'Angelo, and Stephane Durocher. Approximating the smallest k-enclosing geodesic disc in a simple polygon. In *WADS*, volume 14079 of *Lecture Notes in Computer Science*, pages 179–192. Springer, 2023. doi:10.1007/978-3-031-38906-1_13.

11 Prosenjit Bose, Guillermo Esteban, David Orden, Rodrigo Silveira, and Tyler Tuttle. On geodesic disks enclosing many points, 2025. doi:10.48550/arXiv.2506.06477.

12 Mercè Claverol, Clemens Huemer, and Alejandra Martínez-Moraian. On circles enclosing many points. *Discrete Mathematics*, 344(10):112541, 2021. doi:10.1016/J.DISC.2021.112541.

13 Herbert Edelsbrunner, Nany Hasan, Raimund Seidel, and Xiao Jun Shen. Circles through two points that always enclose many points. *Geometriae Dedicata*, 32(1):1–12, 1989.

14 Ryan Hayward. A note on the circle containment problem. *Discrete & Computational Geometry*, 4(3):263–264, 1989. doi:10.1007/BF02187728.

15 Ryan Hayward, David Rappaport, and Rephael Wenger. Some extremal results on circles containing points. *Discrete & Computational Geometry*, 4(3):253–258, 1989.

16 Rolf Klein. *Concrete and Abstract Voronoi Diagrams*, volume 400 of *Lecture Notes in Computer Science*. Springer, 1989. doi:10.1007/3-540-52055-4.

17 Casimir Kuratowski. Sur le probleme des courbes gauches en topologie. *Fundamenta mathematicae*, 15(1):271–283, 1930.

18 Der-Tsai Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE transactions on computers*, 100(6):478–487, 1982. doi:10.1109/TC.1982.1676031.

19 Victor Neumann-Lara and Jorge Urrutia. A combinatorial result on points and circles on the plane. *Discrete mathematics*, 69(2):173–178, 1988. doi:10.1016/0012-365X(88)90015-5.

20 Evanthia Papadopoulou. Abstract Voronoi-like graphs: Extending Delaunay's theorem and applications. In *SoCG*, volume 258 of *LIPIcs*, pages 52:1–52:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.SOCG.2023.52.

21 Richard Pollack, Micha Sharir, and Günter Rote. Computing the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 4(6):611–626, 1989. doi:10.1007/BF02187751.

22 Maria N. Prodromou. A combinatorial property of points and balls, a colored version. *Discrete & Computational Geometry*, 38(4):641–650, 2007. doi:10.1007/S00454-007-9003-4.

23 Pedro A. Ramos and Raquel Viaña. Depth of segments and circles through points enclosing many points: a note. *Computational Geometry*, 42(4):338–341, 2009. doi:10.1016/J.COMGEO.2008.07.001.

24 Godfried Toussaint. Computing geodesic properties inside a simple polygon. *Revue d'intelligence artificielle*, 3(2):9–42, 1989.

25 Pilar Valencia Saravia. *Problemas de cobertura circular*. PhD thesis, Universidad Nacional Autónoma de México, 2005.

Crossing and Independent Families Among Polygons

Anna Brötzner
Malmö University, Sweden

Robert Ganian
Algorithms and Complexity Group, TU Wien, Austria

Thekla Hamm
TU Eindhoven, The Netherlands

Fabian Klute
Universitat Politècnica de Catalunya, Barcelona, Spain

Irene Parada
Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract

Given a set A of points in the plane, a family of line segments forming a matching in A is called crossing (or independent) if each pair of segments in the family intersects (or is non-intersecting, respectively). In past works, these notions have been generalized to polygons by identifying the points in A with the vertices of a given set of polygons and forbidding the line segments from intersecting or overlapping with polygon walls. In this work, we study the computational complexity of computing maximum crossing and independent families in this more general setting.

As our first two results, we show that both problems are NP-hard already when the polygons are triangles. Motivated by this, we turn to parameterized algorithms. For our main algorithmic results, we consider the number of polygons on the input as the natural parameter and under this parameterization obtain a fixed-parameter algorithm for computing a largest crossing family among these polygons, and a separate XP-algorithm for computing a largest independent family that lies in one of the faces of the polygonal domain.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases crossing families, crossing-free matchings, segment intersection graphs, computational geometry, parameterized algorithms

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.11

Funding *Anna Brötzner*: supported by grant 2021-03810 (Illuminate: provably good algorithms for guarding problems) from the Swedish Research Council (Vetenskapsrådet).
Robert Ganian: Projects No. 10.55776/Y1329 and 10.55776/COE12 of the Austrian Science Fund (FWF), Project No. 10.47379/ICT22029 of the Vienna Science Foundation (WWTF).
Fabian Klute: Partially supported by grant PID2023-150725NB-I00 funded by MICIU/AEI/10.13039/501100011033.
Irene Parada: Serra Húnter Fellow. Partially supported by grant PID2023-150725NB-I00 funded by MICIU/AEI/10.13039/501100011033.

1 Introduction

Given n points in the plane, a *crossing family* is a set of straight-line segments with endpoints among these n points such that each pair of segments in the set cross in their interior. The natural counterpart to crossing families are sets of segments which are pairwise non-intersecting (not even at the endpoints); these are studied under the term *crossing-free families*, *plane matchings*, or *independent families* [19].

© Anna Brötzner, Robert Ganian, Thekla Hamm, Fabian Klute, and Irene Parada;
licensed under Creative Commons License CC-BY 4.0

In 1994, Aronov et al. [4] showed that for any point set in general position there is always a crossing family of size $\Omega(\sqrt{n})$ and since then the problem of quantifying the size of the largest possible crossing family has become a prominent and challenging task in discrete geometry. While it is generally conjectured that the size is in fact $\Theta(n)$ [6], only recently Pach, Rubin, and Tardos [20] achieved a breakthrough result establishing an $n^{1-o(n)}$ bound. Similarly, researchers have investigated crossing-free matchings that can be drawn on a point set. While all point sets admit matchings that cover all but at most one point, a large body of research has been devoted to related problems such as counting how many such matchings exist in a point set [24, 25], considering the lengths of segments in a crossing-free matching [2, 11], or finding large matchings in bicolored point sets [5, 13, 15]. Taking an algorithmic viewpoint, it is natural to ask about the complexity of computing a largest crossing or independent family. While the latter problem (IndependentFamily) is known to be solvable in $\mathcal{O}(n \log n)$-time [13], the complexity of the former (CrossingFamily) remains wide open.

In this work, we consider a more general setting in which the points are the vertices of a set of simple polygons and the segments are segments between two polygon vertices. In particular, we only allow segments which do not intersect the polygon boundary edges in their relative interior. In this perspective, these boundaries can be seen as obstacles in the plane and the segments model visibility, whereas the original problem forms a degenerate case where each polygon is a single point. However, compared to the setting of point sets, the complexity landscape of the problems changes drastically. In fact, already for one polygon the problem of computing a maximum independent family becomes intractable [21], while finding a maximum crossing family in one polygon can be solved in polynomial time [12].

A natural way of tackling these two problems is through the intersection graph of the visibility segments. In particular, one can form a graph by considering each segment as a vertex and connecting two vertices whenever their respective segments intersect. Computing largest crossing and independent families then precisely corresponds to finding a maximum clique or independent set (respectively) in this graph.

The study of these two problems on general segment intersection graphs has a long and storied history. For independent sets in this graph class, Kratochvíl and Nešetřil [14] showed NP-hardness in 1990 and Marx showed that the problem is even W[1]-hard when parameterized by the solution size [16]. For clique, the computational complexity on segment intersection graphs was a long-standing open problem; after partial progress in 1992 [17], this was finally settled via the NP-hardness reduction of Cabello, Cardinal, and Langermann [7]. Unfortunately, none of these lower bounds can be directly carried over to the considered generalization of CrossingFamily and IndependentFamily.

Apart from the considerations above, the problem of finding a large independent family in our setting is closely related to the task of augmenting a given geometric structure by a plane matching. Algorithms and lower bounds for this task have also received considerable attention in the literature [1, 21, 22, 23].

Contributions. Our first result concerns the classical complexity of finding maximum crossing and independent families. For both problems, we demonstrate that they are NP-hard even when all polygons are as simple as possible, namely triangles. Our reductions are from finding a maximum clique and independent set, respectively, in segment intersection graphs [7, 14]. The main idea of the reduction is to "emulate" each segment of the given segment intersection graph using the endpoints of special triangles. And while the principal logic of the reduction can be explained in a streamlined fashion, the underlying gadgets

require a careful construction and the precise placement of every triangle. Especially, it is necessary to find a placement of only triangles such that the vertices responsible for the "emulation" of the segment do not interact with gadgets representing other segments.

To overcome the complexity-theoretic intractability of these problems we turn to the tools of parameterized algorithms. We consider the parameterized problems of finding a largest crossing or independent family among a set of k simple polygons, choosing k as our parameter. For parameterized problems whose unparameterized versions are NP-hard the most desirable runtime behavior of an algorithm is of the form $\mathcal{O}(f(k) \cdot n^c)$ for some computable function f, i.e., a fixed-parameter algorithm. Allowing for more running time, we also consider so-called XP-algorithms where the running time may be in $\mathcal{O}(n^{f(k)})$.

As our first algorithmic result, we show that under this parameterization the problem of finding a largest crossing family indeed admits a fixed-parameter algorithm. For this, we consider the segments in a solution ordered by their slopes, together with the polygons on which the segment endpoints lie. Crucially, we argue that this sequence can in fact be viewed as a sequence of what we call *bundles* which are sets of segments that are consecutive in the sequence and are between a pair of two (not necessarily distinct) polygons. We show that the length of this sequence of bundles is bounded in the number of polygons, which allows us to efficiently branch on the order of pairs of polygons associated with the sequence of bundles. We use the resulting sequence as basis of a dynamic programming approach in which we look at one bundle at a time and compute and tabulate a best option for each last segment (delimiter) of the bundle. We show that it is possible to do that only looking at such tabulated information of the previous bundle and the branched information. The main challenge here is that choices made for a bundle that are compatible with preceding bundles may not be compatible with later bundles or segments in them; carefully filtering our choices without missing a potential solution, we can guarantee that this does not happen.

Turning to our second algorithm, we first realize that for independent families, the above mentioned result by Pilz et al. [21] indicates that the problem is already NP-hard for one polygon, ruling out the existence of even an XP-algorithm. Crucial to their reduction is the fact that segments may appear on the outside and the inside of the polygon. Consequently, we consider the case when segments may only appear on one side of each polygon. We give an XP-algorithm solving this case, again with the number k of polygons on the input as the parameter. The idea is to partition a hypothetical solution into $\mathcal{O}(k^2)$ independent parts, each of which is bounded by two segments of the solution. Branching over which segments act as these boundaries allows us to solve the instance by leveraging polynomial-time algorithms for finding maximum independent sets in circle graphs.

2 Preliminaries

We assume familiarity with the foundations of *parameterized complexity* [8] and standard terminology used in the area of computational geometry, including the notions of *(simple) curves* and *(straight-)line segments*, *polygons* and their *vertices* [10].

All of our objects are placed in the plane. We say that a simple closed curve τ *encloses* an object X if X is contained in the face bounded by τ. Two points a, b are *visible* to (or *see*) each other if the a-b line segment neither touches nor crosses any other object in the plane; we call such segments *visibility segments*. Extending this, a set of points is *visible* from a point a if each point in the set is visible from a. We assume all polygons in this work to be simple (i.e., with no self-intersections) and pairwise non-intersecting. Moreover, we assume w.l.o.g. that the vertices of each polygon are in general position, meaning that no

three vertices lie on a straight line. We explicitly note that polygons are allowed to nest each other, i.e., one may be drawn in the interior of another. We use the notation P° to denote the interior of a polygon.

We say that a *crossing family* among a set of polygons is a set of pairwise crossing visibility segments between vertices of the polygons. Similarly, an *independent family* (also called *non-crossing matching*) among a set of polygons is a set of pairwise non-intersecting visibility segments between vertices of the polygons; we explicitly remark that such segments cannot intersect even at the endpoints. We can now define our main problems of interest:

▶ **Problem 1** (MaxCrossingFamily). *Given a set $\mathcal{P}$ of polygons, find a largest crossing family C among $\mathcal{P}$.*

▶ **Problem 2** (MaxIndependentFamily). *Given a set $\mathcal{P}$ of polygons, find a largest independent family F among $\mathcal{P}$.*

Unfortunately, solving MaxIndependentFamily is known to be NP-hard even if $|\mathcal{P}| = 1$ [21], precisely because of the fact that visibility segments may occur both inside and outside of polygons. Hence, it will be useful to also consider a variant of the problem where polygons represent bounded-sized regions of the plane that are inaccessible, i.e., F does not contain any segment that intersects the interior of a polygon. Formally:

▶ **Problem 3** (MaxIndependentFamilyWithHoles). *Given a set $\mathcal{P}$ of polygons, find a largest independent family F among $\mathcal{P}$ that lies completely in the unbounded face.*

We remark that – in contrast to MaxIndependentFamily– the above problem can be solved in polynomial time for $|\mathcal{P}| = 1$, as it can be reduced to the well-studied problem of finding a maximum independent set in a circle graph [12].

3 Finding maximum crossing and independent families is hard

In this section, we prove that both MaxIndependentFamily and MaxCrossingFamily are NP-hard, even if the polygons on the input are triangles. We present reductions from the corresponding problems of finding a maximum independent set and maximum clique, respectively, in segment intersection graphs. Both are well known to be NP-hard [7, 14]. Here we give an overview of the reductions.

In either reduction, we start from a given set of segments $\mathcal{S}$ and construct a set of triangles $\mathcal{T}$ such that a maximum set of pairwise crossing or non-crossing segments in $\mathcal{S}$ corresponds to a maximum crossing family and independent family, respectively, among $\mathcal{T}$.

We first sketch the construction for MaxIndependentFamily. The idea of the reduction is to replace each $s = pq \in \mathcal{S}$ by a so-called *segment gadget*. Each segment gadget consists of ten triangles, divided into two groups of five triangles each, where each group forms an *endpoint gadget*. As the name suggests, the two endpoint gadgets which we denote by $\mathcal{T}_p$ and $\mathcal{T}_q$ are placed at the endpoints p and q of s, respectively. As every endpoint gadget contains exactly 15 vertices, there is at least one vertex in any independent family among the triangles of the endpoint gadget which can never be matched to a vertex also in the same endpoint gadget. The goal is to place the triangles in such a way that a solution will match a pair of vertices in endpoint gadgets if and only if the underlying segment s lies in a corresponding set of non-crossing segments in $\mathcal{S}$.

An endpoint gadget $\mathcal{T}_p$ contains three types of triangles: three *outer* triangles, one *inner triangle*, and one *shooting* triangle. The outer triangles are three congruent triangles placed around the inner triangle in such a way that each of them has one side on the boundary of the

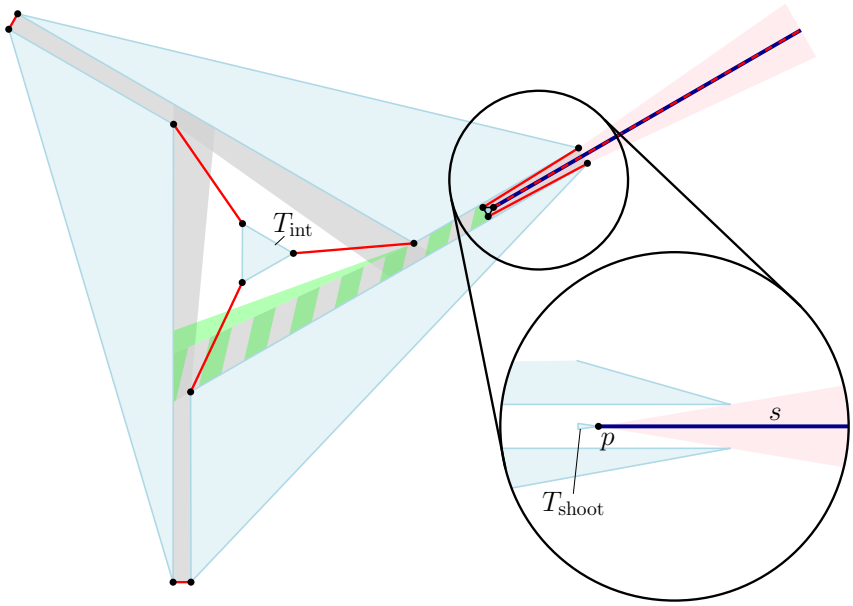


Figure 1 Endpoint gadget with three outer triangles, inner triangle T_{int}, and shooting triangle T_{shoot}. The original segment s is indicated in dark blue. The areas marked in gray are visible from the exterior of the gadget; the area marked in green is the interior area visible to T_{shoot}; the area marked in pink is the area outside the gadget visible from T_{shoot}. The red segments are segments of a maximum independent family within the gadget.

gadget's convex hull, and such that every pair of outer triangles forms a narrow rectangular corridor. One of these corridors is parallel to the supporting line of s and contains the endpoint p in its interior. At the position of p, we place the shooting triangle T_{shoot}. The construction can be seen in Figure 1. An analogous endpoint gadget $\mathcal{T}_q$ is placed at q. By carefully scaling the corridor, we achieve that the vertices of the shooting triangles only see some vertices in their own endpoint gadget, and vertices in the corridor around q in $\mathcal{T}_q$.

Finally, the three outer triangles delimit a triangular region in the interior of the convex hull of $\mathcal{T}_p$ that is visible neither via the corridors nor from the vertices of T_{shoot}. Consequently, the vertices of the inner triangle only see the three vertices of outer triangles that are interior to the convex hull of the endpoint gadget. This forces the corresponding three segments between these vertices to occur in every maximum independent family among $\mathcal{T}$.

Given a set of k pairwise non-crossing segments in $\mathcal{S}$ it is straight-forward to construct an independent family of size $14 \cdot |\mathcal{S}| + k$ among the triangles in $\mathcal{T}$. Vice versa, for any independent family F of size $14 \cdot |\mathcal{S}| + k$ among $\mathcal{T}$ we find that for k segment gadgets a segment in F has to have both its endpoints among the vertices of the two shooting triangles in the segment gadget. Since two segments connecting shooting triangles in different segment gadgets cross if and only if the corresponding segments in $\mathcal{S}$ cross we obtain a set of pairwise non-crossing segments of size k in $\mathcal{S}$ from F.

▶ **Theorem 1.** MaxIndependentFamily *is* NP*-hard even if all polygons are triangles.*

For MaxCrossingFamily we need to modify the above construction, as vertices of the outer triangles may be matched to each other and thereby create a large crossing family independent from the shooting triangles. Therefore, we adapt the endpoint gadgets to ensure that many pairwise crossing visibility segments have to be added for each segment gadget. To this end, we replace in each endpoint gadget the shooting triangle by a set of $7 \cdot |\mathcal{S}|$ shooting

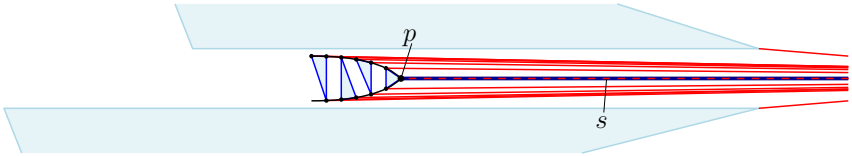


Figure 2 Modification of endpoint gadget. The original segment s is indicated in dark blue. The red segments are segments of a maximum crossing family within the segment gadget.

triangles such that the segments between shooting triangles of the two endpoint gadgets of each segment gadget together form a crossing family of size $21 \cdot |\mathcal{S}|$, see Figure 2. Additionally, four segments between vertices of the outer triangles can be added to this crossing family. Then, a maximum crossing family of segments within a segment gadget has size $21 \cdot |\mathcal{S}| + 4$.

In the whole set of triangles $\mathcal{T}$, a maximum crossing family consists of segments between two vertices in the same segment gadget, and segments between two vertices in different segment gadgets. We can upper-bound the number of the latter ones by $6 \cdot |\mathcal{S}|$, since every endpoint gadget contributes at most six vertices as endpoints of these segments. Consequently, even if all segments between all vertices of outer triangles in $\mathcal{T}$ could be added, this would not offset the contribution of just one segment gadget. Arguing as for independent families, we obtain that $\mathcal{T}$ contains a maximum crossing family of size at least $(21 \cdot |\mathcal{S}| + 4)k$ if and only if there is a set of k pairwise crossing segments in $\mathcal{S}$.

▶ **Theorem 2.** MaxCrossingFamily *is* NP*-hard even if all polygons are triangles.*

4 Finding crossing families among k polygons

Let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be a set of k non-intersecting polygons, possibly nesting. In this section we present a fixed-parameter algorithm to solve MaxCrossingFamily when parameterized by k. Thus, the number k of polygons is bounded, but the numbers $n_1, n_2, \ldots, n_k$ of vertices per polygon may be unbounded. The input size is $n = \sum_{i=1}^{k} n_i$.

The $k > 0$ polygons in $\mathcal{P}$ define $k+1$ faces. Our algorithm will compute a largest crossing family among $\mathcal{P}$ in each face independently. Since segments in different faces cannot cross, the solution to MaxCrossingFamily is the largest computed crossing family. For the remainder of this section, we therefore ask for a largest crossing family in a given face f.

For purely technical reasons, we assume w.l.o.g. that there is no vertical segment connecting two polygon vertices (else we perform a slight rotation of the whole instance). This merely ensures that each segment has a well-defined *slope*, which we will later use to obtain an ordering of the segments. For polygons $P, Q \in \mathcal{P}$ (not necessarily distinct) we define their *class* (P, Q) to be the set of segments in which the right endpoint is a vertex of P and the left endpoint is a vertex of Q. In a crossing family C, we consider the segments uniquely sorted by decreasing slope; note that segments with the same slope cannot cross. This naturally gives rise to the sequence of classes of these segments which we compress to a sequence Π in which consecutive occurrences of a class are merged. Note also that a class can appear more than once in Π; each such appearance corresponds to a *bundle* of segments in C, formally a maximal set of segments of the same class that is consecutive in the ordering of the crossing family by slopes. We use their corresponding order in Π to label the bundles $B_1, \ldots B_\ell$. Despite the repetitions, we can provide the following bound on ℓ:

▶ **Lemma 3.** *The length ℓ of Π is at most $4k - 3$.*

Proof. Let Π_r (Π_l) be the sequence of polygons that arises when picking only the first (resp. second) elements of the classes in Π. Since no two polygons intersect each other and no polygon intersects a segment in the crossing family, neither of these sequences contains a subsequence of the form $PQPQ$ with polygons $P \neq Q$. Indeed, two segments that cross and have one of their endpoints in P, together with the boundary of P, define a bounded region. Polygon Q can either lie inside it or outside it, but not both.

This lack of $PQPQ$ subsequences implies that Π_r and Π_l are Davenport-Schinzel sequences of order 2 on the k polygons, and their length is therefore at most $2k-1$ [9]. In the combined sequence Π, two consecutive classes differ in at least one of the polygons. As there are at most $2k-2$ changes in each of the two sequences of polygons, there are at most $2 \cdot (2k-2) = 4k-4$ changes in Π. Therefore, Π has length at most $4k-3$. ◀

We now describe our algorithm, which on a high level works by dynamically computing the largest crossing family of a consecutive sequence of bundles with specific first and last segments under the condition that the crossing family is consistent with the selected Π.

Step 0: Precomputation. We precompute which pairs of vertices of polygons see each other (without crossing any polygon boundary). These are the $\mathcal{O}(n^2)$ *valid* segments and they can trivially be computed in $\mathcal{O}(n^3)$ time.

Step 1: Branching. In the next step of the algorithm, we guess the sequence Π for a solution of MaxCrossingFamily, that is, we run the algorithm for each possible sequence Π. This fixes the order of the bundles and also determines the subset of polygons that contribute to the solution. We call them the *contributing polygons*, $\mathcal{P}'$. By Lemma 3 and the fact that there are k different polygons, we have at most $k^{2(4k-3)}$ possibilities for Π.

For a crossing family, the *extreme delimiters* d_0 (*start delimiter*) and d_ℓ (*end delimiter*) are the segments with maximum and minimum slopes, respectively. We branch on d_0 in a solution. For a class K_i, let $|K_i|$ denote the product of the number of vertices of the two polygons in class K_i. Since d_0 must be in the first class K_1 of Π, the number of choices is bounded by $|K_1|$. In the worst case, this is $\mathcal{O}(n^2)$.

Thus, we run the algorithm in the next step at most $\mathcal{O}(n^2 k^{2(4k-3)})$ times and select the largest crossing family computed among them.

Step 2: Optimizing delimiters. The key information for the solution are the extreme segments for each bundle. A crossing family C *respects* a sequence of classes Π if the compressed sequence of classes of the segments of C sorted by decreasing slope is Π. For a crossing family C that respects Π and for a bundle B_i, the *delimiter* d_i of B_i in C is the segment of C in B_i that has the smallest slope. For notational convenience, we preface the sequence of classes Π by class K_0, which is equal to K_1. We refer to the resulting extended sequence as Π^+. We say that the bundle and delimiter of K_0 is the start delimiter d_0.

Given two segments a and b we say that a point p lies (strictly) *between* them if p lies in the (open) region bounded by the supporting lines of a and b that does not contain any vertical ray, the *horizontal double wedge* of a and b. (We call the (open) interior of the complement of a horizontal double wedge the *vertical double wedge*.) We say that a segment c is between a and b if both endpoints of c are between a and b, one in each side of the horizontal double wedge. Note that then all the segments of C between two consecutive delimiters d_i and d_{i+1} together with d_{i+1} (and d_0 if $d_0 = d_i$) form the bundle B_{i+1}. We also say that a polygon lies between a and b if all its vertices lie strictly between a and b, on the same side of the horizontal double wedge.

We now outline the procedure of our dynamic programming algorithm. Given a set $\mathcal{P}$ of polygons, a sequence of classes Π, and a segment d_0, we aim to find a largest crossing family C among $\mathcal{P}$ that respects Π and begins with delimiter d_0. For this, we dynamically find an optimal sequence of bundle delimiters. Our algorithm processes the classes in the order given by Π. When processing class $i \in [1, \ell]$, we assume that we have solved the subproblems for every valid delimiter d_{i-1} in class K_{i-1}, and that these solutions are stored in a table with one row per delimiter d_{i-1}. In that row we store the maximum *possible* size $|C_{i-1}(d_0, d_{i-1})|$ of a crossing family between (i.e., with extreme delimiters) d_0 and d_{i-1}. With *possible* we here mean that at each step we already take into account that we will need to complete the crossing family to one that respects Π. This table also includes in each row an iteratively built crossing family with extreme delimiters d_0, d_{i-1}, for convenience of solution reconstruction.

At the beginning of the algorithm, the table only contains one row for delimiter d_0, with size 1 and sequence d_0. In step i this table gets updated to one that contains the maximum possible size of a crossing family between d_0 and d_i, $|C_i(d_0, d_i)|$, for each valid delimiter d_i in class K_i. We calculate the values $|C_i(d_0, t)|$ for each t in class K_i from the previously tabulated values $|C_{i-1}(d_0, s)|$ with s in class K_{i-1}. At step i we compute, for each possible pair s, t of delimiters of classes K_{i-1} and K_i, respectively, the best possible crossing family extension $C_i(s, t)$ consisting of segments of class K_i between delimiters s and t, not including s. We denote its size as $|C_i(s, t)|$. In this computation, we take into account that the final sequence needs to respect Π. Also, not all (s, t) pairs are possible: they need to cross and be sorted by slope, but also t should cross the sequence of previous delimiters of s. Crucially, by checking certain necessary conditions, we can make this check independent of the sequence of previous delimiters of s. This restricts the sizes at each step and disallows certain delimiters (which we mark as having size $-\infty$). With the above notation, for each t in class K_i,

$$|C_i(d_0, t)| = \max_{s \text{ delimiter of } K_{i-1}} \{|C_{i-1}(d_0, s)| + |C_i(s, t)|\}.$$

In this way, our algorithm assigns to each delimiter t in class K_i with $i \geq 1$ a delimiter s in the previous class K_{i-1}. To obtain the crossing family in the row corresponding to t, we take the one corresponding to s and append $C_i(s, t)$.

In the final step of the algorithm, we select a delimiter d_ℓ that maximizes $|C_\ell(d_0, d_\ell)|$; the corresponding crossing family is stored in the last table in the row for d_ℓ.

The next lemma specifies how we compute each value $|C_i(s, t)|$, for s and t in classes K_{i-1} and K_i, respectively, and the underlying crossing family $C_i(s, t)$. Moreover, it shows that such a crossing family is optimal if s and t are the delimiters of the corresponding classes. In particular, all other segments in a crossing family respecting Π will cross them.

▶ **Lemma 4.** *Assume we are given a family of polygons $\mathcal{P} = \{P_1, \ldots, P_k\}$, the start delimiter d_0, an extended sequence of classes $\Pi^+ = (K_0 = K_1, \ldots, K_\ell)$, a superset $\mathcal{Q}$ of the contributing polygons of Π^+, and the two delimiters d_i and d_{i+1} of (consecutive bundles corresponding to) classes K_i and K_{i+1}, respectively, satisfying*

(⋆) *$\{d_0, d_i, d_{i+1}\}$ form a crossing family, are sorted by decreasing slope, and there is no polygon in $\mathcal{Q}$ between d_i and d_{i+1}.*

Then we can compute, in time polynomial in the total number n of polygon vertices, a largest crossing family C of $\mathcal{P}$ such that

(i) *its extreme delimiters are d_i and d_{i+1},*

(ii) *all its segments except maybe d_i are in class K_{i+1}, and*

(iii) *no polygon in $\mathcal{Q}$ lies in a triangle bounded by three segments of C.*

Moreover, for all crossing families among $\mathcal{P}$ that: respect the extended sequence Π^+, no polygon in $\mathcal{Q}$ lies between two consecutive delimiters or in a bounded cell, and have as delimiters d_i and d_{i+1} for the bundles of classes K_i and K_{i+1}, respectively, it is always possible to replace the segments between and including d_i and d_{i+1} by C without decreasing the size of the crossing family, still respecting Π^+ and that no polygon in $\mathcal{Q}$ lies in a bounded cell.

In our algorithm we use Lemma 4 with $\mathcal{Q}$ equal to the contributing polygons of Π^+ (same as of Π). The motivation behind Condition iii and the condition involving $\mathcal{Q}$ in the second part of the statement is that no segments in a solution crossing family can define a region enclosing a contributing polygon. If such a region existed, there would be three segments defining a triangular region enclosing the polygon, which cannot be the case since any segment incident to an endpoint inside the region can only cross two of the segments.

▶ **Observation 5.** *Given a crossing family C whose set of contributing polygons is $\mathcal{P}$, no (three) segments of C define a (triangular) region enclosing a polygon in $\mathcal{P}$.*

Proof of Lemma 4. We first check whether Condition $\star$ holds in $\mathcal{O}(n)$ time. If Condition $\star$ does not hold and $\mathcal{Q}$ is the set of contributing polygons of Π, d_i and d_{i+1} cannot be delimiters for classes K_i and K_{i+1}.

Let $\mathcal{S}$ be the set of segments containing each segment $s \neq d_i, d_{i+1}$ such that $\{d_i, d_{i+1}, s\}$ are a crossing family that fulfills Conditions i–iii. Whether a segment lies in $\mathcal{S}$ can be tested in $\mathcal{O}(n)$ time per segment s and there are at most $\mathcal{O}(n^2)$ such segments. The segments in $\mathcal{S}$ form a circle graph, since their endpoints lie either on the boundaries of two disjoint polygons, or on the boundary of one polygon. Consequently, a maximum crossing family C^- consisting of segments in $\mathcal{S}$ is a clique in this graph and can be computed in time $\mathcal{O}(|\mathcal{S}|^2)$ [3]. Let C be the crossing family $C^- \cup \{d_i, d_{i+1}\}$, which we computed in $\mathcal{O}(n^4)$ time.

We now show that the algorithm above computes the crossing family C as per the lemma statement. First, note that by definition of $\mathcal{S}$, Conditions i and ii hold. In the definition of $\mathcal{S}$, we only checked that Condition iii holds for triangular regions bounded by segments d_i and d_{i+1}. We show that this implies that it also holds for any other triple of segments in C.

Assume for contradiction that a polygon $Q \in \mathcal{Q}$ lies in a triangular region bounded by three segments in C. By Condition $\star$, Q must lie either above the supporting lines of d_i and d_{i+1}, or below both. Being in such a bounded region implies that it lies below or above (respectively) the supporting line of a segment $x \in \mathcal{S}$. In both cases, Q lies in the triangle bounded by x, d_i, and d_{i+1}, which we checked is not the case when we included x in $\mathcal{S}$.

For the second part of the lemma, consider a crossing family X among $\mathcal{P}$ that respects Π^+, no polygon in $\mathcal{Q}$ lies between consecutive delimiters or in a bounded cell, and has delimiters d_i and d_{i+1} for the bundles of classes K_i and K_{i+1}, respectively. Let C' be the set of segments of X between d_i and d_{i+1}. For each segment $y \in C'$ it must hold, by definition, that $\{d_i, d_{i+1}, y\}$ is a crossing family satisfying Conditions i–iii. Thus, $C' \subseteq \mathcal{S}$ and replacing C' by C^- does not decrease the size of the crossing family.

We next show that all segments in $X \setminus C'$ cross all the segments in C^-; refer to Figure 3 (left). This is trivially true for d_i and d_{i+1}. Consider a different such segment z in X. By definition, z cannot lie between d_i and d_{i+1}. Moreover, none of its endpoints can lie between d_i and d_{i+1} since z must cross both d_i and d_{i+1}. Thus, by construction of $\mathcal{S}$, and concretely Condition iii, no endpoint of z can lie in a triangle bounded by d_i, d_{i+1}, and a segment in C^-. This implies that one endpoint of z lies below the supporting line of each segment in $\mathcal{S}$ and the other above. Since z crosses d_{i+1} and no segment in X can cross the boundary of a polygon, no endpoint of a segment in C^- can lie in a triangle bounded by z, d_i, and d_{i+1}.

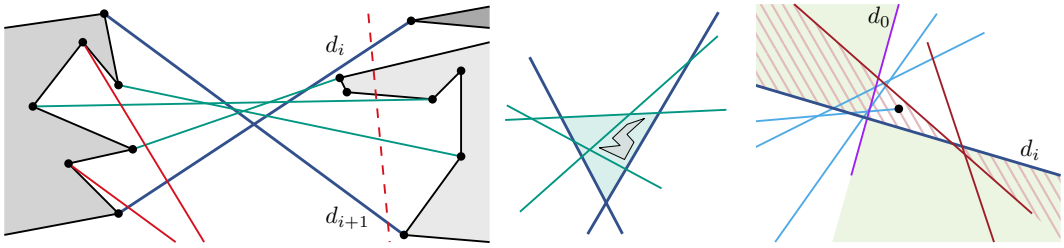


Figure 3 Illustration of the correctness of the FPT-algorithm for MaxCrossingFamily. Left: Segments in red cannot be in a crossing family with the blue consecutive delimiters. The dashed segment is invalid. Middle: a violation of Condition iii. Right: Red segments do not cross all blue delimiters. It can be detected considering the previous dark blue delimiter and the relevant polygons.

As the segments in C^-, by definition, lie between d_i and d_{i+1}, this means that every such segment has one endpoint on either side of z. We can therefore conclude that z crosses all segments in C^-.

By construction and Conditions i and ii, after replacing C' by C^- Π^+ is still respected. It remains to prove that no polygon $Q \in \mathcal{Q}$ lies in a bounded cell. By Condition $\star$, Q cannot lie in the horizontal double wedge of d_i, d_{i+1}. Assume w.l.o.g. that it lies in the top region of the vertical double wedge of d_i, d_{i+1}. If Q lies below the supporting line of a segment in C^-, then that segment together with d_i and d_{i+1} would contradict Condition iii; see Figure 3 (middle). But if it lies above the supporting lines of all segments in C^- then Q can only lie in a bounded cell if it already was in $X \setminus C'$, contradicting the assumption of X. ◀

In our algorithm, when computing $C_i(s,t)$ using Lemma 4, if Condition $\star$ does not hold we do not allow such a pair of delimiters (we can return an empty set and size $-\infty$). The next lemma establishes the correctness of our procedure.

▶ **Lemma 6.** *For a given set $\mathcal{P}$ of polygons, a sequence of classes Π^+, and a segment d_0, our algorithm computes a valid crossing family C among $\mathcal{P}$ that respects Π^+ and has as start delimiter d_0, or reports that no such family exists.*

Proof. Assume first that a solution exists, and let $d_0, d_1, \ldots d_\ell$ be its sequence of delimiters sorted by decreasing slope. Any two consecutive delimiters (together with d_0) must fulfill Condition $\star$ in Lemma 4, and thus our algorithm considered them and a largest crossing family between them, since all conditions are necessary. In particular, by Observation 5 Condition iii must hold for the set of contributing polygons. It might be that our algorithm picks a different sequence of delimiters or crossing family between them than the one in the solution assumed, but it will output an at-least-as-large crossing family.

In the other direction, assume that our algorithm produced a crossing family. We want to show that it is indeed a valid crossing family that respects Π^+ and has as start delimiter d_0. We construct the extended sequence of classes Π^+ from Π and d_0. The computed crossing family corresponds to a sequence of delimiters $d_0, d_1, \ldots d_\ell$. If those would indeed form a crossing family sorted by decreasing slope that respects Π^+ and has as start delimiter d_0, then by the second part of Lemma 4, the whole crossing family computed would be valid. The sequence of delimiters is sorted by decreasing slope and has as start delimiter d_0 by Condition $\star$ at each step. It also respects Π^+ by Condition ii.

It remains to prove that all the delimiters pairwise cross. Assume for contradiction that they do not, and let d_{i+1} be the first delimiter in the sequence that does not cross all the previous delimiters. Thus, $d_0, d_1, \ldots, d_i$ form a crossing family. By Condition $\star$, d_0, d_i, and

d_{i+1} are sorted by decreasing slope and there is no contributing polygon between d_i and d_{i+1}. In particular, the endpoints of d_{i+1} lie in the top and bottom region of the vertical double wedge of d_i and d_0, see Figure 3 (right). In contrast, delimiters $d_1, \ldots, d_{i-1}$ have their endpoints in the left and right region of the horizontal double wedge of d_i and d_0. Since we only consider valid segments as delimiters and there is no contributing polygon between d_i and d_{i+1}, there is no endpoint of the delimiters $d_1, \ldots, d_{i-1}$ enclosed by $\{d_0, d_i, d_{i+1}\}$. Thus, $d_0, d_1, \ldots, d_{i+1}$ form a crossing family, contradicting the assumption. ◀

▶ **Lemma 7.** *For a given set $\mathcal{P}$ of polygons, a sequence of classes Π, and a segment d_0, our algorithm computes a largest crossing family C among $\mathcal{P}$ that respects Π and has as start delimiter d_0.*

Proof. Let $\Pi_i^+ = (K_0 = K_1, \ldots, K_\ell)$ be the extended sequence of classes of Π and let $\mathcal{Q}$ be the set of contributing polygons of Π. We prove by induction on i that our algorithm computes (in step i) a largest crossing family C among $\mathcal{P}$ that respects the extended sequence $\Pi_i^+ = (K_0 = K_1, \ldots, K_i)$, has as start delimiter d_0, as end delimiter a given segment d_i (of class K_i) and the following property: ($\diamond$) there is no polygon in $\mathcal{Q}$ that lies between the extreme delimiters of C or in a triangular region bounded by three segments of C. Note that condition ($\diamond$) is necessary to obtain a crossing family fulfilling the conditions of the lemma.

The induction base trivially holds for $i = 0$ and by Lemma 4, case $i = 1$ also holds. Assume the induction hypothesis for step i, and assume for contradiction that in step $i+1$ there is a larger crossing family C' that respects Π_{i+1}^+, has as start delimiter d_0, as end delimiter d_{i+1}, and fulfills ($\diamond$). Let d_i' be the previous delimiter in C'; it might not coincide with the one chosen by our algorithm, but, for a similar argument as in Lemma 6, it is in the table at the start of step $i+1$. By the induction hypothesis, until d_i' the crossing family computed and tabulated by our algorithm cannot be smaller than the one of C' until d_i'. Moreover, by Lemma 4, between delimiters d_i' and d_{i+1} our algorithm computes a largest crossing family that can replace the one of C' while maintaining its properties (respects Π_{i+1}^+, has as start delimiter d_0, as end delimiter d_{i+1}, and fulfills ($\diamond$)). Thus, the crossing family that our algorithm computes for d_{i+1} in step $i+1$ is at least as large as C'; contradiction.

This implies that in the last step our algorithm computes a largest crossing family for each possible end delimiter. Lemma 6 ensures that we will succeed, and assuming we chose the end delimiter yielding a largest crossing family, the statement follows. ◀

We are now ready to establish our first algorithmic result:

▶ **Theorem 8.** MaxCrossingFamily *is fixed-parameter tractable with respect to the number of polygons k.*

Proof. Given a set $\mathcal{P}$ of polygons, consider a crossing family C' solving MaxCrossingFamily. In our branching step, we consider all possible sequences of classes and start delimiters. In particular, the sequence of classes Π and the start delimiter d_0 of C'. By Lemmas 6 and 7, Step 2 in our algorithm computes a largest valid crossing family C with $|C| \geq |C'|$ respecting Π and with start delimiter d_0, thus solving MaxCrossingFamily.

The algorithm runs in time $f(k) \cdot n^{\mathcal{O}(1)}$ where n is the number of vertices of $\mathcal{P}$. The precomputation step takes $\mathcal{O}(n^3)$ time. We run the polynomial (in n and k) algorithm in Step 2 $\mathcal{O}(n^2 k^{2(4k-3)})$ times. Without trying to optimize its running time, it comprises at most $4k-3$ steps corresponding to the classes, and in each step we consider at most $\mathcal{O}(n^4)$ pairs of delimiters (assuming that we choose the 4 endpoints of the delimiters from all n points). For each such pair, the procedure in Lemma 4 takes $\mathcal{O}(n^4)$ time. ◀

5 Finding independent families among k holes

As our final contribution, we provide an XP-algorithm for MaxIndependentFamilyWithHoles when parameterized by the number of holes – a result which contrasts the known NP-hardness of MaxIndependentFamily with a single polygon [21].

Our proof is based on decomposing sets of independent segments into what we refer to as *bunches*. On a high level, bunches are inclusion-maximal sets of segments that are enclosed by one segment and one connected hole boundary part, or by two segments and two connected hole boundary parts that do not enclose segments of the independent family with endpoints on other hole boundary parts. We will argue that each hypothetical solution can be decomposed into a number of bunches quadratic in the number of polygons, each of which is enclosed by at most two *delimiting* segments of the hypothetical solution and connected parts of at most two polygon boundaries. In XP-time we can branch on a choice of such delimiting segments and compute the other edges of each bunch as an independent set of a circle graph. We formalize below.

▶ **Definition 9.** *Given a family of polygons $\mathcal{P} = \{P_1, \ldots, P_k\}$ and a set $\mathcal{S}$ of pairwise independent segments between vertices of $\mathcal{P}$ that do not intersect P° for any $P \in \mathcal{P}$, a* bunch *B is a maximal subset of $\mathcal{S}$ that is enclosed by at most two elements b_1, b_2 of B and connected parts Q, Q' of the boundaries of at most two arbitrary polygons $P, P' \in \mathcal{P}$ such that the endpoints of each $b \in B$ lie in $Q \cup Q'$. b_1, b_2 are called the* delimiters *of B, and $b_1 \cup b_2 \cup Q \cup Q'$ is called the* boundary *of B.*

The next lemma, whose proof uses similar ideas as that of Lemma 3, allows us to branch on boundaries of bunches that partition a hypothetical target maximum independent family.

▶ **Lemma 10.** *Given a family of polygons $\mathcal{P} = \{P_1, \ldots, P_k\}$ and a set of pairwise independent segments $\mathcal{S}$ between vertices of $\mathcal{P}$ that do not intersect P° for any $P \in \mathcal{P}$, $\mathcal{S} = \dot{\bigcup}_{i \in [\ell]} B_i$ where each B_i is a bunch and $\ell \leq 4k^2$.*

Knowing the boundaries of bunches allows us to compute their other contained segments, or an independent family of at least as many segments in polynomial time. Note that here it is crucial that we are considering holes.

▶ **Theorem 11** ([18])**.** *Given a polygon, a maximal family of independent segments in its interior can be computed in polynomial time.*

The proofs for Lemma 10 and Theorem 11 can be found in the appendix. Combining the above, we obtain our final algorithmic result.

▶ **Theorem 12.** MaxIndependentFamilyWithHoles *is in* XP *parameterized by the number of holes.*

Proof. We can assume a partition into bunches of a maximum independent family of segments between the holes according to Lemma 10, and branch in $n^{2^{\mathcal{O}(k^2)}}$ (n denotes the overall number of polygon vertices) ways on the boundaries of the corresponding bunches: we branch on a set of $\mathcal{O}(k^2)$ pairwise independent delimiting segments, on $\mathcal{O}(k^k)$ pairings of them, and $\mathcal{O}(1)$ hole boundary parts to complete the boundary. Using Theorem 11, we can compute in polynomial time a maximum independent family of segments within each boundary. We return the union of all families computed in this way together with the branched delimiting segments obtained in this way with maximum size.

The described procedure obviously runs in XP-time parameterized by k. Further, the returned family of segments is indeed independent: Segments which are in the solution of the same invocation of Theorem 11 are independent by the correctness of Theorem 11. Segments which are in a solution of an invocation of Theorem 11 are independent from all branched on delimiting segments because they are prohibited from crossing the boundary for which they are in the solution and all delimiting segments lie on or beyond that boundary. Further, the boundary for which a segment was in the solution of an invocation of Theorem 11 separates this segment from all segments in solutions of other invocations of Theorem 11. Independence of branched-on delimiting segments is ensured at branching.

A maximum independent family is returned because in some branch, the delimiting segments will coincide with those that exist in a bunch partition of a maximum independent family according to Lemma 10, and any other segments in a bunch can be replaced by an arbitrary maximum independent family of segments in its boundary. ◀

6 Discussion and open questions

Our reductions establish the intractability of the considered problems even if the polygons are triangles. Nevertheless, the complexity of computing a maximum crossing family among a set of points remains as a prominent open question.

On the algorithmic front, we leave open whether MaxIndependentFamilyWithHoles is fixed-parameter tractable w.r.t. the number of holes. While it seems conceivable that the boundaries of bunches employed in the proof of Theorem 12 could instead be processed via dynamic programming, it is entirely unclear what processing order one should use. Another open question is whether MaxIndependentFamily is fixed-parameter tractable w.r.t. the number of polygons (as opposed to paraNP-hard) if all polygons are convex.

References

1 Hugo A. Akitaya, Matias Korman, Oliver Korten, Mikhail Rudoy, Diane L. Souvaine, and Csaba D. Tóth. Circumscribing polygons and polygonizations for disjoint line segments. *Discrete & Computational Geometry*, 68(1):218–254, 2022. doi:10.1007/S00454-021-00355-8.

2 Noga Alon, Sridhar Rajagopalan, and Subhash Suri. Long non-crossing configurations in the plane. *Fundamenta Informaticae*, 22(4):385–394, 1995. doi:10.3233/FI-1995-2245.

3 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.

4 Boris Aronov, Paul Erdös, Wayne Goddard, Daniel J. Kleitman, Michael Klugerman, János Pach, and Leonard J. Schulman. Crossing families. *Combinatorica*, 14(2):127–134, 1994. doi:10.1007/bf01215345.

5 Mikhail J. Atallah. A matching problem in the plane. *Journal of Computer and System Sciences*, 31(1):63–70, 1985. doi:10.1016/0022-0000(85)90065-0.

6 Peter Brass, William OJ Moser, and János Pach. *Research problems in discrete geometry*, volume 18. Springer, 2005. doi:10.1007/0-387-29929-7.

7 Sergio Cabello, Jean Cardinal, and Stefan Langerman. The clique problem in ray intersection graphs. *Discrete & Computational Geometry*, 50(3):771–783, 2013. doi:10.1007/s00454-013-9538-5.

8 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.

9 Harold Davenport and Andrzej Schinzel. A combinatorial problem connected with differential equations. *American Journal of Mathematics*, 87(3):684–694, 1965. doi:10.2307/2373068.

10 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications, 3rd Edition*. Springer, 2008. doi:10.1007/978-3-540-77974-2.

11 Adrian Dumitrescu and Csaba D. Tóth. Long non-crossing configurations in the plane. *Discrete & Computational Geometry*, 44(4):727–752, 2010. doi:10.1007/S00454-010-9277-9.

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

13 John Hershberger and Subhash Suri. Applications of a semi-dynamic convex hull algorithm. *BIT Numerical Mathematics*, 32(2):249–267, 1992. doi:10.1007/BF01994880.

14 Jan Kratochvíl and Jaroslav Nešetřil. Independent set and clique problems in intersection-defined classes of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 31(1):85–93, 1990. URL: http://dml.cz/dmlcz/106821.

15 Chi-Yuan Lo, Jirí Matousek, and William L. Steiger. Algorithms for ham-sandwich cuts. *Discrete & Computational Geometry*, 11:433–452, 1994. doi:10.1007/BF02574017.

16 Dániel Marx. Parameterized complexity of independence and domination on geometric graphs. In *Proceedings of the Second International Workshop on Parameterized and Exact Computation (IWPEC'06)*, volume 4169 of *LNCS*, pages 154–165. Springer, 2006. doi:10.1007/11847250_14.

17 Matthias Middendorf and Frank Pfeiffer. The max clique problem in classes of string-graphs. *Discrete Mathematics*, 108(1-3):365–372, 1992. doi:10.1016/0012-365X(92)90688-C.

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

19 Monroe Newborn and William O. J. Moser. Optimal crossing-free hamiltonian circuit drawings of k_n. *Journal of Combinatorial Theory, Series B*, 29(1):13–26, 1980. doi:10.1016/0095-8956(80)90041-6.

20 János Pach, Natan Rubin, and Gábor Tardos. Planar point sets determine many pairwise crossing segments. *Advances in Mathematics*, 386:107779, 2021. doi:10.1016/j.aim.2021.107779.

21 Alexander Pilz, Jonathan Rollin, Lena Schlipf, and André Schulz. Augmenting geometric graphs with matchings. In *Proceedings of the 28th International Symposium on Graph Drawing and Network Visualization (GD'20)*, volume 12590 of *Lecture Notes in Computer Science*, pages 490–504. Springer, 2020. doi:10.1007/978-3-030-68766-3_38.

22 David Rappaport. Computing simple circuits from a set of line segments is np-complete. *SIAM Journal on Computing*, 18(6):1128–1139, 1989. doi:10.1137/0218075.

23 David Rappaport, Hiroshi Imai, and Godfried T. Toussaint. On computing simple circuits on a set of line segments. In *Proceedings of the 2nd Annual Symposium on Computational Geometry (SCG'86)*, pages 52–60. ACM Press, 1986. doi:10.1145/10515.10521.

24 Micha Sharir, Adam Sheffer, and Emo Welzl. Counting plane graphs: Perfect matchings, spanning cycles, and kasteleyn's technique. *Journal of Combinatorial Theory, Series A*, 120(4):777–794, 2013. doi:10.1016/j.jcta.2013.01.002.

25 Micha Sharir and Emo Welzl. On the number of crossing-free matchings, cycles, and partitions. *SIAM Journal on Computing*, 36(3):695–720, 2006. doi:10.1137/050636036.

A Omitted proofs from Section 5

▶ **Lemma 10.** *Given a family of polygons $\mathcal{P} = \{P_1, \ldots, P_k\}$ and a set of pairwise independent segments $\mathcal{S}$ between vertices of $\mathcal{P}$ that do not intersect P° for any $P \in \mathcal{P}$, $\mathcal{S} = \dot{\bigcup}_{i\in[\ell]} B_i$ where each B_i is a bunch and $\ell \leq 4k^2$.*

Proof. We show the statement by induction on k.

For the base case, $k = 1$ and $\mathcal{S} = B$ with the delimiting segment of B being given by an arbitrary segment in $\mathcal{S}$ whose endpoints are consecutive in a traversal of the boundary $P \cap \mathcal{S}$ along the boundary of P (if no such segment exists then $\mathcal{S} = \emptyset$ and hence $\ell = 0$ is valid). The boundary of B is given by the above delimiting segment and the part of the boundary of P between its endpoints that contains other endpoints of segments in $\mathcal{S}$ (if there is no other segment in $\mathcal{S}$ we can choose between the two parts of the boundary of P between the endpoints of the delimiting segment arbitrarily).

For the inductive step, assume the statement to hold for independent families on holes $P_1, \ldots, P_{k-1}$. Consider all elements of $\mathcal{S}$ with an endpoint on P_k and an endpoint on P_i for any $i \in [k-1]$ and label them with $i \in [k-1]$ based on the containment of their other endpoint in P_i.

Along a traversal of the boundary of P_k, any distinct labels i, i' do not interleave (i.e., occur in the order i, i', i, i') as this would induce a crossing between two closed curves, one formed by segments with the same label i and a part of the boundary between their endpoints on P_k and P_i and the other formed by segments with the same label i' and a part of the boundary between their endpoints on P_k and $P_{i'}$, contradicting that $\mathcal{S}$ is independent or that P_i, $P_{i'}$, and P_k are pairwise non-intersecting. Now we remove all but the first and last segment or each subsequence of each maximal subsequence of the traversal which has one unique label. We call the resulting sequence σ.

A non-interleaving sequence over $[k-1]$ with at most two consecutive repetitions of each element has length at most $4k-2$ because without repetitions it is a Davenport-Schinzel sequence of order 2, and its length is therefore at most $2k-1$ [9]. Thus, the length of σ is at most $4k-2$.

By construction, a desired partition into bunches can be achieved by combining a desired partition of the subset of $\mathcal{S}$ that does not contain segments with endpoints on P_k and bunches whose delimiting segments are in σ; here, delimiting segments are associated to the same bunch if they are consecutive with the same label. The former partition exists by inductive hypothesis, and by inductive hypothesis we get an overall size bound of $4(k-1)^2+4k-2 \leq 4k^2$ as desired. ◀

▶ **Theorem 11** ([18]). *Given a polygon, a maximal family of independent segments in its interior can be computed in polynomial time.*

Proof. The set of segments in the interior of a polygon can be viewed as the vertex set of a circle graph in a natural way. Finding a maximum independent family among these segments then directly corresponds to finding a maximum independent set in the circle graph. The latter is known to be possible in polynomial time [18]. ◀

Testing Whether a Subgraph Is Convex or Isometric

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

Abstract

We consider the following two algorithmic problems: given a graph G and a subgraph $H \subseteq G$, decide whether H is an isometric or a geodesically convex subgraph of G. It is relatively easy to see that the problems can be solved by computing the distances between all pairs of vertices. We provide a conditional lower bound showing that, for sparse graphs with n vertices and $\Theta(n)$ edges, we cannot expect to solve the problem in $O(n^{2-\varepsilon})$ time for any constant $\varepsilon > 0$. We also show that the problem can be solved in subquadratic time for planar graphs and in near-linear time for graphs of bounded treewidth. Finally, we provide a near-linear time algorithm for the setting where G is a plane graph and H is defined by a few cycles in G.

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

Keywords and phrases convex subgraph, isometric subgraph, plane graph

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.12

Related Version *Full Version*: `https://arxiv.org/abs/2502.16193`

Funding Funded in part by the Slovenian Research and Innovation Agency (P1-0297, N1-0218, N1-0285). Funded in part by the European Union (ERC, KARST, project number 101071836). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Acknowledgements I am very grateful to Vladimir Batagelj and Sandi Klavžar for discussing the closest related work. I am also very grateful to a reviewer that provided several relevant references.

1 Introduction

Isometric and geodesically convex subgraphs[1] are two natural and fundamental concepts in the area of metric graph theory [3, 39]. Our objective in this paper is to study the algorithmic problem of recognizing whether a given subgraph is convex or isometric.

Setting. Let G be an undirected graph with abstract, positive edge lengths $\ell : E(G) \to \mathbb{R}_{>0}$. We call G the ***host graph*** and assume henceforth that it is *connected*. The ***length*** of a walk (or path) π in G, denoted by $\ell(\pi)$, is the sum of the lengths of its edges; thus $\ell(\pi) = \sum_{e \in E(\pi)} \ell(e)$, where the sum is with multiplicity. For any two vertices $x, y \in V(G)$, a ***shortest path*** from x to y is a minimum-length path connecting x and y, and the ***distance*** between x and y in G, denoted by $d_G(x, y)$, is the length of a shortest path from x to y.

[1] There are several distinct concepts of convexity in graphs [11, 39]. Since in this work we only talk about geodetic convexity, we will drop the adjective "geodesically".

© Sergio Cabello;
licensed under Creative Commons License CC-BY 4.0

For any two vertices x, y of G, the ***interval*** $I_G(x, y)$ is the subgraph of G defined by the union of all shortest paths from x to y. Formally

$$V(I_G(x,y)) = \{u \in V(G) \mid d_G(x,u) + d_G(u,y) = d_G(x,y)\}.$$
$$E(I_G(x,y)) = \{uv \in E(G) \mid d_G(x,u) + \ell(uv) + d_G(v,y) = d_G(x,y) \text{ or } d_G(x,v) + \ell(uv) + d_G(u,y) = d_G(x,y)\}.$$

A subgraph H of G is (geodesically) ***convex*** (in G) if and only if

$$\forall x, y \in V(H): \quad I_G(x,y) \subseteq H,$$

and it is ***isometric*** (in G) if and only if

$$\forall x, y \in V(H): \quad d_G(x,y) = d_H(x,y).$$

We consider the following algorithmic problems:

- Given a host graph G and a subgraph $H \subseteq G$, decide whether H is an isometric subgraph.
- Given a host graph G and a subgraph $H \subseteq G$, decide whether H is a convex subgraph.

In graph theory, the concepts of isometric and (geodesically) convex subgraphs are defined for the setting where $\ell(e) = 1$ for all $e \in E(G)$. In this unit-length setting, only induced subgraphs are considered because non-induced graphs cannot be isometric nor convex. In the general case with edge lengths, we may have $d_G(x,y) < \ell(xy)$ for some edges xy of G.

Henceforth, we use n and m for the number of vertices and edges of the host graph G.

Related work. Dourado et al. [16] provide an efficient algorithm to test convexity in the case of unit-length edges. Their approach can be easily adapted to graphs with arbitrary edge lengths, as follows. For each vertex x of the subgraph H, one computes the graph $H_x = \bigcup_{y \in V(H)} I_G(x,y)$. The subgraph H is convex in G if and only if H_x is contained in H for all $x \in V(H)$. The computation of H_x takes $O(m)$ time once we have a shortest-path tree in G from x, and testing whether $H_x \subseteq H$ also takes $O(m)$ time.

We conclude that testing whether a subgraph is convex can be done by computing a shortest-path tree from each vertex, the co-called all-pairs shortest-path (APSP) problem, followed by tests that take $O(nm)$ time. With a simple modification, the same time bound holds for testing whether the given subgraph is isometric. The running time to solve the APSP problem depends on the assumptions about the edge lengths:

- for unit edge lengths, the APSP problem is solved in $O(nm)$ using a simple BFS;
- when the edge lengths are natural numbers and multiplications take constant time, Thorup [46] solves the APSP problem in $O(nm)$ time;
- if we only perform additions and comparisons of edge lengths, then the algorithm of Pettie and Ramachandran [40] for APSP takes $O(nm\,\alpha(m,n))$ time. Using Dijkstra's algorithm with Fibonacci heaps [20], the running time is slightly worse, $O(n^2 \log n + nm)$.

We conclude that testing whether a subgraph is convex or isometric can be done in $\tilde{O}(nm)$ time in all situations.

Convex subgraphs play an important role in the theory of *median graphs* [12, 35, 36, 37], a class of graphs that has received substantial attention [3, 24, 31]. For median graphs, convex sets and so-called *gated sets* are the same concept; see for example [4, Lemma 3.3]. This can be used to test convexity in linear time: it suffices to check whether every vertex outside H has at most one neighbor in H. Imrich and Klavžar [27] provided a characterization of convex subgraphs in bipartite graphs in terms of Θ-classes. The characterization is for unit

edge lengths and leads to an algorithm taking $O(nm)$ time. They used it to obtain a fast algorithm to recognize median graphs. See also [23]. It was later shown that the recognition of median graphs is closely related to the detection of triangles in graphs [28]. If the host graph G is the Cartesian product of graphs, $G = G_1 \square \ldots \square G_k$, and the edges have unit length, a subgraph $H \subseteq G$ is convex if and only if $H = H_1 \square \ldots \square H_k$, where each H_i is a convex subgraph of G_i; see [24, Lemma 6.5]. Together with linear-time algorithms for factorization of graphs [29], this leads to an efficient algorithm for Cartesian products of substantially smaller graphs. Convex subgraphs are being considered also in more applied areas; see for example [21, 34, 44, 45].

A graph is a *partial cube* [15, 19, 48] if it is isomorphic to an isometric subgraph of some hypercube Q_d. Partial cubes enjoy a rich structure and appear in several different combinatorial settings; see for example [3, 24, 38]. Eppstein [18] provided an algorithm to test in $O(n^2)$ time whether a given graph with n vertices is a partial cube. Note that this setting is different to ours because we are not given a subgraph of Q_d, but have to test whether a given graph is isomorphic to an isometric subgraph of Q_d. In our setting, we are given a subgraph. The concept of isometric embeddings in the hypercube or other products of graphs has also been studied recently for weighted graphs; see [5, 42].

Our contribution. We first provide a series of characterizations of isometric (Section 2) and convex subgraphs (Section 3). Most notably, one of the characterizations uses a marked version of the of the so-called Wiener index, which we will define below. We think that the characterizations are new and of independent interest. Although proving each of our characterizations is not difficult, identifying them is closely related to the development of our algorithms.

Our first result regarding algorithms is the following conditional lower bound: if G has n vertices and $\Theta(n)$ edges, then there is no algorithm to test whether a given subgraph H of G is isometric or convex in $O(n^{2-\varepsilon})$ time, for any constant $\varepsilon > 0$, unless the Orthogonal Vectors Conjecture (OVC) and the Strong Exponential Time Hypothesis (SETH) fail. This means that for general graphs we cannot expect to test whether a subgraph is convex or isometric in, say, $O(nm^{0.999})$ time, unless some standard assumptions are wrong. The lower bound holds for graphs with unit edge lengths and implies that the algorithms mentioned above, with running times $\tilde{O}(nm)$, are asymptotically optimal up to polynomial factors $\Theta(n^{\varepsilon})$ for any $\varepsilon > 0$. Our reduction is via the diameter problem. We review OVC and SETH in Section 4, where we also provide our reduction via the diameter problem.

The characterization using the marked version of the Wiener index can be combined with adaptations of known algorithms to obtain the following results about testing whether a given subgraph is isometric or convex:

- If the host graph is planar graph, the test can be performed in $\tilde{O}(n^{5/3})$ time.
- If the host graph has treewidth bounded by a constant $k \geq 3$, the test can be performed in $O(n \log^{k-1} n)$ time; the constant in the O-notation depends on k. When the treewidth k is considered a parameter, the test can be performed in $n^{1+\varepsilon} 2^{O(k)}$, for any constant $\varepsilon > 0$; the constant in the O-notation depends on the choice of ε.

Finally, in Section 6 we consider the following setting. The host graph G is a ***plane graph***, that is, a planar graph with a fixed embedding in the plane. Let C be a cycle in G and let $H = \mathrm{int}(G, C)$ be the subgraph of G bounded by C and including C; see Figure 1 left. We show how to decide whether H is an isometric or a convex subgraph of G in $O(n \log n)$ time. The result can be extended to subgraphs of G defined by a constant number of cycles; see the right of Figure 1 for an example and Section 6 for more precise definitions. Compared

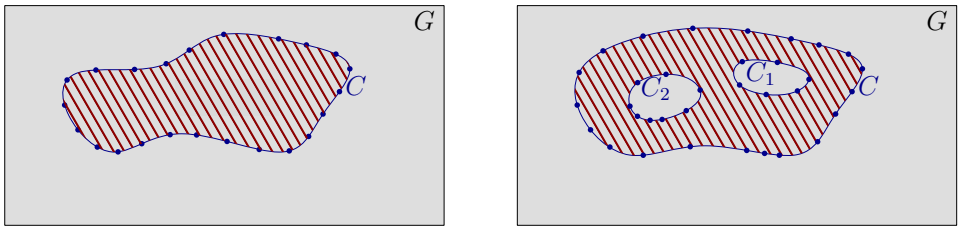


Figure 1 Schematic view of the subgraph $H = \mathrm{int}(G, C)$ defined by the cycle C (left) and the subgraph $\mathrm{holes}(G, C, \{C_1, C_2\})$ defined by a few cycles (right). They are both marked with stripes.

to the other results presented in the paper, this result is interesting because it uses different characterizations and a different algorithmic paradigm. We use multiple-source shortest-path (MSSP) algorithms for planar graphs [9, 32] and combine the distances in the interior and the exterior of C, as the source of the shortest-path tree moves along the cycle C. As far as we know, this is the first application of MSSP where we maintain a combination of the interior and the exterior distances.

Notation and basic concepts. Through our discussion we will assume that the host graph G is fixed and in the notation we will often drop the dependency on G.

For each subgraph H of G, the ***boundary*** of H (in G) is the set ∂H of vertices of H that are incident to some edge outside H. Thus, $\partial H = \{v \in V(H) \mid \exists uv \in E(G) \setminus E(H)\}$. Note that each path in G that has edges from $E(H)$ and edges from $E(G) \setminus E(H)$ must have some vertex of ∂H.

For each subgraph H of G, let H_{out} be the ***complement*** of H in G, defined as the subgraph with edges not contained in H. Thus, $H_{\mathrm{out}} = (V(G), E(G) \setminus E(H))$. The graph H_{out} may have isolated vertices. For distances in H_{out} we write $d_{H_{\mathrm{out}}}(\cdot, \cdot)$.

We will often use without explicit mention that for any subgraph H of G and any vertices $x, y \in V(H)$ we have $d_G(x, y) \le d_H(x, y)$ and $d_G(x, y) \le d_{H_{\mathrm{out}}}(x, y)$.

The ***Wiener index with marked vertices*** for a graph G with respect to $U \subseteq V(G)$ is $\sum_{x,y \in U} d_G(x, y)$. The classical Wiener index is the case of $U = V(G)$. For unweighted graphs, this is a very similar to the vertex-weighted medians considered by Bandelt and Chepoi [2] and it is a particular case of the vertex-weighted Wiener index considered by Bénéteau et al. [4] for median graphs. In their weighted version, the marked vertices have weight 1 and the vertices to be ignored in the sum have weight 0.

2 Characterizing isometric subgraphs

In this section we provide different criteria to test when a subgraph H of G is isometric.

▶ **Lemma 1.** *Let G be a connected graph and let H be a subgraph of G. Then H is an isometric subgraph of G if and only if*

$$\forall x, y \in \partial H : \quad d_H(x, y) \le d_{H_{\mathrm{out}}}(x, y).$$

The next two characterizations use the Wiener index with marked vertices.

▶ **Lemma 2.** *Let G be a connected graph and let H be a subgraph of G. Then H is an isometric subgraph of G if and only if*

$$\sum_{x,y \in V(H)} d_G(x, y) = \sum_{x,y \in V(H)} d_H(x, y).$$

In fact, it suffices to add the distances between vertices on the boundary of H, or any superset of them.

▶ **Lemma 3.** *Let G be a connected graph, let H be a subgraph of G and let U be such that $\partial H \subseteq U \subseteq V(H)$. Then H is an isometric subgraph of G if and only if*

$$\sum_{x,y\in U} d_H(x,y) = \sum_{x,y\in U} d_G(x,y).$$

3 Characterizing convex subgraphs

Here we provide criteria to identify when a subgraph H of G is convex. The structure of the statements and the proofs is parallel to the isometric case considered in Section 2.

▶ **Lemma 4.** *Let G be a connected graph and let H be a subgraph of G. Then H is a convex subgraph of G if and only if*

$$\forall x,y \in \partial H,\ x \neq y: \quad d_H(x,y) < d_{H_{\text{out}}}(x,y).$$

For the following characterizations, we are going to decrease slightly the length of some edges of G so that no new shortest paths are created. For this, we define

$$\varepsilon = \varepsilon(G,\ell) = \frac{\min\{\ell(\pi) - \ell(\pi') \mid \pi,\pi' \text{ paths in } G \text{ with } \ell(\pi) > \ell(\pi')\}}{n}.$$

First note that we can indeed take the minimum because there is a finite number of paths in G. Moreover, for each edge $e \in E(G)$, we have $\varepsilon \leq \ell(e)/n$ because e is a path in G, as well as the empty path. For a subgraph H of G and a δ with $0 < \delta \leq \varepsilon$, we define the edge lengths $\hat{\ell}$ by shortening the edges *not* in H by δ. Thus

$$\hat{\ell}(e) = \begin{cases} \ell(e) & \text{if } e \in E(H), \\ \ell(e) - \delta & \text{if } e \in E(G) \setminus E(H). \end{cases}$$

Note that $\hat{\ell}(e) > 0$ for all $e \in E(G)$ because $\delta \leq \varepsilon < \ell(e)$. The lengths $\hat{\ell}$ depend on H and δ, but we drop this dependency to avoid cluttering the notation. Let $\hat{d}_G(\cdot,\cdot)$ denote the distance in G with respect to the edge lengths $\hat{\ell}$.

▶ **Lemma 5.** *If G is connected and $0 < \delta \leq \varepsilon$, each shortest path in G with respect to $\hat{\ell}$ is a shortest path in G with respect to ℓ. (The converse is not necessarily true.)*

▶ **Lemma 6.** *Let G be a connected graph, let H be a subgraph of G, let δ be a real value such that $0 < \delta \leq \varepsilon$, and let $\hat{d}_G$ be the corresponding perturbed distance. Then H is a convex subgraph of G if and only if*

$$\sum_{x,y\in V(H)} \hat{d}_G(x,y) = \sum_{x,y\in V(H)} d_H(x,y).$$

Proof. Assume first that H is a convex subgraph of G. Consider any two vertices x, y of H and let $\pi_{x,y}$ be a shortest path in G from x to y with respect to $\hat{\ell}$. Therefore $\hat{d}_G(x,y) = \hat{\ell}(\pi_{x,y})$. Because of Lemma 5, the path $\pi_{x,y}$ is a shortest path in G with respect to ℓ, which means that $\ell(\pi_{x,y}) = d_G(x,y)$. Because H is convex, the path $\pi_{x,y}$ is contained in H, and since the edges of H have the same length in ℓ and in $\hat{\ell}$, we have $\hat{\ell}(\pi_{x,y}) = \ell(\pi_{x,y})$. Putting the forthcoming equalities together and using that H is a convex (and thus isometric) subgraph of G, we obtain

$$\hat{d}_G(x,y) = \hat{\ell}(\pi_{x,y}) = \ell(\pi_{x,y}) = d_G(x,y) = d_H(x,y).$$

Since this equality holds for each $x, y \in V(H)$, we obtain

$$\sum_{x,y \in V(H)} \hat{d}_G(x,y) = \sum_{x,y \in V(H)} d_H(x,y).$$

To show the other direction, note first that $\hat{d}_G(x,y) \leq d_G(x,y) \leq d_H(x,y)$ for all $x, y \in V(H)$ because $\hat{\ell}(e) \leq \ell(e)$ for all edges $e \in E(G)$ (for the first inequality) and H is a subgraph of G (for the second inequality). Assume now that H is not a convex subgraph of G. This means that there exists vertices $x_0, y_0 \in V(H)$ and a x_0-to-y_0 shortest path π_0 that is not contained in H. Since π_0 has some edge in $E(G) \setminus E(H)$, we have $\hat{\ell}(\pi_0) < \ell(\pi_0)$ and therefore $\hat{d}_G(x_0,y_0) < d_G(x_0,y_0) \leq d_H(x_0,y_0)$. Therefore

$$\begin{aligned}\sum_{x,y \in V(H)} \hat{d}_G(x,y) &= 2 \cdot \hat{d}_G(x_0,y_0) + \sum_{\substack{x,\,y \,\in\, V(H) \\ \{x,y\} \neq \{x_0,y_0\}}} \hat{d}_G(x,y) \\ &< 2 \cdot d_H(x_0,y_0) + \sum_{\substack{x,\,y \,\in\, V(H) \\ \{x,y\} \neq \{x_0,y_0\}}} d_H(x,y) = \sum_{x,y \in V(H)} d_H(x,y). \end{aligned}$$

◀

▶ **Lemma 7.** *Let G be a connected graph, let H be a subgraph of G, let U be such that $\partial H \subseteq U \subseteq V(H)$, let δ be a real value such that $0 < \delta \leq \varepsilon$, and let $\hat{d}_G$ be the corresponding perturbed distance. Then H is a convex subgraph of G if and only if*

$$\sum_{x,y \in U} d_H(x,y) = \sum_{x,y \in U} \hat{d}_G(x,y).$$

Proof. In the first part of the proof of Lemma 6 we have seen that, if H is a convex subgraph of G, then for all $x, y \in V(H)$ we have $\hat{d}_G(x,y) = d_H(x,y)$. The equality of the two sums follows.

For the other direction, we start assuming that H is not a convex subgraph of G. If H is not connected, then the left sum is ∞ while the right side is bounded, and therefore they are distinct. We continue assuming that H is connected. By Lemma 4, there exist distinct vertices $x_0, y_0 \in \partial H$ such that $d_H(x_0,y_0) \geq d_{H_{\text{out}}}(x_0,y_0)$. Let π_0 be a shortest x_0-to-y_0 path in H and let π_0' be a shortest x_0-to-y_0 path in H_{out}. Note that π_0' indeed exists because $d_{H_{\text{out}}}(x_0,y_0)$ is bounded. We then have $\ell(\pi_0) = d_H(x_0,y_0) \geq d_{H_{\text{out}}}(x_0,y_0) = \ell(\pi_0')$. Since π_0' has edges outside $E(H)$, we have $\hat{\ell}(\pi_0') < \ell(\pi_0')$, and therefore

$$\hat{d}_G(x_0,y_0) \leq \hat{\ell}(\pi_0') < \ell(\pi_0') \leq \ell(\pi_0) = d_H(x_0,y_0).$$

In summary, $\hat{d}_G(x_0,y_0) < d_H(x_0,y_0)$ for some $x_0, y_0 \in \partial H$. In general, for all $x, y \in V(H)$ we have $\hat{d}_G(x,y) \leq d_G(x,y) \leq d_H(x,y)$ because $\hat{\ell}(e) \leq \ell(e)$ for all edges $e \in E(G)$ (for the first inequality) and H is a subgraph of G (for the second inequality). From this it follows that for U such that $\partial H \subseteq U \subseteq V(H)$ we have

$$\sum_{x,y \in U} \hat{d}_G(x,y) < \sum_{x,y \in U} d_H(x,y).$$

◀

4 Conditional lower bound

Our conditional lower bound is based on the following assumptions, where $\langle \cdot, \cdot \rangle$ is the inner product and the diameter of a graph G is $\operatorname{diam}(G) = \max_{x,y \in V(G)} d_G(x,y)$.

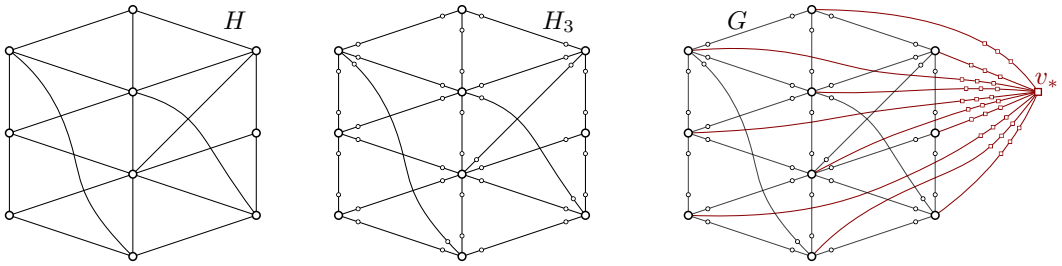


Figure 2 Example for the construction of Theorem 9. Left: a graph H. Center: the graph H_3. Right: the graph G with H_3 as a subgraph.

▶ **Orthogonal Vectors Conjecture** (OVC, [47]). *For every $\varepsilon > 0$ there is a $c \geq 1$ such that the following problem cannot be solved in $O(n^{2-\varepsilon})$ time: given n binary vectors $v_1, \ldots, v_n \in \{0,1\}^d$, where $d = \lceil c \log n \rceil$, are there $i, j \in [n]$ such $\langle v_i, v_j \rangle = 0$?*

▶ **Strong Exponential Time Hypothesis** (SETH, [25, 26]). *For every $\varepsilon > 0$ there exists an integer $k \geq 3$ such that k-SAT with n variables cannot be solved in $O(2^{(1-\varepsilon)n})$ time.*

Williams [47] posed OVC and showed that it is implied by SETH. One of the interesting consequences of OVC, and therefore SETH, is the following result.

▶ **Theorem 8** (Consequence of Theorem 9 in Roditty and Vassilevska Williams [41]). *If one can distinguish in $O(n^{2-\varepsilon})$ time for some constant $\varepsilon > 0$ between diameter 2 and 3 in undirected graphs with n vertices and $O(n)$ edges, all of unit length, then OVC and SETH fail.*

▶ **Theorem 9.** *Assume that, for some $\varepsilon > 0$, there is an algorithm that in $O(n^{2-\varepsilon})$ time solves the following problem: given an undirected graph G with n vertices and $O(n)$ edges, all with unit length, and given a subgraph H of G, decide whether H is an isometric subgraph of G. Then OVC and SETH fail.*

The same result holds for deciding if H is a convex subgraph of G.

Proof. We will reduce from the diameter problem considered in Theorem 8. Let H be a graph with $O(n)$ vertices and $O(n)$ edges, all edges of unit length, for which we want to distinguish whether it has diameter 2 or 3. Consider the graph $G = G(H)$ defined as follows; see Figure 2 for an example. We first construct the graph H_3 from H by subdividing each edge of H twice. Then, we add a new vertex v_* and connect it to each vertex of $V(H)$ with a 4-edge path using new interior vertices. Let $G = G(H)$ be the resulting graph. Note that H_3 is a subgraph of G and $\partial H_3 = V(H)$.

We assign unit length to each edge of G. For all $x, y \in V(H)$, we have $d_{H_3}(x,y) = 3 \cdot d_H(x,y)$ and $d_G(x,y) = \min\{3 \cdot d_H(x,y), 8\}$, where the 8 comes from the path in G through v_*.

We now show that H_3 is an isometric subgraph of G if and only if H has diameter 2. If H has diameter 2, then for all $x, y \in V(H)$ we have

$$d_G(x,y) = \min\{3 \cdot d_H(x,y), 8\} = 3 \cdot d_H(x,y) = d_{H_3}(x,y).$$

Since $\partial H_3 = V(H)$, we have $\sum_{x,y \in \partial H_3} d_G(x,y) = \sum_{x,y \in \partial H_3} d_{H_3}(x,y)$ and H_3 is an isometric subgraph of G because of Lemma 3. If H has diameter 3, then there is some $x_0, y_0 \in V(H)$ with $d_H(x_0,y_0) = 3$, and therefore $d_{H_3}(x_0,y_0) = 3 \cdot d_H(x_0,y_0) = 9 > 8 \geq d_G(x_0,y_0)$. It follows that H_3 is not an isometric subgraph of G.

We now show that H_3 is a convex subgraph of G if and only if H has diameter 2. Let $\hat{d}$ denote the distance in G when we decrease the length of the edges of $E(G) \setminus E(H_3)$ by a sufficiently small value $\delta > 0$. For concreteness, we can take $\delta < 1/10$ in our discussion. Then, for all $x, y \in V(H)$, we have $d_{H_3}(x, y) = 3 \cdot d_H(x, y)$ and $\hat{d}_G(x, y) = \min\{3 \cdot d_H(x, y), 8 - 8\delta\}$.

If H has diameter 2, then, for all $x, y \in V(H)$, we have

$$\hat{d}_G(x, y) = \min\{3 \cdot d_H(x, y), 8 - 8\delta\} = 3 \cdot d_H(x, y) = d_{H_3}(x, y),$$

where we have used that $3 \cdot d_H(x, y) \leq 6$ and $8\delta < 1$. Since $\partial H_3 = V(H)$, we have $\sum_{x,y \in \partial H_3} \hat{d}_G(x, y) = \sum_{x,y \in \partial H_3} d_{H_3}(x, y)$, and H_3 is a convex subgraph of G because of Lemma 7.

If H has diameter 3, then we have already seen that H_3 is not an isometric subgraph of G, and thus cannot be a convex subgraph.

If H has $O(n)$ vertices and $O(n)$ edges, then G has $|V(H)|+2\cdot|E(H)|+3\cdot|V(H)|+1 = O(n)$ vertices and $3 \cdot |E(H)| + 4 \cdot |V(H)| = O(n)$ edges. The construction of G and H_3 from H takes linear time. It follows that, if we could decide whether H_3 is isometric (or convex) in $O(n^{2-\varepsilon})$ time, for some $\varepsilon > 0$, we could decide whether H has diameter 2 or 3 in $O(n^{2-\varepsilon})$ time, and therefore OVC and SETH would be false by Theorem 8. ◀

5 Subquadratic algorithms for special classes of graphs

In this section we provide algorithms to test whether a subgraph is isometric or convex, when the host graph is planar or has small treewidth. The idea is to use the criteria of Lemmas 2 and 6 together with modifications of previous algorithms to compute the classical Wiener index.

5.1 Shortening some edges a bit

To test whether $H \subseteq G$ is convex using Lemma 6 we have to shorten the edges of $E(G) \setminus E(H)$ by a small enough amount δ, where $0 < \delta \leq \varepsilon(G, \ell)$. We do not know how to compute $\varepsilon(G, \ell)$ efficiently, or some other value that could replace it. We solve this by perturbing the edge lengths infinitesimally, as follows. (For unit length edges this is easy by taking $\varepsilon = 1/2n$.)

We define a new, composite length ℓ_2 for G using 2-tuples. The intuition is that a pair $(a, b) \in \mathbb{R}_{\geq 0} \times \mathbb{Z}$ represents the real value $a + b \cdot \delta$ for an infinitesimal $\delta > 0$. This means that we can compare two lengths (a, b) and (a', b') using the lexicographic comparison: (a, b) is smaller than (a', b'), denoted by $(a, b) \prec \ell(a', b')$, if and only if $a < a'$ or if $a = a'$ and $b < b'$. This lexicographic comparison is compatible with the test of whether $a + b \cdot \delta < a' + b' \cdot \delta$ for an infinitesimal $\delta > 0$.

To carry the perturbation, we define a new edge length ℓ_2 in the following way:

$$\forall e \in E(G): \qquad \ell_2(e) = \begin{cases} (\ell(e), 0) & \text{if } e \in E(H), \\ (\ell(e), -1) & \text{if } e \in E(G) \setminus E(H). \end{cases}$$

The length of a walk (or path) π is defined as the vector-sum of the the 2-tuples $\ell_2(e)$ over $e \in E(\pi)$, with multiplicity. Therefore, we have

$$\ell_2(\pi) = \big(\ell(\pi),\ -|E(\pi) \setminus E(H)|\big),$$

which represents the value $\ell(\pi) - \delta \cdot |E(\pi) \setminus E(H)|$ for an infinitesimal $\delta > 0$.

Classical algorithms for computing shortest paths can compute distances using this new edge lengths. In general, any algorithm that manipulates the edge lengths with additions and makes comparisons of the lengths of paths can be adapted to this 2-tuple distances without

an asymptotic increase in the running time. (For the sake of comparison, it is not clear how to adapt efficiently an algorithm that would multiply edge lengths because powers of δ would start showing up and a constant-size tuple would not suffice anymore.)

5.2 Graphs of small treewidth

Let G be a graph of treewidth $\mathrm{tw}(G)$, possibly with edge lengths, and let H be a subgraph of G. It is well-known that H has treewidth at most $\mathrm{tw}(G)$.

Cabello and Knauer [10] showed that, for each constant $k \geq 3$, the Wiener index of graphs of treewidth at most k can be computed in $O(n \log^{k-1} n)$ time; the constant hidden in the O-notation depends on k. Bringmann, Husfeldt and Magnusson [6] carried out the analysis making the dependency on the treewidth explicit and showed that the Wiener index can be computed in $n^{1+\varepsilon} 2^{O(\mathrm{tw}(G))}$, for each constant $\varepsilon > 0$; the constant hidden in the O-notation depends on ε.

Both papers use data structures for orthogonal range searching. The vertices of G are used to define a point set and to define ranges. It is easy to see that those results can be adapted to compute the sum $\sum_{x,y \in U} d_G(x,y)$ for any given $U \subseteq V(G)$. Indeed, it suffices to define the points and the ranges only for the vertices of U. This is explicitly mentioned in [8, Section 5]. The running times of the results are not affected by this. (Actually, if $\log |U| = o(\log n)$, the asymptotic running times can be slightly improved, but we omit this in our worst-case analysis.)

Next, we note that the algorithms in [6, 10] are manipulating and comparing sums of the edge lengths; they do not make other operations with the edge lengths. This means that they can handle edge lengths defined by a 2-tuple, as discussed in Section 5.1. We conclude that those algorithms can be used, without changing the asymptotic running time, to compute

$$\sum_{x,y \in V(H)} d_G(x,y), \qquad \sum_{x,y \in V(H)} \hat{d}_G(x,y), \qquad \sum_{x,y \in V(H)} d_H(x,y),$$

where $\hat{d}_G(x,y)$ is defined via 2-tuples to model a shortening of the edges of $E(G) \setminus E(H)$ by an infinitesimal $\delta > 0$. We can then compare them, taking into account that the ℓ_2-length (a,b) is equal to the ℓ-length a if and only if $b = 0$. We have shown the following.

▶ **Theorem 10.** *Let $k \geq 3$ be a constant. Given a graph G with n vertices and treewidth at most k, we can test whether a given subgraph of G is convex or isometric in $O(n \log^{k-1} n)$ time. The constant hidden in the O-notation depends on k.*

We can decide whether a given subgraph of G is convex or isometric in $n^{1+\varepsilon} 2^{O(\mathrm{tw}(G))}$ time, for each constant $\varepsilon > 0$. The constant hidden in the O-notation depends on ε.

5.3 Planar graphs

Cabello [7] gave a randomized algorithm to compute the Wiener index of a planar graph G in $\tilde{O}(n^{11/6})$ expected time. This was improved by Gawrychowski et al. [22], who provided a deterministic algorithm taking $\tilde{O}(n^{5/3})$ time. Both algorithms have the property that they can compute $\sum_{x,y \in U} d_G(x,y)$ for any given $U \subseteq V(G)$. Also, both algorithms only use the edge lengths via sums and comparisons of values. Therefore, they can also be used to obtain

$$\sum_{x,y \in V(H)} d_G(x,y), \qquad \sum_{x,y \in V(H)} \hat{d}_G(x,y), \qquad \sum_{x,y \in V(H)} d_H(x,y),$$

where $\hat{d}_G(x,y)$ is defined via 2-tuples to model a shortening by an infinitesimal $\delta > 0$. This shows the following.

▶ **Theorem 11.** *Given a planar graph G with n vertices, we can test whether a given subgraph of G is convex or isometric in $\tilde{O}(n^{5/3})$ time.*

5.4 Discussion

We next present classes of graphs illustrating two limitations of our technique based on the Wiener index with marked vertices. One limitation is that for testing convexity we have to consider edge-weights, even if the original graph is unweighted; the other limitation is that subgraphs may loose some useful properties enojoyed by the host graph.

Let us consider the class of K_h-minor-free graphs, where K_h denotes the complete graph on h vertices. Ducoffe et al. [17] showed how to compute the diameter of such graphs in subquadratic time for the unweighted version, but left open the question of computing the Wiener index in subquadratic time. Recent works by Le, Wulff-Nilsen, Karczmarz and Zheng [30, 33] have improved the dependency on h and their new techniques also work for the Wiener index, but still only for the case of unweighted graphs. The currently best result [30, Theorem 1.3] computes the Wiener index in *unweighted* graphs without a K_h-minor in $\tilde{O}(n^{2-1/(3h-2)})$ time. The technique can be easily adapted to compute the Wiener index with marked vertices. Since a subraph of a K_h-minor-free graph is obviously also K_h-minor-free, we can use this algorithm to test whether a given subgraph is isometric via Lemma 2. We summarize.

▶ **Theorem 12.** *Given an unweighted K_h-minor-free graph G with n vertices, we can test whether a given subgraph of G is isometric in $\tilde{O}(n^{2-1/(3h-2)})$ time.*

Since currently we do not know how to compute in subqudratic time the Wiener index of K_h-minor-free graph, we cannot make the modification of edge-lengths needed to test whether a subgraph is convex via Lemma 6. Therefore, we cannot make an analogous claim regarding testing convexity, even for unweighted graphs.

Let us turn our attention now to unweighted median graphs. Bénéteau et al. [4] show that the Wiener index of median graphs can be computed in linear time. In fact, they solve a vertex-weighted variant that generalizes the Wiener index with marked vertices. Consider the problem of testing whether a given subgraph H of a median graph G is isometric. We can apply their algorithm to compute $\sum_{x,y\in V(H)} d_G(x,y)$ in linear time. However, it is not clear (at least to the author) how to compute in subquadratic time $\sum_{x,y\in V(H)} d_H(x,y)$, even if we assume that H is an isometric subgraph of G, and thus a partial cube. When we consider a subgraph of a median graph, we loose many of its structural properties, even if the subgraph is isometric. Similarly, for testing convexity using the Wiener index, we would need to consider the perturbed weights, which in general does not go well with many of the structural properties of median graphs. (In contrast, as mentioned in the introduction, testing whether a subgraph of a median graph is convex amounts to testing whether it is gated, a property that can be tested in linear time.)

6 Subgraphs of plane graphs defined by cycles

Let G be a plane graph, that is, a planar graph with a fixed embedding in the plane. For each cycle C of G, let γ_C be the Jordan curve defined by C. Such a cycle C defines naturally the ***interior graph***, denoted by $\mathrm{int}(G,C)$, which is the subgraph of G contained in the closure of the bounded set of $\mathbb{R}^2 \setminus \gamma_C$. Thus, C is contained in $\mathrm{int}(G,C)$. In this section we consider the problem of testing whether $H = \mathrm{int}(G,C)$ is a convex or isometric subgraph of G. In H we assume the embedding inherited from G and then C is a facial walk of H.

6.1 Data structure for sequences of numbers

We will use a data structure to store a sequence $a_1, \ldots, a_k$ of k values in $\mathbb{R}$, where each value is initialized arbitrarily, and supporting the following operations:

- $\textsc{Init}(a_1^0, \ldots, a_k^0)$: initializes the data structure, setting $a_i = a_i^0$ for each $i \in [k]$.
- $\textsc{Min}()$ returns $\min\{a_1, \ldots, a_k\}$.
- $\textsc{Add}(i, j, \Delta)$ adds the value $\Delta \in \mathbb{R}$ to $a_i, \ldots, a_j$, where $i \leq j$.

An efficient approach is to use a *static* balanced binary search tree storing k elements with keys $[k] = \{1, \ldots, k\}$, such that the node with key i stores a_i. The tree is then augmented with data at the nodes to represent values that have to be added to the whole subtree. See for example Choi, Cabello and Ahn [13] for a comprehensive description. Another alternative, which is a bit overkilling, is to use dynamic trees, such as cut-link trees [43] or top trees [1]; in this case we maintain a path on k nodes, where the ith node stores the value a_i. These data structures allow to add values and to query for the minimum value in a subpath, which in our case corresponds to a contiguous subsequence. We summarize for later reference.

▶ **Lemma 13.** *There is a data structure to maintain a sequence $a_1, \ldots, a_k$ of k numbers that supports the operations $\textsc{Min}$, and $\textsc{Add}$ in $O(\log k)$ time per operation. The initialization of the data structure, $\textsc{Init}$, takes $O(k)$ time.*

6.2 Encoding distances within a face

Let C be a facial cycle in a plane graph G with n vertices, and let $x_1, \ldots, x_k$ be the vertices as we walk along C, starting from an arbitrary vertex x_1 of $V(C)$. For each $i \in [k]$, let D_i be the sequence of numbers telling the distances from x_i to all other vertices; thus $D_i = \big(d_G(x_i, x_j)\big)_{j \in [k]}$.

In a seminal work, Klein [32] showed that in $O(n \log n)$ time one can maintain a shortest-path tree of G rooted at a vertex of C as the root moves along C. An alternative point of view based on parametric shortest-path trees was given by Cabello, Chambers and Erickson [9], and a new approach based on divide-and-conquer has been given by Das et al. [14].

The key insight is that when the root of the shortest path tree moves along the boundary of a face, each directed arc of G appears in the shortest path tree along a contiguous subsequence of roots along the facial walk. In other words, with the move of the root through the facial walk, each directed edge enters and exits the shortest path tree at most once.

These results are the basis for the following property.

▶ **Lemma 14.** *In $O(n \log n)$ time we can compute sequences $\Lambda_2, \ldots, \Lambda_k$ of operations with the following properties*

- *for $i = 2, \ldots, k$, each Λ_i is a sequence of $|\Lambda_i|$ operations of the type $\textsc{Add}$ in a sequence $a_1, \ldots, a_k$ of numbers;*
- *for $i = 2, \ldots, k$, the sequence D_i of distances from x_i is obtained from the sequence D_{i-1} performing the operations given in Λ_i;*
- *the sequences Λ_i together have a linear number of operations; that is $\sum_i |\Lambda_i| = O(n)$.*

Proof. Let T be an arbitrary spanning tree of G rooted at a vertex r, let e be an edge of T, and let $T(r, e)$ be the subtree of $T - e$ that does not contain the root r. Because G is plane and C defines a face of G, the vertices of $V(C)$ that appear in $T(r, e)$ form a continuous subsequence of C. See Figure 3. Note that the subsequence may be empty, meaning that $V(C) \cap V(T(r, e))$ is empty. Let $s(T, r, e)$ and $t(T, r, e)$ be the start and the end of the subsequence, if it is non-empty, such that the vertices $\{x_{s(T,r,e)}, x_{s(T,r,e)+1}, \ldots, x_{t(T,r,e)}\}$ (indices modulo k) are $V(C) \cap V(T(r, e))$.

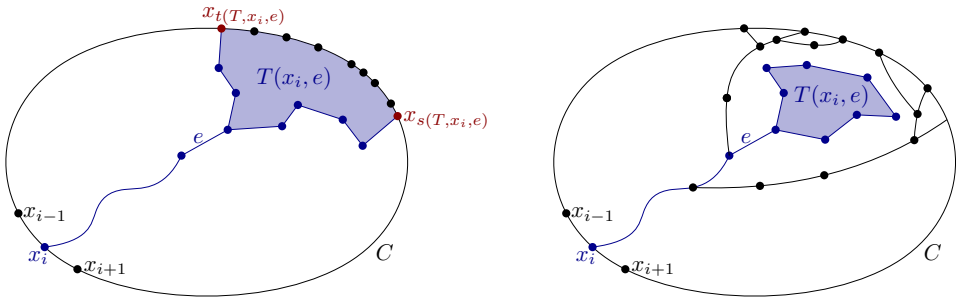


Figure 3 Schematic view of the interaction of $T(x_i, e)$ and the face defined by C. On the left, $V(T(x_i, e)) \cap V(C)$ is not empty, but on the right, $V(T(x_i, e)) \cap V(C)$ is empty.

Using data structures for dynamic forests, one can maintain the tree T under swapping of edges (removing an edge and inserting an edge while maintaining a tree) and changing the root such that: each update takes $O(\log n)$ time; the indices $s(T, x_i, e)$ and $t(T, x_i, e)$ can be obtained in $O(\log n)$ time for any query $(x_i, e) \in V(C) \times E(T)$. The basic idea for this is to maintain the spanning tree T^* of the dual graph that uses the edges dual to those in $E(G) \setminus E(T)$. The two edges defining $s(T, x_i, e)$ and $t(T, x_i, e)$ are, in the dual tree, the only edges dual to the edges of $E(C)$ that lie on the unique path of T^* that connects the two faces bounded by e. It is easy to extend data structures for dynamic trees [1, 43] to handle such type of queries. The edges dual to $E(C)$ are marked as special in T^* and we have to find the special edges that lie in a path of T^* between two given nodes. See [9, 32] for data structures with similar properties.

Let T be a tree, let T' be the tree obtained from T by deleting the edge e and inserting the edge e', let r be a vertex of T and T', let $A = (a_i = d_T(r, x_i))_{i \in [k]}$ the sequence of distances in T from r to x_i (for $i \in [k]$), and let $A' = (a'_i = d_{T'}(r, x_i))_{i \in [k]}$ the sequence of distances in T' from r to x_i (for $i \in [k]$). Assume that e in T and e' in T' have a common target; this means that in T we have $e = x \to z$ and in T' we have $e' = y \to z$, where the vertex z is the same. Then, we can go from A to A' by performing $O(1)$ operations Add. More precisely, for each vertex x_i that lies in $T(r, e)$ we have to decrease the value a_i by $d_T(r, z)$ and increase it by $d_{T'}(r, z)$. (Because e and e' have the same target we have $V(T(r, e)) = V(T'(r, e'))$.) Each one of these operations is done with at most two operations Add. For example, if $s(T, r, e) < t(T, r, e)$, then we perform $\textsc{Add}(s(T, r, e), t(T, r, e), -d_T(r, z))$, but if $s(T, r, e) > t(T, r, e)$, we split it into the two updates $\textsc{Add}(s(T, r, e), k, -d_T(r, z))$ and $\textsc{Add}(1, t(T, r, e), -d_T(r, z))$.

Finally, we note that [9, 32] show how to maintain a shortest-path tree as the root moves along the boundary of the face defined by C performing $O(n)$ swaps of the type $T' = (T - e) + e'$ that have the property that e and e' have the same target vertex. When sliding the root from x_i to x_{i+1}, one has to also update the distances because of the change of root. This amounts to adding $\ell(x_i x_{i+1})$ to all the values of $V(C) \cap V(T(x_{i+1}, x_i x_{i+1})$ and subtracting $\ell(x_i x_{i+1})$) to all the values of $V(C) \cap V(T(x_i, x_i x_{i+1}))$. The result follows. ◀

6.3 Testing

We discuss now how to test if $H = \text{int}(G, C)$ is an isometric subgraph of G. Recall that $E(H_{\text{out}}) = E(G) \setminus E(H)$. We add to H_{out} the edges of C with very large weights, say $2 \sum_{e \in E(G)} \ell(e)$, so that the new edges never appear in a shortest path.

Let $x_1, \ldots, x_k$ be the vertices of C ordered as they appear in C. The cycle C defines a face in H and in H_{out} and therefore we can use Lemma 14 for each of those graphs. For H we obtain the sequence of operations $\Lambda_2, \ldots, \Lambda_k$ that are needed to maintain

$$D_i = \Big(\; a_i = d_H(x_i, x_j) \; \Big)_{j \in [k]},$$

while for H_{out} we obtain the the sequence of operations $\Lambda'_2, \ldots, \Lambda'_k$ that are needed to maintain

$$D'_i = \Big(\; a'_i = d_{H_{\text{out}}}(x_i, x_j) \; \Big)_{j \in [k]}.$$

The objective now is to maintain the sequence of values

$$\tilde{D}_i = \Big(\; \tilde{a}_i = d_{H_{\text{out}}}(x_i, x_j) - d_H(x_i, x_j) = a'_i - a_i \; \Big)_{j \in [k]}.$$

as we move the root x_i along C. We start computing $\tilde{D}_1$ by using a shortest-path tree in H and in H_{out} from x_1. We store the sequence $\tilde{D}_1$ using the data structure of Lemma 13.

For $i = 2$ to $i = k$, to move from $\tilde{D}_{i-1}$ to $\tilde{D}_i$ we have to perform the operations of Λ'_i (as they are) and the operations of Λ_i with reversed sign. Each operation is done in $O(\log n)$ time per operation using the data structure of Lemma 13. Whenever we obtain the sequence $\tilde{D}_i$, we query the data structure with MIN to obtain

$$m_i \;:=\; \min\{\tilde{D}_i\} \;=\; \min\{d_{H_{\text{out}}}(x_i, x_j) - d_H(x_i, x_j) \mid j \in [k]\}.$$

If some m_i is negative, then we know that for some $x_j \in V(C)$ we have

$$d_{H_{\text{out}}}(x_i, x_j) < d_H(x_i, x_j),$$

and therefore H is not an isometric subgraph of G. If $m_i \geq 0$ for all $i \in [k]$, then we have

$$\forall i, j \in [k]: \quad d_{H_{\text{out}}}(x_i, x_j) \geq d_H(x_i, x_j).$$

Because $\partial H \subseteq V(C) = \{x_1, \ldots, x_k\}$, it follows from Lemma 1 that H is an isometric subgraph in G.

Because of Lemma 14 we spend $O(n \log n)$ time to know which $O(n)$ operations in the sequence of numbers have to be performed. Since each operation in the sequence of numbers takes $O(\log n)$ time (Lemma 13), the total running time is $O(n \log n)$.

A similar procedure can be done for testing *convexity* because of Lemma 4: the graph H is convex in G if and only if $m_i > 0$ for all $i \in [k]$. We summarize.

▶ **Theorem 15.** *Let G be a plane graph with n vertices and let $H = \text{int}(G, C)$ be a subgraph of G enclosed by a cycle C of G. In $O(n \log n)$ time we can test whether H is an isometric subgraph of G and whether H is a convex subgraph of G.*

The result can be extended to subgraphs of a plane graph that are defined by a few cycles, as shown in Figure 1. We omit this extension from this shortened version.

References

1 Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, 2005. doi:10.1145/1103963.1103966.

2 Hans-Jürgen Bandelt and Victor Chepoi. Graphs with connected medians. *SIAM J. Discret. Math.*, 15(2):268–282, 2002. doi:10.1137/S089548019936360X.

3 Hans-Jürgen Bandelt and Victor Chepoi. Metric graph theory and geometry: a survey. In *Surveys on discrete and computational geometry*, volume 453 of *Contemp. Math.*, pages 49–86. Amer. Math. Soc., Providence, RI, 2008. doi:10.1090/conm/453/08795.

4 Laurine Bénéteau, Jérémie Chalopin, Victor Chepoi, and Yann Vaxès. Medians in median graphs and their cube complexes in linear time. *J. Comput. Syst. Sci.*, 126:80–105, 2022. doi:10.1016/J.JCSS.2022.01.001.

5 Joseph Berleant, Kristin Sheridan, Anne Condon, Virginia Vassilevska Williams, and Mark Bathe. Isometric Hamming embeddings of weighted graphs. *Discret. Appl. Math.*, 332:119–128, 2023. doi:10.1016/J.DAM.2023.02.005.

6 Karl Bringmann, Thore Husfeldt, and Måns Magnusson. Multivariate analysis of orthogonal range searching and graph distances. *Algorithmica*, 82(8):2292–2315, 2020. doi:10.1007/s00453-020-00680-z.

7 Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. *ACM Trans. Algorithms*, 15(2):21:1–21:38, 2019. doi:10.1145/3218821.

8 Sergio Cabello. Computing the inverse geodesic length in planar graphs and graphs of bounded treewidth. *ACM Trans. Algorithms*, 18(2):14:1–14:26, 2022. doi:10.1145/3501303.

9 Sergio Cabello, Erin W. Chambers, and Jeff Erickson. Multiple-source shortest paths in embedded graphs. *SIAM J. Comput.*, 42(4):1542–1571, 2013. doi:10.1137/120864271.

10 Sergio Cabello and Christian Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. *Comput. Geom.*, 42(9):815–824, 2009. doi:10.1016/j.comgeo.2009.02.001.

11 Manoj Changat, Henry Martyn Mulder, and Gerard Sierksma. Convexities related to path properties on graphs. *Discret. Math.*, 290(2/3):117–131, 2005. doi:10.1016/j.disc.2003.07.014.

12 Victor D. Chepoi. Isometric subgraphs of Hamming graphs and d-convexity. *Cybernetics (Kibernetyka)*, 24:6–11, 1988. doi:10.1007/BF01069520.

13 Jongmin Choi, Sergio Cabello, and Hee-Kap Ahn. Maximizing dominance in the plane and its applications. *Algorithmica*, 83(11):3491–3513, 2021. doi:10.1007/s00453-021-00863-2.

14 Debarati Das, Evangelos Kipouridis, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. A simple algorithm for multiple-source shortest paths in planar digraphs. In *5th Symposium on Simplicity in Algorithms, SOSA*, pages 1–11. SIAM, 2022. doi:10.1137/1.9781611977066.1.

15 Dragomir Ž. Djoković. Distance-preserving subgraphs of hypercubes. *Journal of Combinatorial Theory, Series B*, 14:263–267, 1973. doi:10.1016/0095-8956(73)90010-5.

16 Mitre Costa Dourado, John G. Gimbel, Jan Kratochvíl, Fábio Protti, and Jayme Luiz Szwarcfiter. On the computation of the hull number of a graph. *Discret. Math.*, 309(18):5668–5674, 2009. doi:10.1016/j.disc.2008.04.020.

17 Guillaume Ducoffe, Michel Habib, and Laurent Viennot. Diameter, eccentricities and distance oracle computations on H-minor free graphs and graphs of bounded (distance) Vapnik-Chervonenkis dimension. *SIAM J. Comput.*, 51(5):1506–1534, 2022. doi:10.1137/20M136551X.

18 David Eppstein. Recognizing partial cubes in quadratic time. *J. Graph Algorithms Appl.*, 15(2):269–293, 2011. doi:10.7155/jgaa.00226.

19 V.V. Firsov. Isometric embedding of a graph in a Boolean cube. *Cybernetics (Kibernetyka)*, 1:112–113, 1965. doi:10.1007/BF01074705.

20 Michael L. Fredman and Robert E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

21 Dmitrii Gavrilev and Ilya Makarov. Fast approximate convex hull construction in networks via node embedding. *IEEE Access*, 11:54588–54595, 2023. doi:10.1109/ACCESS.2023.3281337.

22 Pawel Gawrychowski, Haim Kaplan, Shay Mozes, Micha Sharir, and Oren Weimann. Voronoi diagrams on planar graphs, and computing the diameter in deterministic $\tilde{O}(n^{5/3})$ time. *SIAM J. Comput.*, 50(2):509–554, 2021. doi:10.1137/18M1193402.

23 Roland Glantz and Henning Meyerhenke. On finding convex cuts in general, bipartite and plane graphs. *Theor. Comput. Sci.*, 695:54–73, 2017. doi:10.1016/j.tcs.2017.07.026.

24 Richard Hammack, Wilfried Imrich, and Sandi Klavžar. *Handbook of product graphs*. CRC Press, Boca Raton, FL, second edition, 2011.

25 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.

26 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

27 Wilfried Imrich and Sandi Klavžar. A convexity lemma and expansion procedures for bipartite graphs. *Eur. J. Comb.*, 19(6):677–685, 1998. doi:10.1006/eujc.1998.0229.

28 Wilfried Imrich, Sandi Klavžar, and Henry Martyn Mulder. Median graphs and triangle-free graphs. *SIAM J. Discret. Math.*, 12(1):111–118, 1999. doi:10.1137/S0895480197323494.

29 Wilfried Imrich and Iztok Peterin. Recognizing Cartesian products in linear time. *Discret. Math.*, 307(3-5):472–483, 2007. doi:10.1016/j.disc.2005.09.038.

30 Adam Karczmarz and Da Wei Zheng. Subquadratic algorithms in minor-free digraphs: (weighted) distance oracles, decremental reachability, and more. In *Proceedings of the 2025 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 4338–4351, 2025. doi:10.1137/1.9781611978322.147.

31 Sandi Klavžar and Henry Martyn Mulder. Median graphs: Characterizations, location theory and related structures. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 30:103–127, 1999.

32 Philip N. Klein. Multiple-source shortest paths in planar graphs. In *ACM-SIAM Symposium on Discrete Algorithms, SODA 2005*, pages 146–155. SIAM, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070454.

33 Hung Le and Christian Wulff-Nilsen. VC set systems in minor-free (di)graphs and applications. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 5332–5360, 2024. doi:10.1137/1.9781611977912.192.

34 Tilen Marc and Lovro Šubelj. Convexity in complex networks. *Netw. Sci.*, 6(2):176–203, 2018. doi:10.1017/nws.2017.37.

35 Henry Martyn Mulder and Alexander Schrijver. Median graphs and Helly hypergraphs. *Discret. Math.*, 25(1):41–50, 1979. doi:10.1016/0012-365X(79)90151-1.

36 Martyn Mulder. The structure of median graphs. *Discret. Math.*, 24(2):197–204, 1978. doi:10.1016/0012-365X(78)90199-1.

37 Ladislav Nebeský. Median graphs. *Commentationes Mathematicae Universitatis Carolinae*, 12(2):317–325, 1971.

38 Sergei Ovchinnikov. *Partial Cubes*, chapter 5, pages 127–181. Springer New York, 2011. doi:10.1007/978-1-4614-0797-3_5.

39 Ignacio M. Pelayo. *Geodesic convexity in graphs*. SpringerBriefs in Mathematics. Springer, New York, 2013. doi:10.1007/978-1-4614-8699-2.

40 Seth Pettie and Vijaya Ramachandran. Computing shortest paths with comparisons and additions. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 267–276, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545417.

41 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Symposium on Theory of Computing Conference, STOC'13*, pages 515–524. ACM, 2013. doi:10.1145/2488608.2488673.

42 Kristin Sheridan, Joseph Berleant, Mark Bathe, Anne Condon, and Virginia Vassilevska Williams. Factorization and pseudofactorization of weighted graphs. *Discret. Appl. Math.*, 337:81–105, 2023. doi:10.1016/J.DAM.2023.04.019.

43 Daniel D. Sleator and Robert E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

44 Lovro Šubelj, Dalibor Fiala, Tadej Ciglarič, and Luka Kronegger. Convexity in scientific collaboration networks. *J. Informetrics*, 13(1):10–31, 2019. doi:10.1016/j.joi.2018.11.005.

45 Maximilian Thiessen and Thomas Gaertner. Active learning of convex halfspaces on graphs. In *Advances in Neural Information Processing Systems*, volume 34, pages 23413–23425. Curran Associates, Inc., 2021. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/c4bf1e24f3e6f92ca9dfd9a7a1a1049c-Paper.pdf.

46 Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM*, 46(3):362–394, 1999. doi:10.1145/316542.316548.

47 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

48 Peter M. Winkler. Isometric embedding in products of complete graphs. *Discret. Appl. Math.*, 7(2):221–225, 1984. doi:10.1016/0166-218X(84)90069-6.

Algorithms for Distance Problems in Continuous Graphs

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

Delia Garijo
University of Seville, Spain

Antonia Kalb
Technical University of Dortmund, Germany

Fabian Klute
Universitat Politècnica de Catalunya, Barcelona, Spain

Irene Parada
Universitat Politècnica de Catalunya, Barcelona, Spain

Rodrigo I. Silveira
Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract

We study the problem of computing the diameter and the mean distance of a continuous graph, i.e., a connected graph where all points along the edges, instead of only the vertices, must be taken into account. It is known that for continuous graphs with m edges these values can be computed in roughly $O(m^2)$ time. In this paper, we use geometric techniques to obtain subquadratic time algorithms to compute the diameter and the mean distance of a continuous graph for two well-established classes of sparse graphs. We show that the diameter and the mean distance of a continuous graph of treewidth at most k can be computed in $O(n \log^{O(k)} n)$ time, where n is the number of vertices in the graph. We also show that computing the diameter and mean distance of a continuous planar graph with n vertices and F faces takes $O(nF \log n)$ time.

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

Keywords and phrases diameter, mean distance, continuous graph, treewidth, planar graph

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.13

Related Version *Full Version*: `https://arxiv.org/abs/2503.07769` [11]

Funding *Sergio Cabello*: Funded in part by the Slovenian Research and Innovation Agency (P1-0297, N1-0218, N1-0285). Funded in part by the European Union (ERC, KARST, project number 101071836). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.
Delia Garijo: Partially supported by grants PID2019-104129GB-I00 and PID2023-150725NB-I00 funded by MICIU/AEI/10.13039/501100011033.
Fabian Klute: Partially supported by grants PID2019-104129GB-I00 and PID2023-150725NB-I00 funded by MICIU/AEI/10.13039/501100011033.
Irene Parada: Serra Húnter Fellow. Partially supported by grants PID2019-104129GB-I00 and PID2023-150725NB-I00 funded by MICIU/AEI/10.13039/501100011033.
Rodrigo I. Silveira: Partially supported by grants PID2019-104129GB-I00 and PID2023-150725NB-I00 funded by MICIU/AEI/ 10.13039/501100011033.

© Sergio Cabello, Delia Garijo, Antonia Kalb, Fabian Klute, Irene Parada, and Rodrigo I. Silveira;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Graph parameters dealing with distances provide fundamental information on the graph. The *diameter*, defined as the maximum distance between any two vertices of a graph, and the *mean distance*, which gives the average of all those distances, are natural concepts of great importance in real-world applications. While the diameter gives the maximum eccentricity in the graph, the mean distance provides a measure of its compactness, and is closely related to the *sum of the pairwise distances* of the graph and the well-known *Wiener index*.[1]

Computing the diameter and the sum of the pairwise distances of a given graph G is a central problem in algorithmic graph theory. A straightforward algorithm is to perform Dijkstra's algorithm from each vertex, allowing to compute both parameters in $O(nm + n^2 \log n)$ time, for n and m the number of vertices and edges of G, respectively. Given the high computational cost of this approach, considerable effort has been invested in developing faster algorithms, especially for sparse graphs. It turns out that the general problem is notably difficult. In 2013, it was showed that in sparse graphs, no $O(n^{2-\varepsilon})$-time algorithm can distinguish between diameter 2 and 3, unless the Strong Exponential Time Hypothesis fails [29]. From the proof, one can deduce the same conditional lower bound for computing the sum of the pairwise distances of the graph (see also [9]). This justifies the vast amount of ongoing research on identifying classes of sparse graphs for which these parameters can actually be computed in subquadratic time. Currently, such classes include graphs of bounded treewidth [1, 8, 12], graphs of bounded distance VC dimension [18, 26], median graphs [4, 5], and planar graphs [9, 22].

In this work, we tackle the challenge of subquadratic diameter and mean distance computation for *continuous graphs* (these objects are also called *metric graphs* in other areas closer to analysis [19, 27]). Our main motivation arises from *geometric graphs*. A geometric graph is an undirected graph where each vertex is a two-dimensional point, and each edge is a straight line segment between the corresponding two points. These graphs naturally arise in applications involving geographic information, such as road or river networks. The main characteristic of geometric graphs is that every point on each edge is considered part of the graph. Therefore, the graph can be considered an infinite point set. The diameter of a geometric graph is the maximum distance taken over its infinitely many pairs of points. The class of continuous graphs is actually more general than geometric graphs, and is formally defined in Section 2. Distances in continuous graphs, especially the diameter, have received a lot of interest recently, mainly in the context of augmentation problems [2, 13–16, 20, 21, 23, 24]; see also [3] for results on the mean distance in the context of geometric analysis. Another well-known related problem about distances in graphs, also with a continuous aspect, is the computation of the absolute center of a graph, originally proposed by Hakimi [25].

The diameter and the mean distance of a continuous graph with m edges can be computed in roughly $O(m^2)$ time [13, 16, 21][2]. For the diameter, this follows from the fact that, in a continuous graph, any pair of points attaining the diameter, called *diametral pair*, consists of either: (i) two vertices, (ii) two points on distinct non-pendant edges,[3] or (iii) a pendant vertex and a point on a non-pendant edge [13, Lemma 6] (see Figure 1). Regarding the mean distance, one can show that it is given by a weighted sum of the mean distances of all ordered pairs of edges, which can be obtained in constant time, once the distance matrix of the vertices of the graph has been computed [21].

---

[1] The sum of the pairwise distances of a graph is the sum of distances between all ordered pairs of vertices, and half of this value is the Wiener index. This topological index has been studied extensively.

[2] The algorithm to compute the diameter in [13] is for plane geometric graphs, but it also applies here.

[3] An edge $uv \in E(G)$ is *pendant* if either u or v is a *pendant* vertex (i.e., has degree 1).

Figure 1 Types of diametral pairs of a continuous graph.

However, for sparse graphs, one hits again a quadratic running time barrier. Algorithms for diameter in discrete graphs do not carry over to continuous graphs, except in few situations (e.g., if there are only $O(1)$ different edge weights, then $O(1)$ Steiner points can be added to each edge, so that the diameter coincides with that of the continuous graph), and the same conditional lower bound of the discrete setting holds for the continuous case (one can reduce the continuous case to the discrete case by simply adding enough long paths to the graph.)

The main challenge for continuous graphs is that the techniques that have successfully worked to speed-up the computation of the diameter and the sum of the pairwise distances for discrete graphs do not seem to easily extend. The most similar setting to ours is perhaps that of planar graphs, for which recently the first subquadratic algorithms were discovered [9, 22]. These works use Voronoi diagrams in planar graphs to compute those values in the discrete setting. However, it is not clear whether they can be adapted to the continuous setting. More precisely, for a fixed source vertex and a fixed subgraph H of a graph G, they compute the Voronoi diagram of H using some additive weights. As the source moves, the additive weights defining the Voronoi diagram change, and the Voronoi diagrams change. Tracing those changes efficiently seems difficult, especially because the combinatorial structure of the Voronoi diagram may undergo important changes. Moreover, such changes can happen for several different movements of the sources. Thus, to achieve a subquadratic algorithm for planar continuous graphs, it seems that one should be able to treat those parallel changes in groups. The current technology for planar graphs does not seem ready for this.

Contributions. In this work, we present subquadratic algorithms to compute the diameter and the mean distance for two classes of sparse continuous graphs. In Sections 3 and 4, We study continuous graphs of bounded treewidth and show how to compute, respectively, its diameter and its mean distance in subquadratic time. In fact, we consider the slightly more general framework of computing the diameter $\operatorname{diam}(\mathcal{H}, \mathcal{G})$ and the mean distance $\operatorname{mean}(\mathcal{H}, \mathcal{G})$ of a subgraph $\mathcal{H}$ of a continuous graph $\mathcal{G}$ *with respect to* $\mathcal{G}$, which are the diameter and mean distance of $\mathcal{G}$ when $\mathcal{H} = \mathcal{G}$. This concept appears naturally in our algorithm during the recursion. Theorems 1 and 2 below distinguish whether the treewidth is assumed to be constant, as done in [12], or a parameter, as done in [8].

▶ **Theorem 1** (Theorems 11 and 16). *Let $k \geq 2$ be an integer constant. Let G be a graph with n vertices, treewidth at most k, nonnegative edge-lengths, and let $\mathcal{G}$ be the corresponding continuous graph. Let H be a subgraph of G and let $\mathcal{H} \subseteq \mathcal{G}$ be the corresponding continuous subgraph. We can compute the diameter* $\operatorname{diam}(\mathcal{H}, \mathcal{G})$ *and the mean distance* $\operatorname{mean}(\mathcal{H}, \mathcal{G})$ *in* $O(n \log^{4k-2} n)$ *time.*

▶ **Theorem 2** (Theorems 12 and 17). *Let G be a graph with n vertices, treewidth at most k, nonnegative edge-lengths, and let $\mathcal{G}$ be the corresponding continuous graph. Let H be a subgraph of G and let $\mathcal{H} \subseteq \mathcal{G}$ be the corresponding continuous subgraph. We can compute the diameter* $\operatorname{diam}(\mathcal{H}, \mathcal{G})$ *and mean distance* $\operatorname{mean}(\mathcal{H}, \mathcal{G})$ *in* $n^{1+\varepsilon} 2^{O(k)}$ *time, for any fixed* $\varepsilon > 0$.

Similarly to previous algorithms in the discrete setting to compute the diameter and the sum of the pairwise distances for graphs of bounded treewidth [1,8,12], the key technique that we use is orthogonal range searching; see also [17] for a novel application of this technique to compute the eccentricity and the distance-sum of any vertex of a directed weighted graph.

Finally, we investigate planar graphs. For any n-vertex continuous plane graph, we show how to compute the maximum eccentricity (i.e., largest distance from a point) over all points on the boundary of a face in $O(n \log n)$ time. Further, we show that the same approach can be used to compute the mean from all points of a face with respect to the continuous graph in $O(n \log n)$ time. This allows us to compute the diameter and mean of continuous planar graphs in $O(nF \log n)$ time, where F is the number of faces, which is subquadratic, if $F = o(\frac{n}{\log n})$. (By Euler's formula, F is the same for any embedding of a planar graph.)

▶ **Theorem 3.** *For a continuous planar graph $\mathcal{G}$ with n vertices and F faces, we can compute its diameter* $\operatorname{diam}(\mathcal{G})$ *and its mean distance* $\operatorname{mean}(\mathcal{G})$ *in $O(nF \log n)$ time.*

Note that some proofs and the section about planar graphs are deferred to the full version.

2 Preliminaries

2.1 Continuous Graphs

Consider an edge-weighted, connected graph $G = (V(G), E(G), \ell)$,where ℓ is a function that assigns a length $\ell(e) \geq 0$ to each edge $e \in E(G)$ (clearly, for our purposes we assume that not all edge lengths are 0). Informally, the *continuous graph* $\mathcal{G}$ defined by G is the infinite set of points determined by the vertices and edges of G, where each point on an edge is part of the graph. This idea requires to define precisely what we mean by a point on an edge.

For each edge uv of G, we take a closed segment of length $\ell(uv)$ with the usual metric and measure, and denote it $\mathcal{G}(uv)$. Moreover, for such an edge uv, we arbitrarily select one extreme of the segment $\mathcal{G}(uv)$ and denote it $\operatorname{endp}(uv, u)$, and call $\operatorname{endp}(uv, v)$ the other endpoint of $\mathcal{G}(uv)$. Finally, for each vertex u of G, we glue (mathematically, we identify) all points $\operatorname{endp}(uv, u)$ over all edges uv of G incident to u; we denote by $\mathcal{G}(u)$ the identified point. The continuous graph $\mathcal{G}$ defined by $G = (V(G), E(G), \ell)$ is the resulting space. The total length $\ell(\mathcal{G})$ of a continuous graph $\mathcal{G}$ defined by a graph G is defined as $\sum_{uv \in E(G)} \ell(uv)$.

We observe that one may think of $\mathcal{G}$ as a 1-dimensional simplicial complex where each edge uv is isometric to a segment of length $\ell(uv)$.

A point p of $\mathcal{G}$ can be specified by a triple $(uv, u, \lambda) \in E(G) \times V(G) \times [0, \ell(uv)]$, which represents the point of the segment $\mathcal{G}(uv)$ at distance λ (along $\mathcal{G}(uv)$) from the endpoint $\operatorname{endp}(uv, u)$. The triples (uv, u, λ) and $(uv, v, \ell(uv) - \lambda)$ define the same point of $\mathcal{G}$. Similarly, for any two incident edges uv, uv' of G, the triples $(uv, u, 0)$, $(uv, v, \ell(uv))$, $(uv', u, 0)$ and $(uv', v', \ell(uv'))$ define the same point, namely $\mathcal{G}(u)$.

We have already set the notation $\mathcal{G}(uv)$ for an edge uv and $\mathcal{G}(u)$ for a vertex u. With a slight abuse of notation, we do not distinguish uv from $\mathcal{G}(uv)$ and u from $\mathcal{G}(u)$.

In general, for any subgraph H of G, we denote by $\mathcal{G}(H)$ the continuous subgraph of $\mathcal{G}$ defined by the objects of H. This is also the continuous graph defined by H. Also, when H is clear from the context, we denote such an object by $\mathcal{H}$ and talk about $\mathcal{H}$ as a subgraph of $\mathcal{G}$. For a vertex set $A \subseteq V(G)$, we use $G[A]$ to denote the subgraph of G induced by A.

A *walk* in $\mathcal{G}$ between two points $p, q \in \mathcal{G}$ is a sequence $pv_1, v_1v_2, \ldots, v_{k-1}v_k, v_kq$ with $v_1v_2, \ldots, v_{k-1}v_k \in E(G)$. If the first and last point coincide, it is *closed*. The length of a walk is the sum of the length of its pieces, counted with multiplicity. For any two points $p, q \in \mathcal{G}$, the distance $d_{\mathcal{G}}(p, q)$ is the minimum length over all p-to-q walks. A *shortest pq-path* is a p-to-q walk $\pi(p, q)$ in $\mathcal{G}$ such that $\ell(\pi(p, q)) = d_{\mathcal{G}}(p, q)$.

We can regard $d_{\mathcal{G}}(p,q)$ as the discrete graph-theoretic distance in a graph obtained by subdividing aa' with p as a new vertex, subdividing bb' with q as a new vertex, and setting $\ell(ap) = \lambda$, $\ell(pa') = \ell(aa') - \lambda$, $\ell(bq) = \lambda'$, and $\ell(qb') = \ell(bb') - \lambda'$.

2.2 Distance Problems

Consider a continuous graph $\mathcal{G}$ defined by a graph G and a continuous subgraph $\mathcal{H} \subseteq \mathcal{G}$ defined by a subgraph $H \subseteq G$. We are interested in distances between points of $\mathcal{H}$ using the metric given by $\mathcal{G}$. One may think of $\mathcal{G}$ as the ambient space and of $\mathcal{H}$ as the relevant subset.

The *eccentricity* of a point $p \in \mathcal{H}$ with respect to $\mathcal{H}$ is $\operatorname{ecc}(p,\mathcal{H},\mathcal{G}) = \max_{q\in\mathcal{H}} d_{\mathcal{G}}(p,q)$; when $\mathcal{H} = \mathcal{G}$, we just talk about the eccentricity of p in $\mathcal{G}$ and write $\operatorname{ecc}(p,\mathcal{G})$ for $\operatorname{ecc}(p,\mathcal{H},\mathcal{G})$. The *diameter* of $\mathcal{H}$ with respect to $\mathcal{G}$ is defined as $\operatorname{diam}(\mathcal{H},\mathcal{G}) = \max_{p,q\in\mathcal{H}} d_{\mathcal{G}}(p,q)$; when $\mathcal{H} = \mathcal{G}$, we just talk about the diameter of $\mathcal{G}$ and write $\operatorname{diam}(\mathcal{G})$ for $\operatorname{diam}(\mathcal{G},\mathcal{G})$. It is easy to see that $\operatorname{diam}(\mathcal{H},\mathcal{G}) = \max_{p\in\mathcal{H}} \operatorname{ecc}(p,\mathcal{H},\mathcal{G})$. The *sum of distances* of $\mathcal{H}$ in $\mathcal{G}$ is the sum of the pairwise distances in $\mathcal{H}$, using the metric from $\mathcal{G}$, that is, $\operatorname{sumdist}(\mathcal{H},\mathcal{G}) = \iint_{p,q\in\mathcal{H}} d_{\mathcal{G}}(p,q)\,dp\,dq$. The *mean distance* of $\mathcal{H}$ in $\mathcal{G}$ is $\operatorname{mean}(\mathcal{H},\mathcal{G}) = \frac{1}{\ell(\mathcal{H})^2}\operatorname{sumdist}(\mathcal{H},\mathcal{G})$. It is easy to see that $\operatorname{mean}(\mathcal{H},\mathcal{G}) = \frac{1}{\ell(\mathcal{H})}\int_{p\in\mathcal{H}} \operatorname{mean}(p,\mathcal{H},\mathcal{G})\,dp$, where $\operatorname{mean}(p,\mathcal{H},\mathcal{G}) = \frac{1}{\ell(\mathcal{H})}\int_{q\in\mathcal{H}} d_{\mathcal{G}}(p,q)\,dq$ is the mean distance from the point $p \in \mathcal{H}$ with respect to $\mathcal{H}$ in $\mathcal{G}$. When $\mathcal{H} = \mathcal{G}$, then we just write $\operatorname{mean}(\mathcal{G})$ for $\operatorname{mean}(\mathcal{G},\mathcal{G})$ and $\operatorname{mean}(p,\mathcal{G})$ for $\operatorname{mean}(p,\mathcal{G},\mathcal{G})$.

2.3 Treewidth

The *treewidth* of a graph is an important parameter in algorithmic graph theory which, roughly speaking, measures how far the graph is from being a tree (a formal definition is given in [11]). Graphs of treewidth k have $O(kn)$ edges [6].

A *separation* in a graph G is a triple (A,B,S) such that $A,B,S \subset V(G)$, $A \cup B = V(G)$, $S = A \cap B$, and there is no edge incident to both $A \setminus B$ and $B \setminus A$. The elements of S in separation (A,B,S) are called *portals*. Each path in G from A to B must pass through a portal. Informally, we are interested in separations where A and B have a constant fraction of the vertices and S is small. Such separations of at most k portals exist for graphs of treewidth k, can be computed in linear time, and have the additional property that adding edges between the portals does not increase the treewidth [8,12]; . For simplicity, we assume that S contains exactly k portals (it may happen that is has fewer). We use the notation $[k]= \{1,\ldots,k\}$. We have two regimes, depending on whether we consider the treewidth constant or a parameter.

2.4 Orthogonal Range Searching

We use the notation $B(n,d) = \binom{d+\lceil \log n\rceil}{d}$ to bound the performance of some of our data structures. First we note the following asymptotic bounds.

▶ **Lemma 4** (Bringmann, Husfeldt, and Magnusson [8]). *$B(n,d) = O(\log^d n)$ and $B(n,d) = n^{\varepsilon}2^{O(d)}$ for each $\varepsilon > 0$.*

A *rectangle* in $\mathbb{R}^d$ is the Cartesian product of d intervals (whose extremes can be included or not). The analysis of orthogonal range searching performed in [8, Section 3] (see also [28]) leads to the data structure described in the the following theorem. We use the version suggested by Cabello [10] because it adapts better to our needs. (We use $\bigsqcup$ to indicate that the union is between pairwise disjoint sets.)

▶ **Theorem 5** (Cabello [10]). *Given a set P of n points in $\mathbb{R}^d$, there is a family of sets $\mathcal{P} = \{P_j \mid j \in J\}$ and a data structure with the following properties:*

- *$P_j \subseteq P$ for each $P_j \in \mathcal{P}$;*
- *all the sets of $\mathcal{P}$ together have $O(nd \cdot B(n,d))$ points, i.e., $\sum_{P_j \in \mathcal{P}} |P_j| = O(nd \cdot B(n,d))$;*
- *for each query rectangle $R \subset \mathbb{R}^d$, the data structure finds in $O(2^d B(n,d))$ time indices $J_R \subset J$ such that $|J_R| = O(2^d B(n,d))$ and $P \cap R = \bigsqcup_{j \in J_R} P_j$;*
- *the family $\mathcal{P}$ and the data structure can be computed in $O(nd \cdot B(n,d))$ time.*

3 Diameter in Graphs with Bounded Treewidth

In this section, we discuss the computation of the diameter of a continuous graph with treewidth k. Note that computing the diameter of continuous trees (i.e., treewidth 1) can be reduced to the discrete setting, because in trees the diameter is always attained by two vertices. Thus, we restrict our attention to $k > 1$. As in [1, 8, 9, 12], we use orthogonal range searching to work with distance-related problems in graphs of bounded treewidth. However, because we consider continuous graphs, we have to consider pairs of edges instead of pairs of vertices, and the interaction between edges is more complex. We handle this by using more dimensions in the range searching space.

Sometimes we need to consider an orientation for each edge of G, to distinguish its vertices. We orient the edges of G arbitrarily, but keep track of the orientation. We use uv when the orientation is not relevant and (u,v) or (v,u), depending on the orientation, when we consider it oriented. We use $E(G)$ in both cases.

3.1 Characterization of the Diameter via Walks

We start with a characterization of the diameter in the continuous setting that uses the length of walks (compare to Figure 1). For each $aa', bb' \in E(G)$, let $W(aa', bb')$ be a shortest closed walk passing through all the interior points of aa' and bb'.

▶ **Lemma 6.** *For each $aa', bb' \in E(G)$, $\max_{p \in aa',\, q \in bb'} d_{\mathcal{G}}(p,q) = \frac{\ell(W(aa', bb'))}{2}$.*

The following corollary is an immediate consequence of Lemma 6.

▶ **Corollary 7.** *If $H \subseteq G$ defines the continuous subgraph $\mathcal{H}$ of $\mathcal{G}$, then* $\operatorname{diam}(\mathcal{H}, \mathcal{G}) = \frac{1}{2} \cdot \max\{\ell(W(aa', bb')) \mid aa', bb' \in E(H)\}$.

For our computation, we use the following closed formula.

▶ **Lemma 8.** *For each $aa', bb' \in E(G)$, it holds that*

$$\ell(W(aa', bb')) = \ell(aa') + \ell(bb') + \min\{d_G(a,b) + d_G(a',b'),\ d_G(a,b') + d_G(a',b)\}.$$

For each pair of oriented edges $(a_0, a_1), (b_0, b_1) \in E(G)$, Lemma 8 implies that there are two possible values for $\ell(W(a_0a_1, b_0b_1))$. We say that (a_0, a_1, b_0, b_1) is of *type 1* if

$$\ell(W(a_0a_1, b_0b_1)) = \ell(a_0a_1) + \ell(b_0b_1) + d_G(a_0, b_0) + d_G(a_1, b_1),$$

and of *type 2* otherwise. For each oriented edge $(b_0, b_1) \in E(G)$ and type $\tau \in \{1, 2\}$, we define

$$\operatorname{Type}_\tau(b_0, b_1) = \{(a_0, a_1) \in E(G) \mid (a_0, a_1, b_0, b_1) \text{ is of type } \tau\}.$$

Therefore, $(a_0, a_1) \in \operatorname{Type}_1(b_0, b_1) \iff d_G(a_0, b_0) + d_G(a_1, b_1) \leq d_G(a_0, b_1) + d_G(a_1, b_0)$.

3.2 Diameter Across the Portals

Let (A, B, S) be a separation in G with k portals. We fix an enumeration $s_1, \ldots, s_k$ of them. Let $E_A \subseteq E(G[A])$ be the edge set of the subgraph of G induced by A and $E_B \subseteq E(G[B])$, respectively; not necessarily disjoint. For each index $i \in [k]$, each vertex $a \in A$, and each vertex $b \in B$, let $\varphi(i; a, b)$ be the logic predicate that holds whenever s_i is the first portal in the enumeration that lies in some shortest path from a to b. Formally,

$$\varphi(i; a, b) \;=\; \bigwedge_{j<i} \big[d_G(a,b) < d_G(a,s_j) + d_G(s_j,b)\big] \;\wedge\; \big[d_G(a,b) = d_G(a,s_i) + d_G(s_i,b)\big].$$

It is easy to see that, for each $(a, b) \in A \times B$, there exists a unique index $i \in [k]$ where $\varphi(i; a, b)$ holds (in other words, $|\{i \in [k] \mid \varphi(i; a, b)\}| = 1$).

We extend this to the four shortest paths defined by two vertices $a_0, a_1 \in A$ and two vertices $b_0, b_1 \in B$, by defining the following predicate for all $\kappa = (i_{0,0}, i_{0,1}, i_{1,0}, i_{1,1}) \in [k]^4$:

$$\Phi(\kappa; a_0, a_1, b_0, b_1) = \Phi((i_{0,0}, i_{0,1}, i_{1,0}, i_{1,1}); a_0, a_1, b_0, b_1) = \bigwedge_{(\alpha,\beta)\in\{0,1\}^2} \varphi(i_{\alpha,\beta}; a_\alpha, b_\beta).$$

Therefore, this predicate holds if and only if, for each $\alpha, \beta \in \{0, 1\}$, the index $i_{\alpha,\beta}$ is the smallest index i with the property that s_i lies on some shortest path from a_α to b_β. As before, for each $(a_0, a_1, b_0, b_1) \in A^2 \times B^2$, there exists a unique 4-tuple $\kappa \in [k]^4$ where $\Phi(\kappa; a_0, a_1, b_0, b_1)$ holds. For each $\kappa \in [k]^4$, each $(b_0, b_1) \in E_B$, and each type τ we define

$$\Lambda_\tau(\kappa; b_0, b_1) = \max\{\ell(W(a_0a_1, b_0b_1)) \mid (a_0, a_1) \in E_A \cap \mathrm{Type}_\tau(b_0, b_1) \wedge \Phi(\kappa; a_0, a_1, b_0, b_1)\}.$$

This represents the maximum length over all edges (a_0, a_1) of a type-τ walk between (b_0, b_1) and (a_0, a_1), consistent with the portals in κ. We next discuss how to compute efficiently $\Lambda_\tau(\kappa; b_0, b_1)$ for several edges $(b_0, b_1) \in E_B$ simultaneously.

▶ **Lemma 9.** *Consider a fixed type $\tau \in \{1, 2\}$ and indices $\kappa \in [k]^4$. In $O(m2^{4k-3}B(m, 4k-3))$ time we can compute the values $\Lambda_\tau(\kappa; b_0, b_1)$ for all $(b_0, b_1) \in E_B$.*

Proof sketch. We sketch how to use orthogonal range searching to compute $\Lambda_\tau(\kappa; b_0, b_1)$ for a fixed $\kappa \in [k]^4$. The complete proof is in [11].

For each vertex $a \in A$ and each $i \in [k]$, we define the point $p(i; a) \in \mathbb{R}^k$ whose j-th coordinate is $p_j(i; a) = d_G(a, s_i) - d_G(a, s_j)$. For each vertex $b \in B$ and each index $i \in [k]$, we define the box $R(i; b) = I_1(i; b) \times \cdots \times I_k(i; b) \subset \mathbb{R}^k$, where $I_j(i; b)$ is the interval

$$I_j(i; b) \;=\; \begin{cases} \big(-\infty,\ d_G(b, s_j) - d_G(b, s_i)\big) & \text{if } j < i, \\ \mathbb{R} & \text{if } j = i, \\ \big(-\infty,\ d_G(b, s_j) - d_G(b, s_i)\big] & \text{if } j > i. \end{cases}$$

In $p(i; a)$ and $I(i; b)$ we remove the i-th coordinate, as it does not provide any information. We use the same notation for the resulting objects, now in $\mathbb{R}^{k-1}$. It has been noted [8, 10, 12] that the point $p(i; a)$ lies in the rectangle $R(i; b)$ if and only if $\varphi(i; a, b)$ holds.

We do something similar for $\Phi(\kappa; a_0, a_1, b_0, b_1)$. For any two vertices $a_0, a_1 \in A$, we define the point $p(a_0, a_1)$ and, for any two vertices $b_0, b_1 \in B$, we make a rectangle $R(b_0, b_1)$ in $\mathbb{R}^{4k-4}$, such that $p(a_0, a_1) \in R(b_0, b_1)$ if and only if $\Phi(\kappa; a_0, a_1, b_0, b_1)$ holds.

For each edge $(a_0, a_1) \in E_A$, we extend $p(a_0, a_1)$ to $p^+(a_0, a_1)$ by an extra coordinate to distinguish between type 1 and 2. We also introduce the new rectangle $R^+(b_0, b_1)$ such that

$$\forall (b_0, b_1) \in E_B : \quad \Lambda_\tau(\kappa; b_0, b_1) = \max\{\ell(W(a_0a_1, b_0b_1)) \mid p^+(a_0, a_1) \in R^+(b_0, b_1)\}.$$

We use orthogonal range searching to handle the right side of the equation. In total, we spend $O(2^{4k-3}B(m, 4k-3))$ time per edge $(b_0, b_1) \in E_B$ to compute $\Lambda_\tau(\kappa; b_0, b_1)$. ◀

We use Lemma 9 to compute the values $\Lambda_\tau(\kappa; b_0, b_1)$ for each type $\tau \in \{1, 2\}$, all $(b_0, b_1) \in E_B$ and indices $\kappa \in [k]^4$. This requires applying Lemma 9 a total of $O(k^4)$ times, and therefore we spend $O(m2^{4k-3}k^4B(m, 4k-3))$ time. Using Lemma 8, we note that

$$\max_{p\in\mathcal{G}(E_A), q\in\mathcal{G}(E_B)} d_{\mathcal{G}}(p, q) = \tfrac{1}{2} \cdot \max\left\{\Lambda_\tau(\kappa; b_0, b_1) \mid (b_0, b_1) \in E_B,\ \kappa \in [k]^4,\ \tau \in \{1, 2\}\right\}$$

(for details check the proof of Lemma 10 in [11]).

▶ **Lemma 10.** *We can compute* $\max_{p\in\mathcal{G}(E_A), q\in\mathcal{G}(E_B)} d_{\mathcal{G}}(p, q)$ *in* $O(m2^{4k-3}k^4B(m, 4k-3))$ *time.*

▶ Remark. We are aware that some log factors can be shaved off, and that for small k one can improve the analysis slightly. However, we prefer to keep this high-level structure to keep it simpler, and parallel to the forthcoming computation of mean distance.

3.3 Global Diameter

Let G be a graph with n vertices and m edges defining the continuous graph $\mathcal{G}$. Let H be a subgraph of G defining a continuous graph $\mathcal{H} \subseteq \mathcal{G}$. We use a divide-and-conquer approach to compute $\operatorname{diam}(\mathcal{H}, \mathcal{G})$.

Let (A, B, S) be a separation in G. Let G' be the graph obtained from G by adding an edge ss' with length $d_G(s, s')$ for every pair of portals $s, s' \in S$. (If the edge already exists, we redefine its length.) For $X \in \{A, B\}$, let $\mathcal{H}_X$ and $\mathcal{G}'_X$ be the continuous graphs defined by $H[X] - E(G[S])$ and $G'[X]$, respectively. The edges ss' added to G guarantee that $d_{\mathcal{G}}(p, q) = d_{\mathcal{G}'_X}(p, q)$ for any points $p, q \in \mathcal{H}_X$. (We removed $E(G[S])$ because for points on edges between portals whose length is redefined, the statement is undefined.) Therefore

$$\begin{aligned}
\operatorname{diam}(\mathcal{H}, \mathcal{G}) &= \max\left\{\max_{p,q\in\mathcal{H}_A} d_{\mathcal{G}}(p, q),\ \max_{p,q\in\mathcal{H}_B} d_{\mathcal{G}}(p, q),\ \max_{p\in\mathcal{H}_A,\, q\in\mathcal{H}_B} d_{\mathcal{G}}(p, q)\right\} \\
&= \max\left\{\max_{p,q\in\mathcal{H}_A} d_{\mathcal{G}'_A}(p, q),\ \max_{p,q\in\mathcal{H}_B} d_{\mathcal{G}'_B}(p, q),\ \max_{p\in\mathcal{H}_A,\, q\in\mathcal{H}_B} d_{\mathcal{G}}(p, q)\right\} \\
&= \max\left\{\operatorname{diam}(\mathcal{H}_A, \mathcal{G}'_A),\ \operatorname{diam}(\mathcal{H}_B, \mathcal{G}'_B),\ \max_{p\in\mathcal{H}_A,\, q\in\mathcal{H}_B} d_{\mathcal{G}}(p, q)\right\} \qquad (1)
\end{aligned}$$

(see Figure 2). The last term can be computed by Lemma 10, which depends on the size of S.

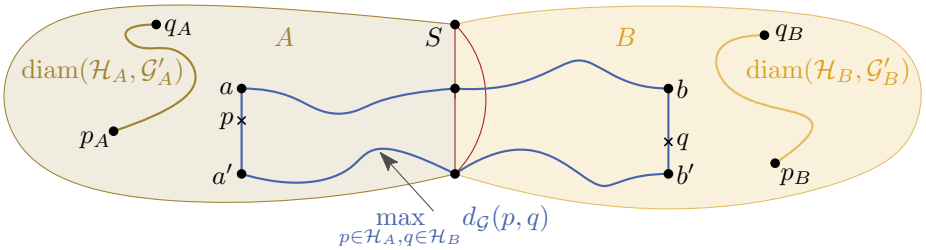


Figure 2 Visualization of the divide-and-conquer approach to compute $\operatorname{diam}(\mathcal{G})$ (see Equation (1)).

Now we have two regimes depending on whether we want to assume that the treewidth is constant, as done in [12], or whether we want to consider the treewidth a parameter, as done in [8]. The same distinction was made in [10]. This difference affects the time to find a tree decomposition and the number of portals in a balanced separation. In both cases we use that an n-vertex graph with treewidth k has $O(kn)$ edges [6].

▶ **Theorem 11.** *Let $k \geq 2$ be an integer constant. Let G be a graph with n vertices, treewidth at most k, nonnegative edge-lengths, and let $\mathcal{G}$ be the corresponding continuous graph. Let H be a subgraph of G and let $\mathcal{H} \subseteq \mathcal{G}$ be the corresponding continuous subgraph. We can compute the diameter* $\operatorname{diam}(\mathcal{H}, \mathcal{G})$ *in $O(n \log^{4k-2} n)$ time.*

Proof. If G has fewer than $2k = O(1)$ vertices, we compute $\operatorname{diam}(\mathcal{H}, \mathcal{G})$ in $O(1)$ time. Otherwise, we find in linear time a separation (A, B, S) such that: $|S| \leq k$, $\frac{n}{k+1} \leq |A| \leq \frac{nk}{k+1}$. Such a separation is given, e.g., in [12]. Further, we add an edge ss' between all portals $s, s' \in S$ with length $d_G(s, s')$. This augmentation costs $O(k(m + n \log n)) = O(n \log n)$ time.

By Lemma 10, the value $\max_{p \in \mathcal{H}_A, q \in \mathcal{H}_B} d_{\mathcal{G}}(p, q)$ is computed in $O(m 2^{4k-3} k^4 B(m, 4k-3))$ time. Using that k is constant, $m = O(n)$, and Lemma 4, this time bound is $O(n \log^{4k-3} n)$.

We construct the graphs G', $G'[A]$, $G'[B]$, $H[A] - E(G[S])$ and $H[B] - E(G[S])$ explicitly, in $O(m) = O(n)$ time. Because adding edges between the portals of S does not increase the treewidth [12][Lemma 3],the graphs $G'[A]$ and $G'[B]$ have treewidth at most k. The values $\operatorname{diam}(\mathcal{H}_A, \mathcal{G}'_A)$ and $\operatorname{diam}(\mathcal{H}_B, \mathcal{G}'_B)$ are computed recursively, and we obtain $\operatorname{diam}(\mathcal{H}, \mathcal{G})$ using Equation (1).

Since $\frac{n}{k+1} \leq |A| \leq \frac{nk}{k+1}$ and k is constant, each side of the recursion has a constant fraction of the vertices $|A| + |B| = n + k$, and the recursion depth is $O(\log n)$, leading to a total running time of $O(n \log^{4k-2} n)$. ◀

▶ **Theorem 12.** *Let G be a graph with n vertices, treewidth at most k, nonnegative edge-lengths, and let $\mathcal{G}$ be the corresponding continuous graph. Let H be a subgraph of G and let $\mathcal{H} \subseteq \mathcal{G}$ be the corresponding continuous subgraph. We can compute the diameter* $\operatorname{diam}(\mathcal{H}, \mathcal{G})$ *in $n^{1+\varepsilon} 2^{O(k)}$ time, for any fixed $\varepsilon > 0$.*

Proof. We use the same divide-and-conquer strategy as in Theorem 11. The difference is in the properties of the separation. If G has $O(k)$ vertices, we compute $\operatorname{diam}(\mathcal{H}, \mathcal{G})$ in $O(k^2)$ time. Otherwise, we proceed as follows.

First, we note that, given a tree decomposition of G of width k', we can obtain in linear time a separation (A, B, S) in G with the following properties: the set S of portals for A has $k' + 1$ portals; both A and $B = (V(G) \setminus A)$ have $\Theta(n - k)$ vertices each; adding edges between the vertices of S does not increase the treewidth *of the tree decomposition*. See for example [6, Theorem 19]; the set S is a bag of the decomposition and thus the tree decomposition keeps being valid with the addition of edges within S.

It is shown in [7] that, for graphs of treewidth at most k, one can find a tree decomposition of width $k' = 3k + 4$ in $2^{O(k)} n \log n$ time. From this we obtain the separation (A, B, S) in G mentioned above, where $|S| \leq k' + 1 = 3k + 5$. By Lemma 10, the value $\max_{p \in \mathcal{H}_A, q \in \mathcal{H}_B} d_{\mathcal{G}}(p, q)$ is computed in $O(m 2^{O(k)} (k' + 1)^4 B(m, O(k)))$ time. Using that $m = O(kn)$ and the estimate of Lemma 4, this time bound becomes

$$O((kn) 2^{O(k)} (k' + 1)^4 \cdot B(n, O(k))) = O(n 2^{O(k)} \cdot n^{\varepsilon} 2^{O(k)}) = n^{1+\varepsilon} 2^{O(k)},$$

where $\varepsilon > 0$ can be chosen arbitrarily small.

To compute $\operatorname{diam}(\mathcal{H}_A, \mathcal{G}'_A)$ and $\operatorname{diam}(\mathcal{H}_B, \mathcal{G}'_B)$ recursively, we pass to the subproblems the tree decomposition we have computed, trimmed to the vertices of A and B, respectively. We also can adapt it to keep the tree decompositions of size $O(|A|)$ and $O(|B|)$, respectively. In this way, at any level of the recursion, we always have a set S with $k' + 1 = 3k + 5$ portals. Thus, we compute the tree decomposition only once, and then pass it to each subproblem trimmed to the relevant vertices. The recursive calls add a logarithmic factor to the total running time, which is absorbed by the polynomial term $n^{1+\varepsilon}$.

(The reason for passing the tree decomposition to the subproblems is that adding the edges between the portals in S may give a clique of size k', which increases the treewidth of G'. Computing an approximate tree decomposition of G' anew would increase the width of the decomposition at each level. However, if we pass the tree decomposition to the subproblems, we keep the width of the decomposition bounded by $3k+5$ at all levels of the recursion.) ◀

4 Mean Distance in Graphs with Bounded Treewidth

In this section, we discuss the computation of the mean distance in continuous graphs with treewidth k. The case of $k = 1$ corresponds to continuous trees, one of the few graph classes for which linear-time algorithms are known [21]. Thus, we focus on $k > 1$. All our efforts will be on the computation of sumdist$(\mathcal{H}, \mathcal{G})$, from which mean$(\mathcal{H}, \mathcal{G})$ can be easily computed. Again, we use orthogonal range searching, but now we need to efficiently retrieve sums of distances. We achieve this by representing the sum of distances between pairs of edges as volumes of collections of triangular prisms, which can be represented in a compact way.

4.1 Mean Distance as a Volume

We compute the sum of all distances in a continuous graph as the volume of a collection of triangular prisms. A *truncated triangular prism* is formed when a prism is sliced by a plane that is not parallel to its bases; the heights are the lengths of the three edges orthogonal to the basis. The volume of a truncated triangular prism with base $\triangle$ and heights h_1, h_2, h_3 is $\frac{1}{3}\operatorname{area}(\triangle)(h_1 + h_2 + h_3)$.

Consider the following setting defined by a complete graph with vertex set $\{a_0, a_1, b_0, b_1\}$ and variable edge lengths, as follows: (i) a_0a_1 has variable length $y > 0$, (ii) b_0b_1 has variable length $z > 0$, (iii) for all $\alpha, \beta \in \{0, 1\}$, $a_\alpha b_\beta$ has length $x_{\alpha,\beta} \geq 0$.

Let $K = K(y, z, x_{0,0}, x_{0,1}, x_{1,0}, x_{1,1})$ denote this graph, and let $\mathcal{K}$ denote the corresponding continuous graph. See Figure 3. We say that the 6-tuple $(y, z, x_{0,0}, x_{0,1}, x_{1,0}, x_{1,1})$ is *compliant* if, for all $\alpha, \beta \in \{0, 1\}$ it holds $x_{\alpha,\beta} = d_K(a_\alpha, b_\beta)$. In our setting we only need to consider compliant cases. (This poses some conditions on the values that the variables can take.)

We want to understand how the total sum of distances between points on a_0a_1 and b_0b_1,

$$\xi(y, z, x_{0,0}, x_{0,1}, x_{1,0}, x_{1,1}) = \iint_{p\in\mathcal{K}(a_0a_1), q\in\mathcal{K}(b_0b_1)} d_{\mathcal{K}}(p, q)\, dp\, dq,$$

looks like. For each $\lambda \in [0, y]$, let $p(\lambda)$ be the point specified by the triple (a_0a_1, a_0, λ), and, for each $\mu \in [0, z]$, let $q(\mu)$ be the point specified by the triple (b_0b_1, b_0, μ). Then

$$\xi(y, z, x_{0,0}, x_{0,1}, x_{1,0}, x_{1,1}) = \iint_{(\lambda,\mu)\in[0,y]\times[0,z]} d_{\mathcal{K}}(p(\lambda), q(\mu))\, d\lambda\, d\mu.$$

Function $(\lambda, \mu) \mapsto d_{\mathcal{K}}(p(\lambda), q(\mu))$, defined in $[0, y] \times [0, z]$, is the lower envelope of four functions

$$x_{0,0} + \lambda + \mu,\ x_{0,1} + \lambda + z - \mu,\ x_{1,0} + y - \lambda + \mu,\ x_{1,1} + y - \lambda + z - \mu.$$

Following [21], we call the graph of function $(\lambda, \mu) \mapsto d_{\mathcal{K}}(p(\lambda), q(\mu))$ a *roof*; the value ξ is the volume below the roof. The minimization diagram of this function consists of convex pieces; see Figure 3. The gradient of each function is of the form $(\pm 1, \pm 1)$. When the variable values are compliant, all four functions appear in the lower envelope, and the minimization diagram has four regions. (Some of them may contain only part of an edge of the domain.)

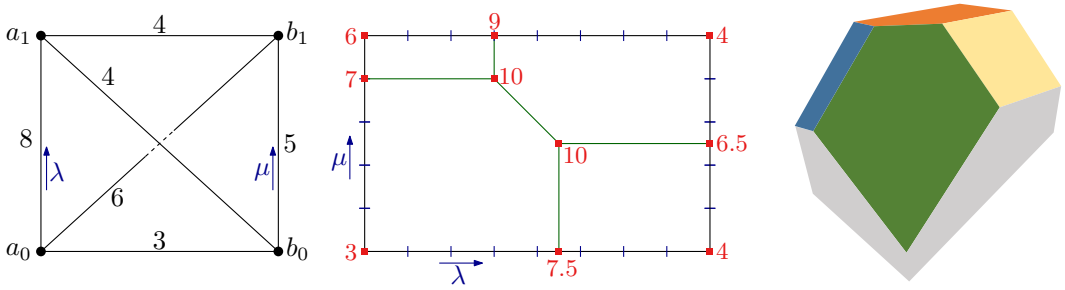


Figure 3 Concrete example of the minimization diagram of $d_{\mathcal{K}}(p(\lambda), q(\mu))$. In the center, some values of $d_{\mathcal{K}}(p(\lambda), q(\mu))$ are shown in red; on the right, 3D visualization of the roofs.

▶ **Lemma 13.** *Consider the 4-variable linear function* $L(x_{0,0}, x_{0,1}, x_{1,0}, x_{1,1}) = x_{1,0} + x_{0,1} - x_{0,0} - x_{1,1}$. *There are two 6-variable polynomials* $\varrho_+(\cdot), \varrho_-(\cdot)$ *of degree at most three with the following property: when* $(y, z, x_{0,0}, x_{0,1}, x_{1,0}, x_{1,1})$ *is compliant,*

$$\xi(y, z, x_{0,0}, x_{0,1}, x_{1,0}, x_{1,1}) = \begin{cases} \varrho_+(y, z, x_{0,0}, x_{0,1}, x_{1,0}, x_{1,1}), & \text{if } L(x_{0,0}, x_{0,1}, x_{1,0}, x_{1,1}) \geq 0, \\ \varrho_-(y, z, x_{0,0}, x_{0,1}, x_{1,0}, x_{1,1}), & \text{otherwise.} \end{cases}$$

4.2 Integral Across the Portals

We reuse much of the notation and ideas from Section 3.2. As before, G is a graph with n vertices and m edges, (A, B, S) is a separation in G with exactly k portals, and we fix an enumeration of the portals as $s_1, \ldots, s_k$. We also fix an orientation for the edges of G. Let $E_A \subseteq E(G[A])$ be the edge sets of the subgraph of G induced by A and $E_B \subseteq E(G[B])$, respectively; not necessarily disjoint. Our objective in this section is to compute the sum of distances between points in E_A and E_B:

$$\iint_{p \in \mathcal{G}(E_A), q \in \mathcal{G}(E_B)} d_{\mathcal{G}}(p, q)\, dp\, dq = \sum_{(a_0, a_1) \in E_A} \sum_{(b_0, b_1) \in E_B} \iint_{p \in \mathcal{G}(a_0 a_1), q \in \mathcal{G}(b_0 b_1)} d_{\mathcal{G}}(p, q)\, dp\, dq.$$

Define the graph $\mathcal{K} = \mathcal{K}(\ell(a_0a_1), \ell(b_0b_1), d_G(a_0, b_0), d_G(a_0, b_1), d_G(a_1, b_0), d_G(a_1, b_1))$, for two edges $(a_0, a_1) \in E_A$, $(b_0, b_1) \in E_B$.

The continuous edges a_0a_1 and b_0b_1 belong to $\mathcal{G}$ and $\mathcal{K}$ and, for each $p \in \mathcal{G}(a_0a_1)$ and $q \in \mathcal{G}(b_0b_1)$ we have $d_{\mathcal{G}}(p, q) = d_{\mathcal{K}}(p, q)$. It follows that

$$\iint_{p \in \mathcal{G}(a_0 a_1), q \in \mathcal{G}(b_0 b_1)} d_{\mathcal{G}}(p, q)\, dp\, dq = \iint_{p \in \mathcal{K}(a_0 a_1), q \in \mathcal{K}(b_0 b_1)} d_{\mathcal{K}}(p, q)\, dp\, dq.$$

It follows that our objective is to compute

$$\sum_{(a_0, a_1) \in E_A} \sum_{(b_0, b_1) \in E_B} \xi(\ell(a_0a_1), \ell(b_0b_1), d_G(a_0, b_0), d_G(a_0, b_1), d_G(a_1, b_0), d_G(a_1, b_1)).$$

We keep using the predicates $\varphi(i; a, b)$ and $\Phi(\kappa; a_0, a_1, b_0, b_1)$ defined in Section 3.2, where $a, a_0, a_1 \in A$, $b, b_0, b_1 \in B$, $i \in [k]$ and $\kappa = (i_{0,0}, i_{0,1}, i_{1,0}, i_{1,1}) \in [k]^4$.

Consider the polynomial L of Lemma 13. For each pair $(a_0, a_1), (b_0, b_1) \in E(G)$, we have

$$(a_0, a_1, b_0, b_1) \text{ of } \textit{type 1} \iff L(d_G(a_0, b_0), d_G(a_0, b_1), d_G(a_1, b_0), d_G(a_1, b_1)) \geq 0.$$

Otherwise, (a_0, a_1, b_0, b_1) is of *type 2*. For each oriented edge $(b_0, b_1) \in E(G)$ and type $\tau \in \{1, 2\}$, we define

$$\text{Type}_\tau(b_0, b_1) = \{(a_0, a_1) \in E(G) \mid (a_0, a_1, b_0, b_1) \text{ of type } \tau\}. \tag{2}$$

For each $\kappa \in [k]^4$, each $(b_0, b_1) \in E_B$, and each type $\tau \in \{1, 2\}$ we define

$$\Psi_\tau(\kappa; b_0, b_1) = \sum \xi\big(\ell(a_0a_1), \ell(b_0b_1), d_G(a_0, b_0), d_G(a_0, b_1), d_G(a_1, b_0), d_G(a_1, b_1)\big), \tag{3}$$

where the sum ranges over all oriented edges $(a_0, a_1) \in E_A$ such that $(a_0, a_1) \in \text{Type}_\tau(b_0, b_1)$ and $\Phi(\kappa; a_0, a_1, b_0, b_1)$ holds.

Next, we show that we can efficiently compute $\Psi_\tau(\kappa; b_0, b_1)$ for all edges $(b_0, b_1) \in E_B$:

▶ **Lemma 14.** *Consider a fixed type $\tau \in \{1, 2\}$ and indices $\kappa \in [k]^4$. We can compute the values $\Psi_\tau(\kappa; b_0, b_1)$ for all $(b_0, b_1) \in E_B$ in $O(m2^{4k-3}B(m, 4k-3))$ time.*

▶ **Lemma 15.** *We can compute $\iint_{p \in \mathcal{G}(E_A), q \in \mathcal{G}(E_B)} d_{\mathcal{G}}(p, q)\, dp\, dq$ in $O(m2^{4k-3}k^4B(m, 4k-3))$ time.*

4.3 Global Mean

Let G be a graph with n vertices and m edges defining the continuous graph $\mathcal{G}$. Let H be a subgraph of G defining a continuous graph $\mathcal{H} \subseteq \mathcal{G}$. We use a divide-and-conquer approach to compute $\text{sumdist}(\mathcal{H}, \mathcal{G})$. The approach is very similar to that in Section 3.3 for the diameter, and thus we only emphasize the differences.

Let (A, B, S) be a separation in G. We use the notation defined before Theorem 11 introducing for $X \in \{A, B\}$, and the graphs $\mathcal{H}_X$ and $\mathcal{G}'_X$. We have

$$\begin{aligned}\text{sumdist}(\mathcal{H}, \mathcal{G}) = &\iint\limits_{p \in \mathcal{G}(E_A), q \in \mathcal{G}(E_B)} d_{\mathcal{G}}(p, q)\, dp\, dq - \iint\limits_{p, q \in \mathcal{G}(H[S])} d_{\mathcal{G}}(p, q)\, dp\, dq \\ &+ \text{sumdist}(\mathcal{H}_A, \mathcal{G}'_A) + \text{sumdist}(\mathcal{H}_B, \mathcal{G}'_B).\end{aligned} \tag{4}$$

The first term can be computed using Lemma 15, which depends on $|S|$. The second term can be computed in $O(km \log n + k^2)$ time because, after computing the distances from S, it is a problem of size $O(k^2)$. Dividing $\text{sumdist}(\mathcal{H}, \mathcal{G})$ by $\ell(\mathcal{H})^2$, we obtain $\text{mean}(\mathcal{H}, \mathcal{G})$. As in Section 3.3, we have two regimes.

▶ **Theorem 16.** *Let $k \geq 2$ be an integer constant. Let G be a graph with n vertices, treewidth at most k, nonnegative edge-lengths, and let $\mathcal{G}$ be the corresponding continuous graph. Let H be a subgraph of G and let $\mathcal{H} \subseteq \mathcal{G}$ be the corresponding continuous subgraph. We can compute $\text{mean}(\mathcal{H}, \mathcal{G})$ in $O(n \log^{4k-2} n)$ time.*

▶ **Theorem 17.** *Let G be a graph with n vertices, treewidth at most k, nonnegative edge-lengths, and let $\mathcal{G}$ be the corresponding continuous graph. Let H be a subgraph of G and let $\mathcal{H} \subseteq \mathcal{G}$ be the corresponding continuous subgraph. We can compute $\text{mean}(\mathcal{H}, \mathcal{G})$ in $n^{1+\varepsilon}2^{O(k)}$ time, for any fixed $\varepsilon > 0$.*

5 Conclusion

We presented the first subquadratic algorithms to compute the diameter and the mean distance in continuous graphs, for two non-trivial graph classes. We expect that the approach for graphs parameterized by the treewidth can be adapted for computing other statistics defined by the distance between two points selected at random in a continuous subgraph $\mathcal{H} \subseteq \mathcal{G}$, like a cumulative density function (CDF) and higher moments:

$$\text{for given } \delta, \text{ compute } \mathrm{CDF}(\delta, \mathcal{H}, \mathcal{G}) = \frac{1}{\ell(\mathcal{H})^2} \iint_{p,q \in \mathcal{H}} \mathbb{1}[d_{\mathcal{G}}(p,q) \leq \delta] \, dp \, dq,$$

$$\text{median distance:} \quad \sup\left\{\delta \in \mathbb{R}_{\geq 0} \mid \mathrm{CDF}(\delta, \mathcal{H}, \mathcal{G}) \leq 1/2\right\},$$

$$\text{higher moments, such as } \frac{1}{\ell(\mathcal{H})^2} \iint_{p,q \in \mathcal{H}} \left(d_{\mathcal{G}}(p,q)\right)^2 dp \, dq.$$

The main open question stemming from our work is whether our approach can be adapted to work in subquadratic time for arbitrary planar graphs. However, as already mentioned in the introduction, this requires dynamic trees with a set of suitable operations that – at the moment – seem to be out of reach.

References

1 A. Abboud, V. Vassilevska Williams, and J. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proc. 27th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.

2 S. Won Bae, M. De Berg, O. Cheong, J. Gudmundsson, and C. Levcopoulos. Shortcuts for the circle. *Comput. Geom. Theory Appl.*, 79:37–54, 2019. doi:10.1016/J.COMGEO.2019.01.006.

3 L. N. Baptista, J. B. Kennedy, and D. Mugnolo. Mean distance on metric graphs. *The Journal of Geometric Analysis*, 34(137):1–25, 2024. doi:10.1007/s12220-024-01574-0.

4 P. Bergé, G. Ducoffe, and M. Habib. Subquadratic-time algorithm for the diameter and all eccentricities on median graphs. *Theory of Computing Systems*, 68(1):144–193, 2024. doi:10.1007/S00224-023-10153-9.

5 P. Bergé, G. Ducoffe, and M. Habib. Quasilinear-time eccentricities computation, and more, on median graphs. In *Proc. 36th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 1679–1704. SIAM, 2025. doi:10.1137/1.9781611978322.52.

6 H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

7 H. L. Bodlaender, P. Grønås Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. doi:10.1137/130947374.

8 K. Bringmann, T. Husfeldt, and M. Magnusson. Multivariate analysis of orthogonal range searching and graph distances. *Algorithmica*, 82:2292–2315, 2020. doi:10.1007/S00453-020-00680-Z.

9 S. Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. *ACM Trans. Algorithms*, 15(2):21:1–21:38, 2019. doi:10.1145/3218821.

10 S. Cabello. Computing the inverse geodesic length in planar graphs and graphs of bounded treewidth. *ACM Trans. Algorithms*, 18(2):14:1–14:26, 2022. doi:10.1145/3501303.

11 S. Cabello, D. Garijo, A. Kalb, F. Klute, I. Parada, and R. I. Silveira. Algorithms for distance problems in continuous graphs, 2025. arXiv:2503.07769.

12 S. Cabello and C. Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. *Comput. Geom. Theory Appl.*, 42(9):815–824, 2009. doi:10.1016/j.comgeo.2009.02.001.

13 J. Cáceres, D. Garijo, A. González, A. Márquez, M. L. Puertas, and P. Ribeiro. Shortcut sets for the locus of plane Euclidean networks. *Applied Mathematics and Computation*, 334:192–205, 2018. doi:10.1016/j.amc.2018.04.010.

14 J.-L. De Carufel, C. Grimm, A. Maheshwari, S. Schirra, and M. H. M. Smid. Minimizing the continuous diameter when augmenting a geometric tree with a shortcut. *Comput. Geom. Theory Appl.*, 89:101631, 2020. doi:10.1016/J.COMGEO.2020.101631.

15 J.-L. De Carufel, C. Grimm, A. Maheshwari, and M. H. M. Smid. Minimizing the continuous diameter when augmenting paths and cycles with shortcuts. In *Proc. 15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT'16)*, volume 53 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.SWAT.2016.27.

16 C. E. Chen and R. S. Garfinkel. The generalized diameter of a graph. *Networks*, 12(3):335–340, 1982. doi:10.1002/net.3230120310.

17 G. Ducoffe. Eccentricity queries and beyond using hub labels. *Theoretical Computer Science*, 930(21):128–141, 2022. doi:10.1016/j.tcs.2022.07.017.

18 G. Ducoffe, M. Habib, and L. Viennot. Diameter, eccentricities and distance oracle computations on h-minor free graphs and graphs of bounded (distance) vapnik–chervonenkis dimension. *SIAM Journal on Computing*, 51(5):1506–1534, 2022. doi:10.1137/20m136551x.

19 L. Friedlander. Genericity of simple eigenvalues for a metric graph. *Israel Journal of Mathematics*, 146:149–156, 2005. doi:10.1007/BF02773531.

20 D. Garijo, A. Márquez, N. Rodríguez, and R. I. Silveira. Computing optimal shortcuts for networks. *Eur. J. Oper. Res.*, 279(1):26–37, 2019. doi:10.1016/j.ejor.2019.05.018.

21 D. Garijo, A. Márquez, and R. I. Silveira. Continuous mean distance of a weighted graph. *Results in Mathematics*, 78(139):1–36, 2023. doi:10.1007/s00025-023-01902-w.

22 P. Gawrychowski, H. Kaplan, S. Mozes, M. Sharir, and O. Weimann. Voronoi diagrams on planar graphs, and computing the diameter in deterministic $O(n^{5/3})$ time. *SIAM J. Comput.*, 50(2):509–554, 2021. doi:10.1137/18M1193402.

23 J. Gudmundsson and Y. Sha. Augmenting graphs to minimize the radius. *Comput. Geom. Theory Appl.*, 114(101996):1–14, 2023. doi:10.1016/j.comgeo.2023.101996.

24 J. Gudmundsson and S. Wong. Improving the dilation of a metric graph by adding edges. *ACM Trans. Algorithms*, 18(3-Article 20):1–14, 2022. doi:10.1145/3517807.

25 S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12(3):450–459, 1964.

26 H. Le and C. Wulff-Nilsen. Vc set systems in minor-free (di)graphs and applications. In *Proc. 35th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 5332–5360. SIAM, 2024. doi:10.1137/1.9781611977912.192.

27 G. Lumer. Connecting of local operators and evolution equations on networks. In *Potential Theory Copenhagen 1979*, pages 219–234. Springer-Verlag, 1980. doi:10.1007/BFb0086338.

28 L. Monier. Combinatorial solutions of multidimensional divide-and-conquer recurrences. *Algorithms*, 1(1):60–74, 1980. doi:10.1016/0196-6774(80)90005-X.

29 L. Roditty and V. Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proc. 45th Annu. ACM Sympos. Theory Comput. (STOC)*, pages 515–524. ACM, 2013. doi:10.1145/2488608.2488673.

Quantum Speedups for Polynomial-Time Dynamic Programming Algorithms

Susanna Caroppo
Roma Tre University, Rome, Italy

Giordano Da Lozzo
Roma Tre University, Rome, Italy

Giuseppe Di Battista
Roma Tre University, Rome, Italy

Michael T. Goodrich
University of California, Irvine, CA, USA

Martin Nöllenburg
TU Wien, Austria

Abstract

We introduce a quantum dynamic programming framework that allows us to directly extend to the quantum realm a large body of classical dynamic programming algorithms. The corresponding ***quantum dynamic programming algorithms*** retain the same space complexity as their classical counterpart, while achieving a computational speedup. For a combinatorial (*search* or *optimization*) problem $\mathcal{P}$ and an instance I of $\mathcal{P}$, such a speedup can be expressed in terms of the average degree δ of the dependency digraph $G_{\mathcal{P}}(I)$ of I, determined by a recursive formulation of $\mathcal{P}$. The nodes of this graph are the subproblems of $\mathcal{P}$ induced by I and its arcs are directed from each subproblem to those on whose solution it relies. In particular, our framework allows us to solve the considered problems in $\tilde{O}(|V(G_{\mathcal{P}}(I))|\sqrt{\delta})$ time. As an example, we obtain a quantum version of the Bellman-Ford algorithm for computing shortest paths from a single source vertex to all the other vertices in a weighted n-vertex digraph with m edges that runs in $\tilde{O}(n\sqrt{nm})$ time, which improves the best known classical upper bound when $m \in \Omega(n^{1.4})$.

2012 ACM Subject Classification Theory of computation → Dynamic programming; Theory of computation → Quantum complexity theory

Keywords and phrases Dynamic Programming, Quantum Algorithms, Quantum Random Access Memory

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.14

Related Version *Full Version*: `https://arxiv.org/abs/2507.00823` [16]

Funding This research was supported, in part, by MUR of Italy (PRIN Project no. 2022ME9Z78 – NextGRAAL and PRIN Project no. 2022TS4Y3N – EXPAND), and the U.S. NSF under grant 2212129.

1 Introduction

Quantum computing represents a paradigm shift in computation, leveraging the unique principles of quantum mechanics – superposition, entanglement, and interference – to solve problems that are intractable for classical computers. These principles allow quantum algorithms to achieve significant speedups for tasks such as factoring large numbers [54], searching unsorted databases [39], simulating complex physical systems [34], computational geometry [2, 5, 7, 29], and graph drawing [14, 15, 27, 28]. Classical computing has introduced fundamental algorithmic design paradigms that enable the efficient solution of combinatorial problems. Among these, for problems that exhibit a recursive structure, *dynamic programming*

© Susanna Caroppo, Giordano Da Lozzo, Giuseppe Di Battista, Michael T. Goodrich, and Martin Nöllenburg;
licensed under Creative Commons License CC-BY 4.0

and *divide and conquer*, stand out predominantly for their efficiency and wide applicability. In this research, we introduce a framework that extends classical dynamic programming algorithms [9, 10, 11, 20, 21, 24, 33, 40, 41, 42, 43, 47, 49] to faster quantum counterparts. This framework applies to combinatorial search and optimization problems tackled by computing dynamic programming tables.

Dynamic Programming. Let $\mathcal{P}$ be a combinatorial problem and let I be an instance of $\mathcal{P}$ of size n. The **dynamic programming paradigm** for algorithm design can often be applied to compute a solution $\mathrm{sol}(I)$ of I for $\mathcal{P}$, if $\mathcal{P}$ satisfies the *optimal substructure property*, i.e., an optimal solution for I can be decomposed into (interchangeable) optimal solutions for subinstances of I, and the recursive computations of solutions for larger instances require solutions for *overlapping subproblems* – unlike most divide-and-conquer algorithms, in which the subproblems do not overlap. The underlying idea of (bottom-up) dynamic programming is to create a table D, which stores the (values of) optimal solutions for all relevant subinstances of I, starting from the smallest ones and then computing optimal solutions for larger and larger subinstances by suitably combining the existing solutions of smaller subinstances. Often, the construction of $\mathrm{sol}(I)$, given D, is a simple task[1]. Therefore, the time and space complexity for solving $\mathcal{P}$ are asymptotically bounded by those of constructing and storing D, respectively. In particular, the construction time of D can be easily upper bounded by multiplying its size by the time required to recursively compute each table entry. For instance, a table D of size n^2 and a linear-time computation for each entry yield a running time of $O(n^3)$.

Quantum Dynamic Programming. Quantum dynamic programming algorithms have recently attracted considerable interest in the exponential-time regime (which we review later). Surprisingly, instead, polynomial-time quantum algorithms have not yet received as much attention, despite the potential for significant speedups in practical applications. Khadiev and Safina [45] proposed polynomial-time quantum dynamic programming algorithms for solving problems on directed acyclic graphs (DAGs). Furrow [30] gives a polynomial-time quantum dynamic programming algorithm for the Coin Change problem and for the Maximum Subarray Sum[2] problem.

The pioneering work by Ambainis et al. [4] introduced two quantum dynamic programming frameworks that provide a novel approach for speeding up some **exponential-time** classical dynamic programming algorithms, developed to address NP-complete problems. Both frameworks use classical computation to construct partial dynamic programming tables, stored in QRAM, to be accessed via quantum search subroutines. The first, and simpler, framework addresses a subclass of set problems where the solution for a set S of size n can be determined by considering all partitions of S into two sets of sizes k and $n-k$ (for any positive integer k), and by selecting the optimal option. In this setting, a quantum advantage is obtained by suitably selecting the subset of the entries to be classically precomputed so as to balance the time needed to compute the solution (obtained by identifying optimal combinations of subsets) to the problem by performing (bounded-depth) recursive quantum

[1] The solution $\mathrm{sol}(I)$ of the actual instance I is frequently found in the "last" cell of D. Such entry is later denoted as $D[i_1^*][i_2^*]\dots[i_k^*]$.

[2] It is worth remarking that, as discussed in [30], the proposed algorithm, albeit based on computing an auxiliary table, is not solved via dynamic programming, but rather by following a greedy approach.

search primitives. For instance, this framework allows to obtain fast exponential-time quantum algorithms for Traveling Salesman [4], for One-Sided Crossing Minimization [13, 14][3], for Graph Coloring [53], and for Dynamic programming across the subset [56]. The second, more complex, framework is based on the ability of efficiently solving a *Path in the Hypercube* (PHC) problem. Given a subgraph G of the Boolean hypercube, where edges are directed from vertices of lower Hamming weight to vertices of higher Hamming weight, the problem asks to determine if there exists a directed path in G from the vertex 0^n to the vertex 1^n. In this setting, a quantum advantage is reached by combining the access to precomputed partial dynamic programming tables, with recursive applications of the quantum algorithm solving PHC. For instance, this framework allows to obtain fast exponential-time quantum algorithms for Domatic Number [6] and for Treewidth Computation [48]. Furthermore, Glos et al. [36] generalized this framework to a quantum algorithm for finding a path in n-dimensional lattice graphs.

Our contributions. While exponential-time quantum algorithms aim to tackle problems beyond the reach of classical computation, polynomial-time quantum algorithms can provide substantial efficiency gains for problems already considered tractable. In this work, we focus on polynomial-time quantum algorithms and introduce a framework that systematically extends classical dynamic programming algorithms into accelerated quantum counterparts, by harnessing quantum parallelism and amplitude amplification. Specifically, we developed quantum subroutines that integrate quantum search primitives, such as *min*, *max*, *find*, and *findAll*, within a dynamic programming context. These subroutines construct a superposition enabling access to the specific subset of previously computed entries, stored in a Quantum Random Access Memory (QRAM), that is needed for computing the currently-considered entry. This, in turn, enables us to harness the full potential of quantum search primitives by restricting the search space to the relevant candidates.

To demonstrate the broad usability of our framework, we apply it to several well-known problems for which the best worst-case classical algorithms rely on dynamic programming; see Table 1. For space reasons, in this paper, we only discuss in detail the application of the framework to the Single-Source Shortest Path and Membership in Context-Free Language problems. The applications to the remaining problems in Table 1 are presented in the full version of the paper [16].

2 Preliminaries

In this section, we introduce preliminary notation and definitions.

Notation. Given two k-tuples of integers, $\langle a_1, a_2, \ldots, a_k\rangle$ and $\langle b_1, b_2, \ldots, b_k\rangle$, we say that $\langle a_1, a_2, \ldots, a_k\rangle$ *lexicographically precedes* $\langle b_1, b_2, \ldots, b_k\rangle$, denoted by $\langle a_1, a_2, \ldots, a_k\rangle \prec \langle b_1, b_2, \ldots, b_k\rangle$, if there exists an index $j \in \{1, 2, \ldots, k\}$ such that $a_i = b_i$ for all $i < j$, and $a_j < b_j$. The relation $\prec$ defines a total order among the k-tuples of integers. Given a directed graph $G = (V, E)$, we denote by $\deg(v)$ the *degree* of a vertex v of G, that is, the number of arcs in E having v as their tail or head. Moreover, we denote by $\deg_{out}(v)$ the *outdegree* of a vertex v of G, that is, the number of arcs in E having v as their tail. The average degree of

[3] A quantum algorithm for Two-Sided Crossing Minimization, solely based on Grover's search, has been presented in [17, 15].

Table 1 Comparison of classical and quantum time complexities, classified based on the operator $\mathrm{op} \in \{\min, \max, \mathrm{find}, \mathrm{findAll}\}$ in their recursive fomulation.

Problem	op	Classical Complexity	Quantum Complexity
Minimum-Weight Triangulation of Convex Polygon	min	$O(n^3)$ [49]	$\tilde{O}(n^2\sqrt{n})$
All-Pairs Shortest Paths	min	$O(n^3 \log\log n/\log^2 n)$ [40]	$\tilde{O}(n^2\sqrt{n}\log n)$
Single-Source Shortest Paths Section 5.1	min	$\tilde{O}(mn^{4/5})$ [42]	$\tilde{O}(n\sqrt{nm})$
Multi-Criteria Boundary Labeling	min	*po*-leaders $O(n^3)$ [10, 11]	$\tilde{O}(n^2\sqrt{n})$
	min	*do*-leaders $O(n^5)$ [10, 11]	$\tilde{O}(n^4\sqrt{n})$
Segmented Least Squares	min	$O(n^2)$ [9, 47]	$\tilde{O}(n\sqrt{n})$
RNA Secondary Structure	max	$O(n^3)$ [47]	$\tilde{O}(n^2\sqrt{n})$
Rod Cutting	max	$O(n^2)$ [20]	$\tilde{O}(n\sqrt{n})$
Largest Divisible Subset in Array	max	$O(n^2)$ [25]	$\tilde{O}(n\sqrt{n})$
Unbounded Knapsack	max	$O(Wn)$ [21]	$\tilde{O}(W\sqrt{n})$
Viterbi Path Problem	max	$O(T \times \lvert S\rvert^2)$ [43]	$\tilde{O}(T \times \lvert S\rvert \times \sqrt{\lvert S\rvert})$
Text Segmentation	find	$O(n^2)$ [24]	$\tilde{O}(n\sqrt{n})$
Membership in Context-Free Language (CYK) Section 5.2	findAll	$O(n^{2.37})$ [58]	$\tilde{O}(n^2\sqrt{n})$

G is $\delta = \frac{1}{n}\sum_{v\in V(G)} \deg(v) = \frac{2m}{n}$. In order to simplify the notation, we use $[h]$ to denote the set $\{0, \ldots, h-1\}$, where h is a positive integer. Also, given positive integers a and b, we denote $\lceil \frac{a}{b} \rceil$ as $\frac{a}{b}$ and $\lceil \log a \rceil$ as $\log a$. If $f(n) = O(\log^c n)$ for some constant c, we write $f(n) = polylog(n)$. In case $f(n) = O(n^d polylog(n))$ for some constant d, we use the notation $f(n) = \tilde{O}(n^d)$ (see, e.g., [59]).

Combinatorial problems. A *combinatorial search problem* $\mathcal{P}$ is a triple $\langle \Lambda, S, R\rangle$, where:

- Λ is the set of *instances* of $\mathcal{P}$;
- S is the set of *solutions* of $\mathcal{P}$; and
- $R \subseteq \Lambda \times S$ is a binary relation that associates each instance $I \in \Lambda$ with a set $SOL(I) = \{s \in S : (I, s) \in R\}$. The elements in $SOL(I)$ are the *feasible solutions* of $\mathcal{P}$ for I.

A *combinatorial optimization problem* $\mathcal{P}$ is a quintuple $\langle \Lambda, S, R, f_{\mathcal{P}}, g\rangle$, where:

- $\langle \Lambda, S, R\rangle$ is a combinatorial search problem,
- $f_{opt} : S \to Y$ is the *optimization function*, where Y is a totally ordered set (usually $Y \in \{\mathbb{N}, \mathbb{R}\}$);
- $g : 2^S \to S$ is the *comparator function*, with $g \in \{\min, \max\}$.

An *optimal solution* of $\mathcal{P}$ for I is any feasible solution s^* in $SOL(I)$ such that $s^* = \arg\min_{s\in SOL(I)} f_{opt}(s)$, if $g = \min$, and $s^* = \arg\max_{s\in SOL(I)} f_{opt}(s)$, if $g = \max$. For ease of notation, in the following, we denote by $\mathrm{sol}(I)$, both a feasible solution of a combinatorial search problem and an optimal solution of a combinatorial optimization problem. Also, we refer to combinatorial search/optimization problems simply as *combinatorial problems*.

Dynamic programming. Let $\mathcal{P}$ be a combinatorial problem and let I be an instance of $\mathcal{P}$. A (bottom-up) dynamic programming algorithm for $\mathcal{P}$ can be implemented by executing the following steps:

Dynamic Programming Steps:

Table Setup: Determine integers $d_1, d_2, \ldots, d_k$ that depend on I and create a table D of dimension $d_1 \times d_2 \times \ldots \times d_k$. Initialize some "easy" entries of D (Base Case) directly using information from I and initialize the remaining "difficult" entries with a default value representing the fact that such entries have not yet acquired their final value.

Table Update: Compute the "difficult" entries of D (Recursive Case) by evaluating a **recursive formula** that expresses each entry $D[i_1][i_2]\ldots[i_k]$ in terms of a subset of the (already computed) entries $D[j_1][j_2]\ldots[j_k]$ corresponding to optimal solutions for structurally related subproblems such that $(j_1, j_2 \ldots, j_k) \prec (i_1, i_2 \ldots, i_k)$.

Solution Retrieval: Return the value contained in a specific entry of $D[i_1^*][i_2^*]\ldots[i_k^*]$ (whose position $\langle i_1^*, i_2^*, \ldots, i_k^*\rangle$ in D depends on I and $\mathcal{P}$) or retrieve a solution of I by inspecting D (usually exploiting the information contained in $D[i_1^*][i_2^*]\ldots[i_k^*]$).

Whereas the **efficiency** of this algorithm design paradigm lies in the fact that solutions of *overlapping problems* are stored in the table and hence need to be computed only once, its **correctness** lies in the fact that "difficult" entries can be computed by exploiting the values of previously computed ("easy" and "difficult") entries (by the *optimal substructure property*). Clearly, the **Table Update** is the most interesting and challenging step in the overall approach. Fortunately, many optimization problems $\mathcal{P}$, including those considered in this paper, naturally exhibit a simple recursive formulation for the entries of their dynamic programming table D. Consider an entry $D[i_1][i_2]\ldots[i_k]$ of D. Let $S_{i_1,i_2,\ldots,i_k}$ be the *dependency set* of $D[i_1][i_2]\ldots[i_k]$, composed of the indices of the entries of D on which the computation of the entry $D[i_1][i_2]\ldots[i_k]$ depends. Observe that $S_{i_1,i_2,\ldots,i_k}$ is some subset of $\mathbb{N}^k$ such that for each entry $\langle j_1, j_2, \ldots, j_k\rangle$ in $S_{i_1,i_2,\ldots,i_k}$ we have that $\langle j_1, j_2, \ldots, j_k\rangle \prec \langle i_1, i_2, \ldots, i_k\rangle$. Then, the recursive formula for $D[i_1][i_2]\ldots[i_k]$ is of the form:

$$D[i_1][i_2]\ldots[i_k] = \mathrm{op}_{\mathcal{X}\in\mathcal{C}_{i_1,i_2,\ldots,i_k}} f_{\mathcal{P}}(i_1, i_2, \ldots, i_k, \mathcal{X}), \tag{1}$$

where: **(1)** $\mathrm{op} \in \{\min, \max, \mathrm{find}, \mathrm{findAll}\}$; **(2)** $C_{i_1,i_2,\ldots,i_k}$ is a set, called *generating set*, composed of h-element subsets of $S_{i_1,i_2,\ldots,i_k}$, where each subset provides the input to construct a particular candidate value for $D[i_1][i_2]\ldots[i_k]$ (where h is an integer constant, called *dependency index*, determined by $\mathcal{P}$, which specifies the number of subinstances into which each instance is divided in the recursive definition); **(3)** $f_{\mathcal{P}}$ is a function specific for problem $\mathcal{P}$ that computes a candidate value for $D[i_1][i_2]\ldots[i_k]$, assuming that all entries of D with indices in $S_{i_1,i_2,\ldots,i_k}$ have already been computed.

▶ Remark 1. The *optimal substructure property* of a problem $\mathcal{P}$ is formally captured by the dependency set $S_{i_1,i_2,\ldots,i_k}$, whose entries $\langle j_1, j_2, \ldots, j_k\rangle$ must satisfy $\langle j_1, j_2, \ldots, j_k\rangle \prec \langle i_1, i_2, \ldots, i_k\rangle$.

In most problems, the dependency index is a small integer, usually equal to 1 or 2. A textbook example of a problem fitting Equation (1) with $h = 1$ is the Coin Change problem [47]. Given a set of positive integer coin denominations $c_1 < c_2 < \cdots < c_r$ and a target sum T, the goal is to achieve T using the fewest possible number of coins, assuming an unlimited supply of each denomination, or determine if it is not possible to obtain T. Let D be a dynamic programming table of size $T + 1$, whose entries $D[i]$ represent the minimum number of coins needed to make up the sum i, with $D[i] = \infty$ if it is not possible. The base case of the dynamic programming approach is $D[0] = 0$. For the recursive case, to compute

$D[i]$ for $i > 0$, we have to consider all possible choices for the first coin. Once the first coin is selected, the remaining amount must be reached optimally. This can be expresses by the recursive formula

$$D[i] = \begin{cases} 0, & \text{if } i = 0 \\ \min\limits_{j:c_j \leq i}(1 + D[i - c_j]), & \text{if } i > 0 \end{cases} \tag{2}$$

Clearly, the recursive case of Equation (2) matches the pattern of Equation (1). In fact, we can set $S_i = \{j : c_j \leq i\}$, $C_i = \{\{j\} : c_j \leq i\}$, $h = 1$, and $f_{\mathcal{P}}(i, \{j\}) = 1 + D[i - c_j]$.

A notable example of a problem fitting Equation (1) with $h = 2$ is the Matrix Chain Multiplication problem. Given a sequence of n matrices, $A_1, A_2, \ldots, A_n$, and their dimensions $p_0, p_1, p_2, \ldots, p_n$, where for $i = 1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$, the problem asks to determine the order of matrix multiplications that minimizes the total number of scalar multiplications needed to obtain $A_1 \times A_2 \times \ldots \times A_n$. Note that, in the Matrix Chain Multiplication problem, we are not actually multiplying matrices; instead, the goal is only to determine an order for multiplying matrices that has the lowest cost. Recall that matrix multiplication is associative, therefore we aim at grouping the above multiplications to minimize the total number of scalar multiplications.

Let D be a dynamic programming table of size $n \times n$, whose entries $D[i][j]$, with $1 \leq i \leq j \leq n$, store the minimum number of scalar multiplications needed to compute the product $A_i \times A_{i+1} \times \ldots \times A_j$. The base case of the dynamic programming approach is $D[i][i] = 0$. For the recursive case, to compute $D[i][j]$ for $j > i$, we have to consider all possible choices to split the product at a matrix A_k, with $i \leq k < j$. This can be expressed by the recursive formula:

$$D[i][j] = \begin{cases} 0, & \text{if } i = j \\ \min\limits_{i \leq k < j}(D[i][k] + D[k+1][j] + p_{i-1}p_k p_j), & \text{if } i < j \end{cases} \tag{3}$$

Clearly, the recursive case of Equation (3) matches the pattern of Equation (1). In fact, we can set $S_{i,j} = \{(i, k), (k + 1, j) : i \leq k < j\}$, $C_{i,j} = \{\{(i, k), (k + 1, j)\} : i \leq k < j\}$, $h = 2$, and $f_{\mathcal{P}}(i, j, \{(i, k), (k + 1, j)\}) = D[i][k] + D[k + 1][j] + p_{i-1}p_k p_j$.

3 Quantum Tools

In this section, we provide the reader with the quantum primitives needed in this research. For a comprehensive introduction to the quantum computing field see, e.g., Nielsen and Chuang [50], and Aaronson [1].

Qubits are the fundamental units of quantum computing. They differ from classical bits in that they can exist in a superposition of the two classical states **0** and **1**. This unique characteristic enables quantum computers to perform, in parallel, multiple computations, allowing in some cases to achieve a substantial (and sometimes exponential) speedup over classical systems. Mathematically, a qubit is represented in a Hilbert space as a two-dimensional vector[4] $|\psi\rangle = \binom{\alpha}{\beta} \in \mathbb{C}^2$, that is, $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, where $|0\rangle = \binom{1}{0}$ and $|1\rangle = \binom{0}{1}$ are the two orthonormal basis states and the complex coefficients α and β are the *amplitudes* of such states. The likelihood of measuring the qubit in state $|0\rangle$ or $|1\rangle$ is given by $|\alpha|^2$

[4] In Dirac's notation, a *ket* such as $|v\rangle$, where v is an arbitrary label, represents a vector corresponding to a quantum state.

and $|\beta|^2$, respectively. Therefore, α and β must satisfy the **normalization condition** $|\alpha|^2+|\beta|^2= 1$, ensuring that the total probability remains 1. A *quantum state* over n qubits is a unit vector in the Hilbert space $\mathbb{C}^{2^n}$. The *computational basis* $\{|j\rangle\}_{j\in[2^n]}$ consists of quantum states, where $|j\rangle$ is the unit vector with a 1 in the j-th index and 0 elsewhere. A computational basis state $|j\rangle$ can be interpreted as a classical bit string j, allowing quantum systems to simulate classical algorithms. Any quantum state can be expressed as a weighted sum (or *superposition*) of these basis states:

$$|\Psi\rangle = \sum_{j=0}^{2^n-1} \alpha_j |j\rangle ,$$

where the amplitudes satisfy the normalization condition $\sum_{j=0}^{2^n-1}|\alpha_j|^2= 1$. If at least two coefficients α_j in the above expression are nonzero, the state is said to be in *superposition*. When measuring $|\Psi\rangle$, the state collapses to $|j\rangle$ with probability $|\alpha_j|^2$.

In this paper, we focus on quantum computations performed in the **circuit model of computation**. In this model, quantum algorithms are specified by *quantum circuits*, obtained by composing quantum gates, that perform specific quantum computations. The process begins with the initialization of qubits, followed by the application of gate operations that modify their states. Finally, the circuit's output is obtained by measuring the qubits. This operation collapses their quantum states into classical binary outcomes. *Quantum gates* are the fundamental building blocks of quantum circuits, analogous to classical logic gates in traditional computing. They are used to manipulate quantum states while preserving key quantum properties like superposition and entanglement. Specifically, a quantum gate performs a linear transformation on its input quantum state, meaning that a superposition of states is mapped to the corresponding superposition of their images. In particular, any such a transformation U must be *unitary*, satisfying the condition $\mathbb{I} = U^\dagger U = UU^\dagger$, where $\mathbb{I}$ denotes the identity matrix and $U^\dagger$ denotes the transpose conjugate of U.

As a consequence, quantum computation is inherently *reversible* as long as no measurement is performed. That is, given an output quantum state $|\phi\rangle$, obtained by applying U to an initial quantum state $|\psi\rangle$, the original state can be fully recovered by applying $U^{-1} = U^\dagger$ to $|\phi\rangle$. An important quantum gate (often used in the initialization of qubits) is the Hadamard gate $H = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$, which can be used a preliminary step to transform the $|0\rangle$ state into to a uniform state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ of the two basis states.

Quantum Random Access Memory (QRAM) is the "quantum analog" of conventional RAM, designed to store and access data in a quantum format. Like RAM, QRAM consists of three main components: an input (or address) register a, an output (or data) register d, and memory arrays. However, unlike traditional RAM, QRAM's input and output registers are composed of qubits rather than classical bits, while the memory arrays can be either classical or quantum, depending on the application [35]. A key feature of QRAM lies in its method of memory access. Instead of retrieving data from a single memory location at a time, QRAM leverages quantum superposition to access multiple memory locations simultaneously. Specifically, while a classical RAM uses n bits to randomly access one of $N = 2^n$ distinct memory cells, a QRAM uses n qubits to address a quantum superposition of all N memory cells simultaneously. When a quantum computer requires access to a superposition of memory cells, the address register[5] a must hold a superposition of addresses,

[5] A set comprising multiple qubits is a *register*. A quantum register a with n qubits $|q_i\rangle$, with $i \in [n]$, is denoted as a tensor product $|\psi\rangle_a = |q_0\rangle \otimes |q_1\rangle \otimes \cdots \otimes |q_{n-1}\rangle$.

Algorithm 1 Procedure StatePrep prepares a uniform superposition over the set $C_{i_1,i_2,\ldots,i_k}$.

procedure StatePrep($i_1 i_2 \ldots i_k$)
  input: Registers $|i_1 i_2 \ldots i_k\rangle$ and an ℓ-bit zero register $|0^\ell\rangle$
  output: Uniform superposition state $|\Psi_{i_1,i_2,\ldots,i_k}\rangle$
  Apply the quantum circuit U_{f_C} to determine $\lambda_{i_1,i_2,\ldots,i_k} = |C_{i_1,i_2,\ldots,i_k}|$
  Prepare the state $|i_1 i_2 \ldots i_k\rangle\,|\lambda_{i_1,i_2,\ldots,i_k}\rangle$
  Introduce an additional $\log(\lambda_{i_1,i_2,\ldots,i_k})$-bit zero register $|0^{\log \lambda_{i_1,i_2,\ldots,i_k}}\rangle$
  Apply Hadamard gates to the qubits of the last register to prepare the uniform superposition:

$$|\Psi_{i_1,i_2,\ldots,i_k}\rangle = \frac{1}{\sqrt{\lambda_{i_1,i_2,\ldots,i_k}}}\,|i_1, i_2, \ldots, i_k\rangle\,|\lambda_{i_1,i_2,\ldots,i_k}\rangle \sum_{u=0}^{\lambda_{i_1,i_2,\ldots,i_k}-1} |u\rangle$$

  return $|\Psi_{i_1,i_2,\ldots,i_k}\rangle$
end procedure

represented as $\sum_j \alpha_j\,|j\rangle_a$. In response, the QRAM returns a corresponding superposition of data in the data register d, ensuring that the retrieved data remains correlated with the address register:

$$\sum_j \alpha_j\,|j\rangle_a\,|0^k\rangle_d \overset{QRAM}{\longrightarrow} \sum_j \alpha_j\,|j\rangle_a\,|\delta_j\rangle_d\,,$$

where δ_j represents the content of the j-th memory cell, k bits suffice to classically encode the content of any memory cell, and $|0^k\rangle$ denotes the quantum basis state composed of k qubits set to $|0\rangle$. QRAM plays a crucial role in several quantum algorithms, such as quantum searching [39], minimum/maximum finding [23], counting [12], period finding and discrete logarithm [54], to cite a few. In particular, the use of QRAM enables us to exploit quantum search primitives that involve condition checking on data stored in random access memory. Specifically, the QRAM may be used by an oracle to check conditions based on the data stored in memory, marking the superposition states that correspond to feasible or optimal solutions [35, 51, 60, 62].

We now describe the **quantum subroutines** used in our algorithms. Observe that every function f implemented as a classical circuit can be approximated with arbitrary accuracy, using a discrete set of quantum gates, by a quantum circuit with only a polylogarithmic overhead [22, 46, 50]. Therefore, in the following, we assume that every classical function f, provided as classical circuit C_f, can be passed to and accessed by our quantum procedures as a quantum version of C_f. In the remainder, for any computable function f, we use the notation U_f to denote the corresponding quantum circuit.

Let W be a ground set and let ω be an element of W. Let $bin(\omega)$ be the integer corresponding to a binary encoding of ω. For ease of notation, in the remainder, we will denote the state $|bin(\omega)\rangle$ simply as $|\omega\rangle$. In particular, given an element $\mathcal{X} \in C_{i_1,i_2,\ldots,i_k}$, we will use the state $|\mathcal{X}\rangle$.

Consider a computational problem $\mathcal{P}$ that can be solved via dynamic programming building a k-dimensional table D of size $d_1 \times \ldots \times d_k$. Recall that by $C_{i_1,i_2,\ldots,i_k}$ we denote the generating set of an entry $D[i_1][i_2]\ldots[i_k]$ of D. We let $\lambda_{i_1,i_2,\ldots,i_k} = |C_{i_1,i_2,\ldots,i_k}|$ and we let $\sigma = \sum_{i=1}^{k} \log d_i$.

Algorithm 2 Procedure TableIndexPrep for retrieving the u-th element of $C_{i_1,i_2,\ldots,i_k}$.

procedure TableIndexPrep$(i_1, i_2, \ldots, i_k, u)$
  input: Registers $|i_1, i_2, \ldots, i_k\rangle$, the register $|u\rangle$, and an $h \cdot \sigma$-bit zero register $|0^{h\cdot\sigma}\rangle$
  output: State $|E_{i_1,i_2,\ldots,i_k,u}\rangle$ encoding the u-th element $\gamma(\langle i_1, i_2, \ldots, i_k\rangle, u)$ of $C_{i_1,i_2,\ldots,i_k}$
  Apply the quantum circuit U_γ to retrieve the element $X_u = \gamma(\langle i_1, i_2, \ldots, i_k\rangle, u)$
  Store X_u in the output register, thus obtaining the state:

$$|E_{i_1 i_2 \ldots i_k,u}\rangle = |i_1 i_2 \ldots i_k\rangle\,|u\rangle\,|X_u\rangle$$

  return $|E_{i_1 i_2 \ldots i_k,u}\rangle$
end procedure

Subroutine StatePrep. The first subroutine prepares a uniform superposition $\Psi_{i_1,i_2,\ldots,i_k}$ of the integers in $[\lambda_{i_1,i_2,\ldots,i_k}]$; see the pseudocode of Algorithm 1. It assumes the existence of a classical procedure f_C that determines $\lambda_{i_1,i_2,\ldots,i_k}$ given the values $i_1, i_2, \ldots, i_k$. The subroutine starts with the register $|i_1 i_2 \ldots i_k\rangle$ and with a register storing an ℓ-bit zero string $|0^\ell\rangle$ (where the value of ℓ will be discussed later). It exploits the quantum gate U_λ to determine $\lambda_{i_1,i_2,\ldots,i_k}$ and prepares the state $|i_1 i_2 \ldots i_k\rangle\,|\lambda_{i_1,i_2,\ldots,i_k}\rangle$. Then, the subroutine takes in input an additional register storing a $\log(\lambda_{i_1,i_2,\ldots,i_k})$-bit zero string and applies Hadamard gates to each of the qubits of such a register to prepare the uniform superposition state:

$$|\Psi_{i_1,i_2,\ldots,i_k}\rangle = \frac{1}{\sqrt{\lambda_{i_1,i_2,\ldots,i_k}}}\,|i_1 i_2 \ldots i_k\rangle\,|\lambda_{i_1,i_2,\ldots,i_k}\rangle \sum_{u=0}^{\lambda_{i_1,i_2,\ldots,i_k}-1} |u\rangle$$

We use the signature `StatePrep`$(i_1, i_2, \ldots, i_k)$ to denote calls to the subroutine StatePrep.

Subroutine TableIndexPrep. The second quantum subroutine prepares the state $|E_{i_1 i_2 \ldots i_k,u}\rangle$ that correlates the integers in $[\lambda_{i_1,i_2,\ldots,i_k}]$ with the elements of $C_{i_1 i_2 \ldots i_k}$; see the pseudocode of Algorithm 2. Our quantum subroutine assumes the existence of a classical **injective function** $\gamma : \mathbb{N}^k \times \mathbb{N} \to C_{i_1 i_2 \ldots i_k}$ that maps the pair $\langle\langle i_1 i_2 \ldots i_k\rangle, u\rangle$ to an element of $C_{i_1 i_2 \ldots i_k}$. Observe that the elements of $C_{i_1,i_2,\ldots,i_k}$ admit a binary representation $bin(\mathcal{X})$ of length $h \cdot \sigma$. The subroutine starts with the register $|i_1 i_2 \ldots i_k\rangle$, the register $|u\rangle$, and with a register storing an $(h \cdot \sigma)$-bit zero string $|0^{h\cdot\sigma}\rangle$ and uses U_γ to prepare the state:

$$|E_{i_1 i_2 \ldots i_k,u}\rangle = |i_1 i_2 \ldots i_k\rangle\,|u\rangle\,|\gamma(\langle i_1, i_2, \ldots, i_k\rangle, u)\rangle\,.$$

We use the signature `TableIndexPrep`$(i_1, i_2, \ldots, i_k, u)$ to denote calls to the subroutine TableIndexPrep.

Subroutine DP. The third quantum subroutine uses either **(i)** the quantum min/max finding algorithms (QMin and QMax) due to Dürr and Høyer [23], or **(ii)** the quantum finding algorithm (QFind) due to Grover to search for an item satisfying a condition in an unsorted list [1] or **(iii)** the quantum finding algorithm (QFindAll) due to Ambainis to search for all items satisfying a condition in an unsorted list [3]. In particular, QMin and QMax allow to determine the element in a ground set of size N that minimizes/maximizes a given function in $\tilde{O}(\sqrt{N})$ time, QFind (resp. QFindAll) allows to determine an element (resp. all the elements) in a ground set of size N that satisfy a certain condition in $\tilde{O}(\sqrt{N})$ time (resp. $\sqrt{NM}$ time, where M is the number of elements satisfying the condition).

Algorithm 3 Procedure DP for computing the value $\mathrm{op}_{\mathcal{X}\in\mathcal{C}_{i_1,i_2,\ldots,i_k}} f_{\mathcal{P}}(i_1,i_2,\ldots,i_k,\mathcal{X})$. We use the signature $\texttt{QMaxMin}(i_1,i_2,\ldots,i_k,f_{\mathcal{P}},\preccurlyeq)$, $\texttt{QMaxMax}(i_1,i_2,\ldots,i_k,f_{\mathcal{P}},\max,\preccurlyeq)$, $\texttt{QFind}(i_1,i_2,\ldots,i_k,f_{\mathcal{P}})$, and $\texttt{QFindAll}(i_1,i_2,\ldots,i_k,f_{\mathcal{P}})$ to denote calls to the algorithms QMaxMin, QMaxMin, QFind, and QFindAll, respectively.

procedure $\texttt{DP}(i_1,i_2,\ldots,i_k,f_{\mathcal{P}},op,\preccurlyeq=\texttt{null})$
  input: Registers $|i_1,i_2,\ldots,i_k\rangle$, the function $f_{\mathcal{P}}$, and an operator op $\in$ {min, max, find, findAll}. Additionally, a comparator $\preccurlyeq$, if op $\in$ {min, max}.
  output: The value $\mathrm{op}_{\mathcal{X}\in\mathcal{C}_{i_1,i_2,\ldots,i_k}} f_{\mathcal{P}}(i_1,i_2,\ldots,i_k,\mathcal{X})$ to be assigned to $D[i_1][i_2]\ldots[i_k]$
  Invoke the subroutine $\texttt{StatePrep}(i_1,i_2,\ldots,i_k)$ to prepare the superposition state $|\Psi_{i_1,i_2,\ldots,i_k}\rangle$
  Apply the subroutine $\texttt{TableIndexPrep}(i_1,i_2,\ldots,i_k,u)$, in parallel to each basis state of $|\Psi_{i_1,i_2,\ldots,i_k}\rangle$, to prepare the superposition $|\Phi_{i_1,i_2,\ldots,i_k}\rangle$
  if op = min **then**
    Apply $\texttt{QMaxMin}(i_1,i_2,\ldots,i_k,f_{\mathcal{P}},\preccurlyeq)$ to $|\Phi_{i_1,i_2,\ldots,i_k}\rangle$
  else if op = max **then**
    Apply $\texttt{QMaxMax}(i_1,i_2,\ldots,i_k,f_{\mathcal{P}},\preccurlyeq)$ to $|\Phi_{i_1,i_2,\ldots,i_k}\rangle$
  else if op is find **then**
    Apply $\texttt{QFind}(i_1,i_2,\ldots,i_k,f_{\mathcal{P}})$ to $|\Phi_{i_1,i_2,\ldots,i_k}\rangle$
  else if op is findAll **then**
    Apply $\texttt{QFindAll}(i_1,i_2,\ldots,i_k,f_{\mathcal{P}})$ to $|\Phi_{i_1,i_2,\ldots,i_k}\rangle$
  end if
  return $\mathrm{op}_{\mathcal{X}\in\mathcal{C}_{i_1,i_2,\ldots,i_k}} f_{\mathcal{P}}(i_1,i_2,\ldots,i_k,\mathcal{X})$
end procedure

The subroutine (see the pseudocode of Algorithm 3) takes as input at least: **(1)** the integers $i_1,i_2,\ldots,i_k$; **(2)** the function $f_{\mathcal{P}}$; and **(3)** an operator op $\in$ {min, max, find, findAll}. Furthermore, if op $\in$ {min, max}, it additionally takes as an input a comparator, $\preccurlyeq$, to maximize (or minimize) over, that defines a total ordering of the values in the codomain of $f_{\mathcal{P}}$ (i.e., the values stored in D). First, the subroutine invokes $\texttt{StatePrep}$ $(i_1,i_2,\ldots,i_k)$ to prepare the superposition state $|\Psi_{i_1,i_2,\ldots,i_k}\rangle$ and then applies $\texttt{TableIndexPrep}$, in parallel to each basis state of $|\Psi_{i_1,i_2,\ldots,i_k}\rangle$, to prepare the superposition state:

$$|\Phi_{i_1,i_2,\ldots,i_k}\rangle = \frac{1}{\sqrt{\lambda_{i_1,i_2,\ldots,i_k}}}\,|i_1,i_2,\ldots,i_k\rangle\,|\lambda_{i_1,i_2,\ldots,i_k}\rangle\sum_{u=0}^{\lambda_{i_1,i_2,\ldots,i_k}-1}|u\rangle\,|\gamma(\langle i_1,i_2,\ldots,i_k\rangle,u)\rangle\,.$$

▶ Remark 2. The time $T'_{i_1,i_2,\ldots,i_k}$ required to prepare the state $|\Phi_{i_1,i_2,\ldots,i_k}\rangle$ is bounded by the time needed to execute gates $U_{f_{\mathcal{C}}}$ and U_{γ}.

Observe that each of the states $|i_1,i_2,\ldots,i_k\rangle\,|\lambda_{i_1,i_2,\ldots,i_k}\rangle\,|u\rangle\,|\gamma(\langle i_1,i_2,\ldots,i_k\rangle,u)\rangle$ composing $|\Phi_{i_1,i_2,\ldots,i_k}\rangle$ provides the input for computing, using $f_{\mathcal{P}}$ and the quantum search subroutines described above, a candidate value for $D[i_1][i_2]\ldots[i_k]$, that is, the registers $|i_1,i_2,\ldots,i_k\rangle$ and $|\gamma(\langle i_1,i_2,\ldots,i_k\rangle,u)\rangle$. Therefore, the subroutine proceeds by applying the algorithm QMin, QMax, QFind, or QFindAll, depending on whether the chosen operator op is equal to min (or max), find, or findAll, respectively, **only to** the candidate values for $D[i_1][i_2]\ldots[i_k]$ determined by the entries in $C_{i_1,i_2,\ldots,i_k}$.

We use the signatures $\texttt{DP}(i_1,i_2,\ldots,i_k,f_{\mathcal{P}},op)$ and $\texttt{DP}(i_1,i_2,\ldots,i_k,f_{\mathcal{P}},op,\preccurlyeq)$ to denote calls to the subroutine DP, when op $\in$ {find, findAll} or op $\in$ {min, max}, respectively.

Altogether, we obtain the following main algorithmic theorem.

▶ **Theorem 3.** *Let $\mathcal{P}$ be a combinatorial problem, let I be an instance of $\mathcal{P}$, and let n be the size of I. Also, let $T_{\mathcal{P}}$ be the time needed to compute quantumly the function $f_{\mathcal{P}}$ and let $T'_{i_1,i_2,\ldots,i_k}$ be the time needed to prepare $|\Phi_{i_1,i_2,\ldots,i_k}\rangle$. Suppose that each of the values of the entries of D can be represented using w bits with $w \in O(polylog(n))$. Then, the following holds for subroutine* `DP`*:*

- *If* op $=$ find, `DP` *determines a value of $f_{\mathcal{P}}$ for an entry $D[i_1][i_2]\ldots[i_k]$ in $\tilde{O}(T'_{i_1,i_2,\ldots,i_k} + T_{\mathcal{P}}\sqrt{\lambda_{i_1,i_2,\ldots,i_k}})$ time.*
- *If* op $=$ findAll, `DP` *determines all the M possible distinct values of $f_{\mathcal{P}}$ for an entry $D[i_1][i_2]\ldots[i_k]$ in $\tilde{O}(T'_{i_1,i_2,\ldots,i_k} + T_{\mathcal{P}}\sqrt{\lambda_{i_1,i_2,\ldots,i_k} M})$ time.*
- *If* op $\in \{\min, \max\}$*, consider some total ordering defined by $\lessdot$ of the data values in D such that comparison according to such an ordering can be performed in $O(w)$ time. Then,* `DP` *determines the minimum (or maximum) value of $f_{\mathcal{P}}$ for an entry $D[i_1][i_2]\ldots[i_k]$, under the specified ordering, in $\tilde{O}(T'_{i_1,i_2,\ldots,i_k} + T_{\mathcal{P}}\sqrt{\lambda_{i_1,i_2,\ldots,i_k}})$ time.*

4 Quantum Dynamic Programming

In this section, we describe a framework that exploits the quantum subroutines defined in Section 3 to obtain quantum speedups for many computational problems solved classically using dynamic programming algorithms.

The recursive formula that specifies how to compute the entries of the dynamic programming table D used for solving a computational problem $\mathcal{P}$ on a instance I determines a **dependency digraph** $G_{\mathcal{P}}(I)$. The nodes of $G_{\mathcal{P}}(I)$ are in one-to-one correspondence with the entries of D, i.e., the subproblems of $\mathcal{P}$ defined by I. For each node $n_{i_1,i_2,\ldots,i_k}$ of $G_{\mathcal{P}}(I)$ associated with the entry $D[i_1][i_2]\ldots[i_k]$ of D, graph $G_{\mathcal{P}}(I)$ contains an arc directed from $n_{i_1,i_2,\ldots,i_k}$ to each of the nodes $n_{j_1,j_2,\ldots,j_k}$ associated with the entries $D[j_1][j_2]\ldots[j_k]$ that occur in the recursive formula for $D[i_1][i_2]\ldots[i_k]$. These are the entries indexed by the tuples $\langle j_1, j_2, \ldots, j_k\rangle \in S_{i_1,i_2,\ldots,i_k}$. Clearly, by the optimal substructure property, $G_{\mathcal{P}}(I)$ is a directed acyclic graph.

Let $\mathcal{P}$ be a combinatorial problem and suppose that it admits a dynamic programming algorithm that relates the dependency set $S_{i_1,i_2,\ldots,i_k}$ and the generating set $C_{i_1,i_2,\ldots,i_k}$ of an entry $D[i_1][i_2]\ldots[i_k]$ as follows. For each element $\langle j_1, j_2, \ldots, j_k\rangle \in S_{i_1,i_2,\ldots,i_k}$, there exists a unique set $\mathcal{X} \in C_{i_1,i_2,\ldots,i_k}$ such that $\langle j_1, j_2, \ldots, j_k\rangle \in \mathcal{X}$. We say that a dynamic programming algorithm exhibiting the above characteristic is *simple* and that $\mathcal{P}$ is a *simple problem*. Observe that, for a simple dynamic programming algorithm, it holds that $\lambda_{i_1,i_2,\ldots,i_k} = |C_{i_1,i_2,\ldots,i_k}| = \frac{|S_{i_1,i_2,\ldots,i_k}|}{h}$. In particular, this implies that, for each node $n_{i_1,i_2,\ldots,i_k}$ of $G_{\mathcal{P}}(I)$, it holds that $\deg_{out}(n_{i_1,i_2,\ldots,i_k}) = |S_{i_1,i_2,\ldots,i_k}| = h \cdot \lambda_{i_1,i_2,\ldots,i_k}$. We have that several combinatorial problems, including those listed in Table 1, as we prove later, are simple problems.

The next lemma shows how classical simple dynamic programming algorithms can be quantumly enhanced to reduce their time complexity, while maintaining the same storage requirements as in the classical setting. Such an improvement applies to problems whose recursive formulation satisfies Equation (1). Table 1 provides an overview of the speedups obtainable via Theorem 4 for many well-known problems.

▶ **Theorem 4.** *Let $\mathcal{P}$ be a simple combinatorial problem, let I be an instance of $\mathcal{P}$, and let n be the size of I. Suppose that each of the values of the entries of D can be represented using w bits with $w \in O(polylog(n))$. Consider a simple classical dynamic programming algorithm $\mathcal{A}$ that solves $\mathcal{P}$ for I by computing each entry $D[i_i][i_2]\ldots[i_k]$ of a table D of size $d_1 \times d_2 \times \ldots \times d_k$, where the values d_i depend on I, using a recurrence formula of the same form as Equation* (1)*:*

$$D[i_1][i_2]\dots[i_k] = \mathrm{op}_{\mathcal{X}\in\mathcal{C}_{i_1,i_2,\dots,i_k}} f_{\mathcal{P}}(i_1,i_2,\dots,i_k,\mathcal{X}),$$

where $\mathrm{op} \in \{\min,\max,\mathrm{find},\mathrm{findAll}\}$. *Also, let* $T_{\mathcal{P}}$, T_{f_C} *and* T_γ *be the time needed to compute quantumly the functions* $f_{\mathcal{P}}$, f_C, *and* γ, *respectively. Finally, let* $T' = T_{f_C} + T_\gamma$, *let* M *be the maximum number of solutions of any subproblem of* $\mathcal{P}$ *for* I, *and let* δ *be the average degree of* $G_{\mathcal{P}}(I)$.

Then, there exists a quantum dynamic programming algorithm $Q_{\mathcal{A}}$ *that solves* $\mathcal{P}$ *for* I, *using QRAM, with the following time and space bounds:*

- *If* $\mathrm{op} \in \{\mathrm{find},\min,\max\}$, *then* $Q_{\mathcal{A}}$ *solves* $\mathcal{P}$ *for* I *using* $\tilde{O}(|V(G_{\mathcal{P}}(I))|(T' + T_{\mathcal{P}}\sqrt{\delta}))$ *time and* $\tilde{O}(|V(G_{\mathcal{P}}(I))|)$ *space.*
- *If* $\mathrm{op} = \mathrm{findAll}$, *then* $Q_{\mathcal{A}}$ *solves* $\mathcal{P}$ *for* I *using* $\tilde{O}(|V(G_{\mathcal{P}}(I))|(T' + T_{\mathcal{P}}\sqrt{\delta\cdot M}))$ *time and* $\tilde{O}(M|V(G_{\mathcal{P}}(I))|)$ *space.*

Proof. In the following, we describe the algorithm $Q_{\mathcal{A}}$ and prove its space and time complexity. We describe $Q_{\mathcal{A}}$ in terms of the three steps that implement the dynamic programming approach provided by $\mathcal{A}$.

Table Setup. As in the classical scenario, the algorithm starts by initializing a dynamic programming table QD of size $d_1 \times d_2 \times \ldots \times d_k$. Differently from the classical dynamic programming table D used by $\mathcal{A}$, however, the table QD is stored in QRAM. Suppose that each of the values of the entries of D (and thus of QD) can be represented using w bits. Let f_{init} be the classical function that, provided with the tuple $\langle i_1,i_2,\dots,i_k\rangle$ that addresses the entry $D[i_1][i_2]\dots[i_k]$, computes the initialization value of $D[i_1][i_2]\dots[i_k]$. To this aim, for each $\langle i_1,i_2,\dots,i_k\rangle \in [d_1]\times[d_2]\times\ldots\times[d_k]$, we provide the gate U_{init} with the address register $|i_1 i_2\dots i_k\rangle_a$ and with a data register storing a w-bit zero string $|0^w\rangle$. The application of U_{init} to these registers results in the state $|i_1 i_2\dots i_k\rangle_a |f_{init}(\langle i_1,i_2,\dots,i_k\rangle)\rangle_d$, which forms the input for a `QRAM write` operation that initializes $QD[i_1][i_2]\dots[i_k]$.

Table Update. We process the tuples $\langle i_1,i_2,\dots,i_k\rangle \in [d_1]\times[d_2]\times\ldots\times[d_k]$ in lexicographic order. For each tuple $\langle i_1,i_2,\dots,i_k\rangle$, the algorithm invokes either the quantum subroutine `DP`$(i_1,i_2,\dots,i_k,f_{\mathcal{P}},op)$ or `DP`$(i_1,i_2,\dots,i_k,f_{\mathcal{P}},op,\lessdot)$, based on whether $\mathrm{op} \in \{\mathrm{find},\mathrm{findAll}\}$ or $\mathrm{op} \in \{\min,\max\}$, respectively. This allows us to compute $\mathrm{op}_{\mathcal{X}\in\mathcal{C}_{i_1,i_2,\dots,i_k}} f_{\mathcal{P}}(i_1,i_2,\dots,i_k,\mathcal{X})$, which is stored in an output register $|d_{i_1,i_2,\dots,i_k}\rangle_o$ correlated with the address register $|i_1 i_2\dots i_k\rangle_a$. Finally, the state $|i_1 i_2\dots i_k\rangle_a |d_{i_1,i_2,\dots,i_k}\rangle_o$ forms the input for a `QRAM write` operation that updates $QD[i_1][i_2]\dots[i_k]$.

Solution Retrieval. Let $\langle i_1^*,i_2^*,\dots,i_k^*\rangle$ be the entry of D whose value determines the solution to $\mathcal{P}$. Then, a solution for $\mathcal{P}$ can be obtained by reading the entry $QD[i_1^*][i_2^*]\dots[i_k^*]$.

Next, we analyze the space complexity of $Q_{\mathcal{A}}$. As in its classic counterpart, the space complexity of $Q_{\mathcal{A}}$ is asymptotically bounded by the storage requirement of the dynamic programming table, which can be upper bounded by multiplying the number of entries of QD, the number w of bits to represent each of the values of the entries of QD (i.e., the solutions to the subproblems of $\mathcal{P}$), and the maximum number M of solutions of any subproblem of $\mathcal{P}$ for I. Also, recall that the entries of QD are in one-to-one correspondence with the nodes of $G_{\mathcal{P}}(I)$. Therefore, the space complexity of $Q_{\mathcal{A}}$ is in $O(w(|V(G_{\mathcal{P}}(I))|)$, if $\mathrm{op} = \{\mathrm{find},\min,\max\}$, and in $O((w\cdot M)|V(G_{\mathcal{P}}(I))|)$, if $\mathrm{op} = \mathrm{findAll}$.

Finally, we analyze the time complexity of $Q_{\mathcal{A}}$. Obviously, the time needed to compute function f_{init} is in $O(T_{\mathcal{P}})$. Therefore, the time complexity $\mathrm{Time}(I)$ of $Q_{\mathcal{A}}$ on input I is asymptotically bounded by the time required to execute the **Table Update** step. We now

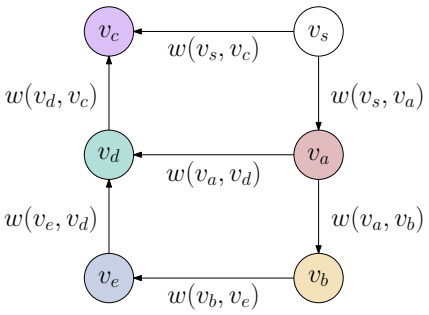
give the corresponding bound. For simplicity, we assume first that op $\in \{\text{find}, \min, \max\}$. Let $n = |V(G_{\mathcal{P}}(I))|$ and recall that δ denotes the average degree of $G_{\mathcal{P}}(I)$. In order to upper bound $\text{Time}(I)$, we proceed as follows. For $i = 1, \ldots, \log\log n$, let V_i be the subset of $V(G_{\mathcal{P}}(I))$ whose degree is between $2^{2^{i-1}}\delta$ and $2^{2^i}\delta$, and let V_0 be the subset of $V(G_{\mathcal{P}}(I))$ whose degree is less than 2δ. Recall that, by Theorem 3, the quantum subroutine DP determines a value of $f_{\mathcal{P}}$ for an entry $D[i_1][i_2]\ldots[i_k]$ in $\tilde{O}(T'_{i_1,i_2,\ldots,i_k} + T_{\mathcal{P}}\sqrt{\lambda_{i_1,i_2,\ldots,i_k}})$ time, that $T'_{i_1,i_2,\ldots,i_k} \leq T'$, and that, for each node $n_{i_1,i_2,\ldots,i_k}$ of $G_{\mathcal{P}}(I)$, it holds that $\lambda_{i_1,i_2,\ldots,i_k} \leq \deg_{out}(n_{i_1,i_2,\ldots,i_k})$ since $\mathcal{P}$ is simple. Therefore, we have:

$$\begin{aligned}
\text{Time}(I) &= \sum_{n_{i_1,i_2,\ldots,i_k} \in V(G_{\mathcal{P}}(I))} \tilde{O}\Big(T'_{i_1,i_2,\ldots,i_k} + T_{\mathcal{P}}\sqrt{\lambda_{i_1,i_2,\ldots,i_k}}\Big) \leq \\
&\leq \sum_{n_{i_1,i_2,\ldots,i_k} \in V(G_{\mathcal{P}}(I))} \tilde{O}\Big(T' + T_{\mathcal{P}}\sqrt{\deg_{out}(n_{i_1,i_2,\ldots,i_k})}\Big) \leq \\
&\leq \tilde{O}\Big(|V_0|(T' + T_{\mathcal{P}}\sqrt{2\delta})\Big) + \sum_{i=1}^{\log\log n} \tilde{O}\Big(|V_i|(T' + T_{\mathcal{P}}\sqrt{2^{2^i}\delta})\Big) = \\
&= \tilde{O}(|V_0|T') + \tilde{O}(|V_0|T_{\mathcal{P}}\sqrt{\delta}) + \sum_{i=1}^{\log\log n} \tilde{O}(|V_i|T') + (T_{\mathcal{P}} \cdot \sqrt{\delta}) \sum_{i=1}^{\log\log n} \tilde{O}(|V_i|2^{2^{i-1}}) \quad (4)
\end{aligned}$$

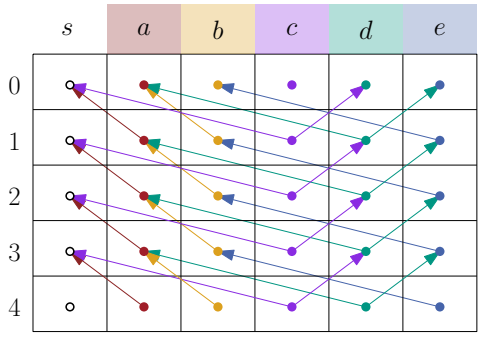
Next, we provide an upper bound for $|V_i|$. Clearly, there exist at most $2n$ nodes $n_{i_1,i_2,\ldots,i_k}$ of $G_{\mathcal{P}}(I)$ such that $\deg_{out}(n_{i_1,i_2,\ldots,i_k}) < 2\delta$ (that is, $|V_0| \leq 2n$). Also, there exist at most n nodes $n_{i_1,i_2,\ldots,i_k}$ of $G_{\mathcal{P}}(I)$ such that $2\delta \leq \deg_{out}(n_{i_1,i_2,\ldots,i_k}) < 4\delta$ (that is, $|V_1| \leq n$). Similarly, there exist at most $\frac{n}{2}$ nodes $n_{i_1,i_2,\ldots,i_k}$ of $G_{\mathcal{P}}(I)$ such that $4\delta \leq \deg_{out}(n_{i_1,i_2,\ldots,i_k}) < 16\delta$ (that is, $|V_2| \leq \frac{n}{2}$). More in general, for $i = 1, \ldots, \log\log n$, there exist at most $\frac{n}{2^{2^{i-1}-1}} = \frac{2n}{2^{2^{i-1}}}$ nodes of $G_{\mathcal{P}}(I)$ such that $2^{2^{i-1}}\delta \leq \deg_{out}(n_{i_1,i_2,\ldots,i_k}) < 2^{2^i}\delta$ (that is, $|V_i| \leq \frac{n}{2^{2^{i-1}-1}} = \frac{2n}{2^{2^{i-1}}}$). In Equation (4), we can then upper bound V_0 with $2n$, and $|V_i|$ with $\frac{2n}{2^{2^{i-1}}}$, if $i > 1$. Therefore, we have:

$$\begin{aligned}
\text{Time}(I) &= \tilde{O}(|V_0|T') + \tilde{O}(|V_0|T_{\mathcal{P}}\sqrt{\delta}) + \sum_{i=1}^{\log\log n} \tilde{O}(|V_i|T') + (T_{\mathcal{P}} \cdot \sqrt{\delta}) \sum_{i=1}^{\log\log n} \tilde{O}(|V_i|2^{2^{i-1}}) \leq \\
&\leq \tilde{O}(n \cdot T') + \tilde{O}(nT_{\mathcal{P}}\sqrt{\delta}) + \tilde{O}(n \cdot T') + (T_{\mathcal{P}} \cdot \sqrt{\delta}) \sum_{i=1}^{\log\log n} \tilde{O}(|V_i|2^{2^{i-1}}) \leq \\
&\leq \tilde{O}(n \cdot T') + \tilde{O}(nT_{\mathcal{P}}\sqrt{\delta}) + (T_{\mathcal{P}} \cdot \sqrt{\delta}) \sum_{i=1}^{\log\log n} \tilde{O}(\frac{2n}{2^{2^{i-1}}}2^{2^{i-1}}) = \\
&= \tilde{O}(n \cdot T') + \tilde{O}(nT_{\mathcal{P}}\sqrt{\delta}) + (T_{\mathcal{P}} \cdot \sqrt{\delta})\tilde{O}(n) \sum_{i=1}^{\log\log n} 2 \\
&= \tilde{O}\Big(n(T' + (T_{\mathcal{P}} \cdot \sqrt{\delta})(1 + 2\log\log n))\Big) = \tilde{O}\Big(n(T' + T_{\mathcal{P}} \cdot \sqrt{\delta})\Big)
\end{aligned} \quad (5)$$

Altogether, Equation (5) shows the desired bound for the running time of $Q_{\mathcal{A}}$ when op $\in \{\text{find}, \min, \max\}$. For the case when op = findAll the same analysis applies with $\sqrt{\delta}$ replaced by $\sqrt{\delta \cdot M}$ according to Theorem 3. This concludes the proof. ◀



(a) An instance $I = (G, w, s = v_s)$ of SSSP.



(b) The dependency graph $G_{\mathcal{P}}(I)$ of I and the dynamic programming table D for instance I. Each node of $G_{\mathcal{P}}(I)$ is placed inside the corresponding entry of D.

Figure 1 Illustrations for problem SSSP.

5 Applications of the Quantum Dynamic Programming Framework

In this section, we consider the Single-Source Shortest Paths (SSSP) problem and the Membership in Context-Free Language (MCFL) problem. Since both the celebrated Bellman-Ford algorithm for SSSP and Cocke-Younger-Kasami algorithm for MCFL are based on dynamic programming recurrences that respect Equation (1), they provide primary examples for computational problems amenable to a quantum speedup via Theorem 4.

In the following for the SSSP and the MCFL problems, and in the full version [16] for the remaining problems listed in Table 1, we demonstrate the applicability of Theorem 4 as follows. First, we present the corresponding classical dynamic programming algorithm. In particular, we describe (i) the **subproblems** whose solution is stored in the dynamic programming table, (ii) the **optimal substructure property** of the problem, and (iii) the **overlap** among subproblems. Then, if needed, we transform the recurrence relation of the dynamic programming algorithm in such a way that it matches the pattern of Equation (1), and we define the dependency set $S_{i_1,i_2,\ldots,i_k}$, the generating set $C_{i_1,i_2,\ldots,i_k}$, the dependency index h, and the function $f_{\mathcal{P}}$. Also, we bound the average degree of the dependency digraph $G_{\mathcal{P}}(I)$ in terms of the size of the instances. Further, we show that the dynamic programming algorithm is simple. Finally, we bound the time complexity of the functions $f_{\mathcal{P}}$, f_C, and γ. Altogether, this allows us to the establish our quantum speedups via Theorem 4.

5.1 Quantum Bellman-Ford for Single-Source Shortest Paths

Let $G = (V, E)$ be a weighted n-vertex m-edge digraph, where each arc $e = uv \in E$, directed from u to v, has an associated weight $w(e)$. Let $p = (u_0, u_1, \ldots, u_k)$ be a directed path from u_0 to u_k. The *weight* of p is $\sum_{(u_iu_{i+1})\in E(p)} w(u_iu_{i+1})$, and the *length* of p is $|E(p)| = k$. A *shortest path* from u to v is a minimum-weight directed path from u to v. The *shortest path distance* from u to v is the weight of a shortest path from u to v.

Next, we define the Single-Source Shortest Paths problem (see Figure 1a for the illustration of an instance of this problem).

Single-Source Shortest Paths (SSSP)

Input: A weighted digraph $G = (V, E)$, a function $w : E \to \mathbb{R}$, and a source vertex $s \in V$.
Output: The shortest-path distances between s and v, for all $v \in V$.

Subproblems. For all $i \in [n]$ and for all $v_j \in V$, a subproblem is defined as finding the smallest weight of a directed path, of length at most i, from the source vertex s to the vertex v_j. Since the maximum length of a shortest path is at most $n - 1$, we have n^2 subproblems.

Optimal substructure. Let D be a dynamic programming table of size $n \times n$, whose entries $D[i][j]$ store the smallest weight of a directed path, of length at most i, from s to v_j, for all $i \in [n]$ and for all $v_j \in V$. If the smallest-weight path p_{s,v_j}, of length at most i, from s to v_j uses uv_j as its last arc, then the subpath of p_{s,v_j} from s to u must be a smallest-weight path, of length at most $i - 1$, from s to u. Consider a smallest-weight path P, of length at most i, from s to v_j, whose weight is stored in $D[i][j]$:

- If P is of length at most $i - 1$, then $D[i][j] = D[i - 1][j]$;
- If P is of length (at most) i and its last edge is $v_k v_j$, then $D[i][j] = D[i - 1][k] + w(v_k v_j)$.

Overlapping subproblems. The value $D[i - 1][k]$ must be accessed to compute all entries $D[i][y]$, where $v_k v_y \in E$.

The above yields the following dynamic programming algorithm for SSSP, due to Bellman and Ford [8, 26].

DP 5.1 (SSSP)

The entries of D can be computes as follows:

Base case: if $i = 0$:

$$D[i][j] = \begin{cases} \infty, \text{ for all } v_j \neq s \\ 0, \text{ if } v_j = s \end{cases}$$

Recursive case: if $i < j$:

$$D[i][j] = \min(D[i - 1][j], \min_{v_k v_j \in E}(D[i - 1][k] + w(v_k v_j))) \tag{6}$$

Note that Equation (6) can be rewritten as:

$$D[i][j] = \min_{v_k v_j \in E}(\min(D[i - 1][j], D[i - 1][k] + w(v_k v_j))),$$

which matches the pattern of Equation (1). In particular, we have that: $S_{i,j} = \{(i - 1, k) : v_k v_j \in E\}$, $C_{i,j} = \{\{(i - 1, k)\} : v_k v_j \in E\}$, $h = 1$, and $f_{\mathcal{P}}(i, j, \{(i - 1, k)\}) = \min(D[i - 1][j], D[i - 1][k] + w(v_k v_j))$. Note that, SSSP is a simple problem, since each entry of $C_{i,j}$ stems from a distinct entry of $S_{i,j}$. Also observe that, the values in D are bounded by $W = (n - 1) \max_{v_i v_j \in E} w(v_i v_j)$. As in [31, 32, 37], we assume that the weights defined by w are in $O(n^c)$, where c is a constant. This implies that each of the entries of D can be represented using $\log W \in O(\log n)$ bits. The time $T_{\mathcal{P}}$, T_{f_C}, and T_γ needed for computing quantumly $f_{\mathcal{P}}$, f_C, and γ is then $O(\log W) = O(\log n)$, $O(\log n)$, and $O(\log n)$, respectively.

Thus, we have that the time $T' = T_{f_C} + T_\gamma$ (see the statement of Theorem 3) is in $O(\log n)$. Finally, we bound the average degree of $G_{\mathcal{P}}(I)$. Recall that an instance I of SSSP is a tuple $I = (G, w, s)$. Note that, $|V(G_{\mathcal{P}}(I))|$ is the number of subproblems, that is, n^2. The n nodes of $V(G_{\mathcal{P}}(I))$ corresponding to the subproblems $D[0][\cdot]$ have outdegree 0. Instead, by the definition of $C_{i,j}$, each node $n_{i,j}$ of $G_{\mathcal{P}}(I)$ corresponding to a subproblem $D[i][j]$ has outdegree equal to $\deg_{out}(v_j)$; refer to Figure 1b. Therefore, for any $i \in \{1, \ldots, n-1\}$, we have that $\sum_{v_j \in V} \deg_{out}(n_{i,j}) = m$. It follows that the average degree δ of $G_{\mathcal{P}}(I)$ is equal to $\frac{2m(n-1)}{n^2} = \frac{2m}{n} - \frac{2m}{n^2} < \frac{2m}{n}$. Therefore, by Theorem 4, there exists a quantum dynamic programming algorithm that solves SSSP for an n-vertex m-edge digraph, using QRAM, in $\tilde{O}(n\sqrt{nm})$ time using $O(n^2)$ space.

Finally, we compare the current-best classical time bound for the SSSP problem with our quantum bound. We remark that the running time of the Bellman-Ford algorithm is $O(n^3)$ time. The current-fastest classical algorithm for SSSP, due to Huang, Jin, and Quanrud [42], runs in $\tilde{O}(mn^{4/5})$ time. Thus, our quantum dynamic programming version of the Bellman-Ford algorithm improves upon [42] when $m \in \Omega(n^{\frac{7}{5}}) = \Omega(n^{1.4})$.

We remark that, as in the classical setting, by explicitly computing the dynamic programming table, our algorithm allows to detect a negative cycle in $O(m)$ time.

5.2 Quantum Cocke-Younger-Kasami for Membership in Context-Free Language

A *formal* grammar G is defined[38] a 4-tuple (V, Σ, R, S), where: **(i)** V is the finite set of non-terminal symbols; **(ii)** T is the finite set of terminal symbols; **(iii)** P is is the set of production rules; and **(iv)** $S \in V$ is the start symbol.

Let G be a *Context-Free Grammar* (CFG) in **Chomsky Normal Form (CNF)**.

A CFG is in CNF if all its production rules are of one of the following forms: **(1)** $A \to BC$ where A, B, C are non-terminal symbols (variables), i.e., a non-terminal symbol A can be replaced by two non-terminal symbols, **(2)** $A \to a$ where A is a non-terminal symbol and a is a terminal symbol, i.e., a non-terminal symbol A can be replaced by a single terminal symbol a.

Let $L(G)$ be the language composed of the strings that can be generated by the grammar G.

Next, we define the Membership in Context-Free Language problem.

Membership in Context-Free Language (MCFL)

Input: A constant-size context-free grammar G in CNF and a nonempty string $w = w_1 w_2 \ldots w_n$, where each w_i is a terminal symbol.

Output: A boolean value `True`, if the string w can be derived from the start non-terminal symbol S of the grammar G, i.e., $w \in L(G)$; `False`, otherwise.

Subproblems. A subproblem is defined as determing whether a consecutive substring of w can be derived from a non-terminal symbol A. Since a string of n symbols has $\binom{n}{2}$ consecutive substrings, we have $\frac{n(n+1)}{2}$ subproblems.

Optimal substructure. Let D be a dynamic programming table of size $n \times n$ (where the entries above the main diagonal are not used), whose entries $D[i][j]$ store the set of non-terminal symbols that can generate the substring $w_j w_{j+1} \ldots w_{j+i-1}$ of w, i.e., the substring of w of length i that starts at the j-th symbol of w. The set of non-terminal symbols in $D[i][j]$ is defines as follows:

- $D[i][j] = \{A : (A \to BC) \in G \wedge \exists k \in [1, i) : (B \in D[k][j]) \wedge (C \in D[i-k][j+k])\}$

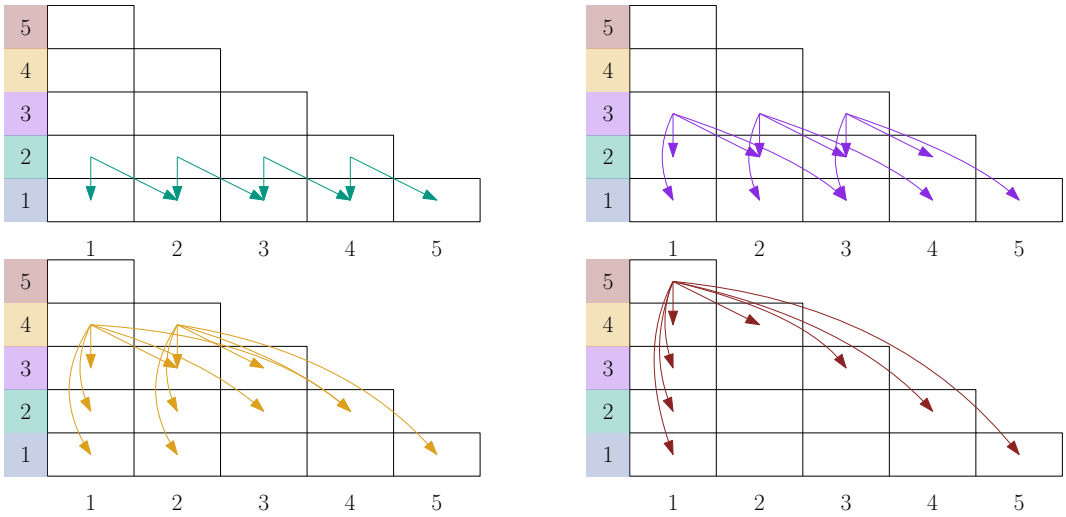


Figure 2 Graph $G_{\mathcal{P}}(I)$ for the dynamic programming algorithm that solves MCFL.

Overlapping subproblems. The value $D[k][j]$ must be accessed to compute all the entries of the form $D[u][j]$, with $u > k$; whereas the value $D[i-k][j+k]$ must be accessed to compute all the entries of the form $D[u][j+k]$, with $u > i-k$.

The above yields the following dynamic programming algorithm for MCFL, due to Cocke-Younger-Kasami [61, 44, 18].

DP 5.2 (MCFL)

The entries of D can be computes as follows:

BASE CASE: if $i = 1$:

$$D[i][j] = \{A \mid (A \to w_j) \in G\}$$

RECURSIVE CASE: if $i < j$:

$$D[i][j] = \text{findAll}_{1 \le k < i}\{A : (A \to BC \in G) \wedge (B \in D[k][j]) \wedge (C \in D[i-k][j+k])\} \tag{7}$$

Note that Equation (7) matches the pattern of Equation (1) of Theorem 4. Observe that, w can be generated from G if and only if $S \in D[n][1]$. In particular, we have that: $S_{i,j} = \{(k,j),(i-k,j+k) : 1 \le k < i\}$, $C_{i,j} = \{\{(k,j),(i-k,j+k)\} : 1 \le k < i\}$, $h = 2$, and $f_{\mathcal{P}}(i,j,\{(k,j),(i-k,j+k)\}) = \{A : (A \to BC \in G) \wedge (B \in D[k][j]) \wedge (C \in D[i-k][j+k])\}$. Note that, MCFL is a simple problem, since each entry of $C_{i,j}$ stems from a distinct entry of $S_{i,j}$. The time needed for computing quantumly $f_{\mathcal{P}}$, f_C, and γ is then $O(1)$, $O(\log n)$, and $O(\log n)$, respectively. Therefore, we have that $T' \in O(\log n)$.

Finally, we bound the average degree of $G_{\mathcal{P}}(I)$. Recall that an instance I of MCFL is a pair (G,w). Note that, $|V(G_{\mathcal{P}}(I))|$ is the number of subproblems defined by I, that is, $\frac{n(n+1)}{2}$. Note that, the n nodes of $V(G_{\mathcal{P}}(I))$ corresponding to the subproblems $D[1][j]$ have outdegree 0. Instead, by the definition of $C_{i,j}$, each node $n_{i,j}$ of $G_{\mathcal{P}}(I)$ corresponding to

a subproblem $D[i][j]$ has outdegree equal to $2(i-1)$; refer to Figure 2. Therefore, for any $i \in \{1,\ldots,n\}$, we have that $\sum_{j=1}^{n+1-i} \deg_{out}(n_{i,j}) = 2(n+1-i)(i-1)$. The total number of edges of $G_{\mathcal{P}}(I)$ is equal to

$$\sum_{i=1}^{n} 2(n+1-i)(i-1) \leq 2n \sum_{i=1}^{n} i = 2n\Big(\frac{n(n+1)}{2}\Big) = n^2(n+1)$$

It follows that the average degree δ of $G_{\mathcal{P}}(I)$ is upperbounded by $(n^2(n+1))/(\frac{n(n+1)}{2}) = 2n$. Therefore, by Theorem 4, there exists a quantum dynamic programming algorithm that solves MCFL for a constant-size context-free grammar G and a string w on length n, using QRAM, in $\tilde{O}(n^2\sqrt{n})$ time using $O(n^2)$ space.

Finally, we compare the current-best classical time bound for the MCFL problem with our quantum bound. We remark that the running time of the CYK algorithm is $O(n^3)$. Valiant demonstrated in 1975 that membership in context-free language recognition can be performed at least as efficiently as Boolean matrix multiplication [57]. Since Boolean matrix multiplication can be done in $O(n^{2.81})$ time using Strassen's algorithm [55], this implies an indirect $O(n^{2.81})$-time algorithm for testing membership in a context-free language. However, such an algorithm does not allow to directly retrieve the sequence of productions used to derive w. The current-fastest classical algorithm for matrix multiplication has a time complexity of $O(n^{2.371552})$ [58], building upon the Coppersmith-Winograd algorithm [19]. Observe that, the constant factors hidden by the Big O notation are alone so substantial that these fast matrix multiplication algorithms are impractical for matrix sizes manageable by current computers. Finally, we remark that our approach is methodologically simpler than relying upon matrix multiplication, as it directly accelerates an "easy" algorithm for context-free parsing. In fact, using matrix multiplication for parsing requires a more complex, three-step process: **(1)** transforming the parsing instance into a format suitable for matrix multiplication, **(2)** applying the sophisticated (and practically cumbersome) matrix multiplication algorithm described in [58], and **(3)** converting the output of the recognition process (provided by the matrix multiplication) into a parsing solution[6].

6 Conclusions and Open Problems

In this paper, we presented a framework that systematically extends classical dynamic programming algorithms that exhibit specific characteristics in the computation of their dynamic programming table, into accelerated quantum counterparts. These characteristics pertain to the structure of the *dependency set*, specifying the global set of entries involved in computing a table entry, and to the structure of the *generating set*, specifying the groups of entries simultaneously involved in providing a candidate value for a table entry.

By leveraging quantum search primitives, such as *find*, *findAll*, *min*, and *max*, and the QRAM, we transform dynamic programming recurrence relations into efficient quantum subroutines. Our approach lowers the computational cost of constructing each entry in the dynamic programming table, often achieving a significant speedup over the best-known classical algorithms. We demonstrate the versatility of this framework by applying it to several well-known problems, including Single-Source Shortest Paths.

This work establishes a foundation for developing more efficient quantum algorithms for a class of dynamic programming problems, paving the way for several research directions. A natural extension is to investigate the applicability of our framework to dynamic programming

[6] Note that Ruzzo showed parsing is slightly harder than recognition by a logarithmic factor [52].

algorithms that do not have the above characteristics. It is also worth investigating whether similar quantum speedups can be achieved for algorithms that iteratively refine estimates of the optimal solution, such as the Floyd-Warshall algorithm. Finally, a key challenge is optimizing the space complexity of quantum dynamic programming algorithms. Due to the limitations of quantum memory, it is crucial to develop strategies that avoid storing the entire dynamic programming table. This may involve computing entries on-the-fly, leveraging quantum speedups for efficient on-demand computation, and minimizing the number of entries stored in superposition.

References

1 Scott Aaronson. Introduction to quantum information science lecture notes, April 2019. URL: https://www.scottaaronson.com/qclec.pdf.

2 Scott Aaronson, Nai-Hui Chia, Han-Hsuan Lin, Chunhao Wang, and Ruizhe Zhang. On the quantum complexity of closest pair and related problems. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020*, volume 169 of *LIPIcs*, pages 16:1–16:43. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.CCC.2020.16.

3 Andris Ambainis. Quantum search algorithms. *SIGACT News*, 35(2):22–35, June 2004. doi:10.1145/992287.992296.

4 Andris Ambainis, Kaspars Balodis, Janis Iraids, Martins Kokainis, Krisjanis Prusis, and Jevgenijs Vihrovs. Quantum speedups for exponential-time dynamic programming algorithms. In Timothy M. Chan, editor, *SODA 2019*, pages 1783–1793. SIAM, 2019. doi:10.1137/1.9781611975482.107.

5 Andris Ambainis and Nikita Larka. Quantum algorithms for computational geometry problems. In Steven T. Flammia, editor, *15th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2020*, volume 158 of *LIPIcs*, pages 9:1–9:10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.TQC.2020.9.

6 Andris Ambainis and Ilja Repko. Quantum algorithm for the domatic number problem. *Balt. J. Mod. Comput.*, 12(3), 2024. doi:10.22364/BJMC.2024.12.3.03.

7 Vladimirs Andrejevs, Aleksandrs Belovs, and Jevgenijs Vihrovs. Quantum algorithms for hopcroft's problem. In Rastislav Královic and Antonín Kucera, editors, *49th International Symposium on Mathematical Foundations of Computer Science, MFCS 2024*, volume 306 of *LIPIcs*, pages 9:1–9:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.MFCS.2024.9.

8 Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958. URL: http://www.jstor.org/stable/43634538.

9 Richard Bellman. On the approximation of curves by line segments using dynamic programming. *Commun. ACM*, 4(6):284, 1961. doi:10.1145/366573.366611.

10 Marc Benkert, Herman J. Haverkort, Moritz Kroll, and Martin Nöllenburg. Algorithms for multi-criteria one-sided boundary labeling. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Graph Drawing, 15th International Symposium, GD 2007*, volume 4875 of *Lecture Notes in Computer Science*, pages 243–254. Springer, 2007. doi:10.1007/978-3-540-77537-9_25.

11 Marc Benkert, Herman J. Haverkort, Moritz Kroll, and Martin Nöllenburg. Algorithms for multi-criteria boundary labeling. *J. Graph Algorithms Appl.*, 13(3):289–317, 2009. doi:10.7155/JGAA.00189.

12 Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum counting. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *ICALP'98*, volume 1443 of *LNCS*, pages 820–831. Springer, 1998. doi:10.1007/BFB0055105.

13 Susanna Caroppo, Giordano Da Lozzo, and Giuseppe Di Battista. Quantum algorithms for one-sided crossing minimization. In Stefan Felsner and Karsten Klein, editors, *32nd International Symposium on Graph Drawing and Network Visualization, GD 2024*, volume 320 of *LIPIcs*, pages 20:1–20:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.GD.2024.20.

14 Susanna Caroppo, Giordano Da Lozzo, and Giuseppe Di Battista. Quantum algorithms for one-sided crossing minimization. *Theoretical Computer Science*, 1052:115424, 2025. doi:10.1016/j.tcs.2025.115424.

15 Susanna Caroppo, Giordano Da Lozzo, and Giuseppe Di Battista. Quantum graph drawing. *J. Graph Algorithms Appl.*, 29(2):3–47, 2025. doi:10.7155/JGAA.V29I2.3039.

16 Susanna Caroppo, Giordano Da Lozzo, Giuseppe Di Battista, Michael T. Goodrich, and Martin Nöllenburg. Quantum speedups for polynomial-time dynamic programming algorithms, 2025. arXiv:2507.00823.

17 Susanna Caroppo, Giordano Da Lozzo, and Giuseppe Di Battista. Quantum graph drawing. In Ryuhei Uehara, Katsuhisa Yamanaka, and Hsu-Chun Yen, editors, *Proc. 18th International Conference and Workshops on Algorithms and Computation, WALCOM 2024*, volume 14549 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2024. doi:10.1007/978-981-97-0566-5_4.

18 John Cocke. *Programming languages and their compilers: Preliminary notes*. New York University, 1969.

19 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.

20 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.

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

22 Christopher M. Dawson and Michael A. Nielsen. The solovay-kitaev algorithm. *Quantum Inf. Comput.*, 6(1):81–95, 2006. doi:10.26421/QIC6.1-6.

23 Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum. *CoRR*, quant-ph/9607014, 1996. arXiv:quant-ph/9607014.

24 Jeff Erickson. *Algorithms*. Independently published, 2019. URL: http://jeffe.cs.illinois.edu/teaching/algorithms/.

25 Geeks for Geeks. Largest divisible subset in array. https://www.geeksforgeeks.org/largest-divisible-subset-array/, 2024. Accessed: (21/02/2025).

26 L. R. Ford. *Network Flow Theory*, pages 253–313. Springer New York, New York, NY, 1997. doi:10.1007/0-387-22633-8_9.

27 Shion Fukuzawa, Michael T. Goodrich, and Sandy Irani. Quantum Tutte embeddings. *CoRR*, abs/2307.08851, 2023. doi:10.48550/arXiv.2307.08851.

28 Shion Fukuzawa, Michael T. Goodrich, and Sandy Irani. Quantum Tutte embeddings. In Michael A. Bekos and Markus Chimani, editors, *31st International Symposium on Graph Drawing and Network Visualization, GD*, volume 14466 of *LNCS*, pages 241–243. Springer, 2023.

29 Shion Fukuzawa, Michael T. Goodrich, and Sandy Irani. Quantum combine and conquer and its applications to sublinear quantum convex hull and maxima set construction. In Oswin Aichholzer and Haitao Wang, editors, *41st International Symposium on Computational Geometry, SoCG 2025*, volume 332 of *LIPIcs*, pages 51:1–51:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.SOCG.2025.51.

30 Bartholomew Furrow. A panoply of quantum algorithms. *Quantum Inf. Comput.*, 8(8):834–859, 2008. doi:10.26421/QIC8.8-9-11.

31 Harold N. Gabow. Scaling algorithms for network problems. *J. Comput. Syst. Sci.*, 31(2):148–168, 1985. doi:10.1016/0022-0000(85)90039-X.

32 Harold N. Gabow and Robert Endre Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989. doi:10.1137/0218069.

33 Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. Trajectory-based dynamic map labeling. In Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam, editors, *ISAAC 2013*, volume 8283 of *LNCS*, pages 413–423. Springer, 2013. doi:10.1007/978-3-642-45030-3_39.

34 I. M. Georgescu, S. Ashhab, and Franco Nori. Quantum simulation. *Rev. Mod. Phys.*, 86:153–185, March 2014. doi:10.1103/RevModPhys.86.153.

35 Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Phys. Rev. Lett.*, 100:160501, April 2008. doi:10.1103/PhysRevLett.100.160501.

36 Adam Glos, Martins Kokainis, Ryuhei Mori, and Jevgenijs Vihrovs. Quantum speedups for dynamic programming on n-dimensional lattice graphs. In Filippo Bonchi and Simon J. Puglisi, editors, *MFCS 2021*, volume 202 of *LIPIcs*, pages 50:1–50:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.MFCS.2021.50.

37 Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM J. Comput.*, 24(3):494–504, 1995. doi:10.1137/S0097539792231179.

38 Raymond Greenlaw and James H. Hoover. *Fundamentals of the Theory of Computation: Principles and Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

39 Lov K. Grover. A fast quantum mechanical algorithm for database search. In Gary L. Miller, editor, *STOC 1996*, pages 212–219. ACM, 1996. doi:10.1145/237814.237866.

40 Yijie Han and Tadao Takaoka. An $O(n^3 \log\log n/\log^2 n)$ time algorithm for all pairs shortest paths. *J. Discrete Algorithms*, 38-41:9–19, 2016. doi:10.1016/J.JDA.2016.09.001.

41 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007.

42 Yufan Huang, Peter Jin, and Kent Quanrud. Faster single-source shortest paths with negative real weights via proper hop distance. In Yossi Azar and Debmalya Panigrahi, editors, *SODA 2025*, pages 5239–5244. SIAM, 2025. doi:10.1137/1.9781611978322.178.

43 Daniel Jurafsky and James H. Martin. *Speech and language processing - an introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall series in artificial intelligence. Prentice Hall, 2000.

44 Tadao Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. *Coordinated Science Laboratory Report no. R-257*, 1966.

45 Kamil Khadiev and Liliya Safina. Quantum algorithm for dynamic programming approach for dags. applications for zhegalkin polynomial evaluation and some problems on dags. In Ian McQuillan and Shinnosuke Seki, editors, *UCNC 2019*, volume 11493 of *LNCS*, pages 150–163. Springer, 2019. doi:10.1007/978-3-030-19311-9_13.

46 Alexei Y. Kitaev, A. H. Shen, and Mikhail N. Vyalyi. *Classical and Quantum Computation*, volume 47 of *Graduate studies in mathematics*. American Mathematical Society, 2002. URL: https://bookstore.ams.org/gsm-47/.

47 Jon M. Kleinberg and Éva Tardos. *Algorithm design*. Addison-Wesley, 2006.

48 Vladislavs Klevickis, Krisjanis Prusis, and Jevgenijs Vihrovs. Quantum speedups for treewidth. In François Le Gall and Tomoyuki Morimae, editors, *TQC 2022*, volume 232 of *LIPIcs*, pages 11:1–11:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.TQC.2022.11.

49 G.T. Klincsek. Minimal triangulations of polygonal domains. In Peter L. Hammer, editor, *Combinatorics 79*, volume 9 of *Annals of Discrete Mathematics*, pages 121–123. Elsevier, 1980. doi:10.1016/S0167-5060(08)70044-X.

50 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information (10th Anniversary edition)*. Cambridge University Press, 2016. URL: https://www.cambridge.org/de/academic/subjects/physics/quantum-physics-quantum-information-and-quantum-computation/quantum-computation-and-quantum-information-10th-anniversary-edition.

51 Koustubh Phalak, Avimita Chatterjee, and Swaroop Ghosh. Quantum random access memory for dummies. *Sensors*, 23(17):7462, 2023. doi:10.3390/S23177462.

52 Walter L. Ruzzo. On the complexity of general context-free language parsing and recognition (extended abstract). In Hermann A. Maurer, editor, *Automata, Languages and Programming, 6th Colloquium*, volume 71 of *LNCS*, pages 489–497. Springer, 1979. doi:10.1007/3-540-09510-1_39.

53 Kazuya Shimizu and Ryuhei Mori. Exponential-time quantum algorithms for graph coloring problems. *Algorithmica*, 84(12):3603–3621, 2022. doi:10.1007/S00453-022-00976-2.

54 Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.*, 41(2):303–332, 1999. doi:10.1137/S0036144598347011.

55 Volker Strassen. Gaussian elimination is not optimal. *Matematika*, 14(3):127–128, 1970.

56 Vincent T'kindt, Federico Della Croce, and Mathieu Liedloff. Moderate exponential-time algorithms for scheduling problems. *Ann. Oper. Res.*, 343(2):753–783, 2024. doi:10.1007/S10479-024-06289-7.

57 Leslie G. Valiant. General context-free recognition in less than cubic time. *J. Comput. Syst. Sci.*, 10(2):308–315, 1975. doi:10.1016/S0022-0000(75)80046-8.

58 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In David P. Woodruff, editor, *SODA 2024*, pages 3792–3835. SIAM, 2024. doi:10.1137/1.9781611977912.134.

59 Gerhard J. Woeginger. Open problems around exact algorithms. *Discret. Appl. Math.*, 156(3):397–405, 2008. doi:10.1016/J.DAM.2007.03.023.

60 Shifan Xu, Connor T. Hann, Ben Foxman, Steven M. Girvin, and Yongshan Ding. Systems architecture for quantum random access memory. In *MICRO 2023*, pages 526–538. ACM, 2023. doi:10.1145/3613424.3614270.

61 Daniel H. Younger. Recognition and parsing of context-free languages in time n^3. *Inf. Control.*, 10(2):189–208, 1967. doi:10.1016/S0019-9958(67)80007-X.

62 Mohammed Zidan, Abdel-Haleem Abdel-Aty, Ashraf Khalil, Mahmoud A. Abdel-Aty, and Hichem Eleuch. A novel efficient quantum random access memory. *IEEE Access*, 9:151775–151780, 2021. doi:10.1109/ACCESS.2021.3119588.

Enumerating Minimal Dominating Sets and Variants in Chordal Bipartite Graphs

Emanuel Castelo
Aix-Marseille Université, CNRS, LIS, Marseille, France

Oscar Defrain
Aix-Marseille Université, CNRS, LIS, Marseille, France

Guilherme C. M. Gomes
LIRMM, Université de Montpellier, CNRS, Montpellier, France
Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Abstract

Enumerating minimal dominating sets with polynomial delay in bipartite graphs is a long-standing open problem. To date, even the subcase of chordal bipartite graphs is open, with the best known algorithm due to Golovach, Heggernes, Kanté, Kratsch, Sæther, and Villanger running in incremental-polynomial time. We improve on this result by providing a polynomial delay and space algorithm enumerating minimal dominating sets in chordal bipartite graphs. Additionally, we show that the total and connected variants admit polynomial and incremental-polynomial delay algorithms, respectively, within the same class. This provides an alternative proof of a result by Golovach et al. for total dominating sets, and answers an open question for the connected variant. Finally, we give evidence that the techniques used in this paper cannot be generalized to bipartite graphs for (total) minimal dominating sets, unless $\mathsf{P} = \mathsf{NP}$, and show that enumerating minimal connected dominating sets in bipartite graphs is harder than enumerating minimal transversals in general hypergraphs.

2012 ACM Subject Classification Mathematics of computing → Enumeration; Mathematics of computing → Graph enumeration

Keywords and phrases algorithmic enumeration, minimal dominating sets, connected dominating sets, total dominating sets, chordal bipartite graphs, sequential method, polynomial delay

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.15

Related Version *Full Version*: `https://arxiv.org/abs/2502.14611`

Funding *Oscar Defrain*: Supported by the ANR project PARADUAL (ANR-24-CE48-0610-01). *Guilherme C. M. Gomes*: Funded by the European Union, project PACKENUM, grant number 101109317. Views and opinions expressed are however those of the author only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

1 Introduction

Hypergraphs are encountered in countless areas of computer science or discrete mathematics. Formally, they consist of a pair $\mathcal{H} = (V(\mathcal{H}), E(\mathcal{H}))$ where $V(\mathcal{H})$ is a set of elements called *vertices*, and $E(\mathcal{H})$ is a family of subsets of $V(\mathcal{H})$ called hyperedges (or simply edges). Hypergraphs are known as *graphs* when their edges are of size precisely two. A *transversal* in a hypergraph $\mathcal{H}$ is a set $T \subseteq V(\mathcal{H})$ which intersects every edge of $\mathcal{H}$. It is called *minimal* if it is inclusion-wise minimal, i.e., if *none* of its proper subsets is a transversal. The problem of enumerating the minimal transversals of a hypergraph, usually denoted by Trans-Enum, is one of the most important problems in algorithmic enumeration, for it has far-reaching

© Emanuel Castelo, Oscar Defrain, and Guilherme C. M. Gomes;
licensed under Creative Commons License CC-BY 4.0

implications in various areas of computer science such as logic, database theory, data profiling, or artificial intelligence [5, 16, 30, 37]; we refer the reader to [16] and the surveys [18, 22] for a comprehensive overview on the fundamental and practical aspects of this problem.

In a typical enumeration problem such as Trans-Enum, a first observation to be made is that the number of solutions can be exponential in both n, the number of vertices, and m, the number of edges. Thus, asking for running times which are polynomial in $n + m$ is meaningless. Rather, we are interested in algorithms performing in output-polynomial time, i.e., that run in a time which is polynomial in both the input size, and the output size. More strict notions of tractability can be required, such as incremental-polynomial times, and polynomial delay. We say that an algorithm runs in *incremental-polynomial* time if it produces the i^{th} solution in a time which is polynomial in the input size plus i. An algorithm is said to run with *polynomial delay* if the times before the first output, after the last output, and in between two consecutive outputs are bounded by a polynomial in the input size only. To date, the best-known algorithm solving Trans-Enum is due to Fredman and Khachiyan [21] and runs in quasi-polynomial incremental time. Since then, output-polynomial time algorithms were obtained for several important classes, including those generalizing that of bounded degree [17], or bounded edge size [38].

Dominating sets are among the most studied objects in graphs. A *dominating set* in a graph G is a set $D \subseteq V(G)$ such that every vertex is either in D, or adjacent to a vertex of D. Again, it is called *minimal* if it is inclusion-wise minimal. The problem of enumerating minimal dominating sets in graphs is denoted by Dom-Enum. Note that being a dominating set of G can be rephrased as being a transversal of each closed neighborhood of G, so, Dom-Enum is a special case of Trans-Enum with a linear number of hyperedges.

The problem Dom-Enum has gained lots of interest since it has been shown, perhaps surprisingly, to be polynomially equivalent to Trans-Enum, even when restricted to cobipartite graphs [33]. In particular, this work has initiated a fruitful line of research of characterizing which graph classes admit output-polynomial algorithms, in the hope of better understanding the underlying structure that makes Trans-Enum tractable. For example, polynomial-delay algorithms were obtained for various graph classes including chordal graphs [34] (with linear delay on some subclasses [13, 33]), graphs of bounded degeneracy [17], line graphs [35], graphs of bounded clique-width [11], or graph classes related to permutation graphs and their generalizations [8,23,32]. Incremental-polynomial time algorithms were obtained for graphs of bounded conformality [38], $C_{\leq 6}$-free graphs [27], chordal bipartite graphs [24], (C_4, C_5, claw)-free graphs [32], or graphs arising in geometrical contexts [20, 23] including unit-disk and unit-square graphs. Output-polynomial time algorithms are known for $\log n$-degenerate graphs [17], and graphs of bounded clique number [7].

In [33], the authors also include two natural variants of domination in their study: total and connected dominating sets. A *total dominating set* is a set D such that every vertex of G has a neighbor in D. In other words, elements of D cannot "dominate" themselves and need a neighbor in the set. A *connected dominating set* is a dominating set that induces a connected subgraph. Note that any connected dominating set is a total dominating set, as long as it has more than one element, making the intersection of these two notions irrelevant to study. Perhaps not surprisingly, the authors in [33] show that the problems of enumerating minimal total and connected dominating sets, denoted TDom-Enum and CDom-Enum, respectively, are equivalent to Trans-Enum when restricted to split graphs. These equivalences on a relatively restricted graph class suggest that these variants of Dom-Enum are challenging, and may explain why they have received less attention in the literature.

Regarding total dominating sets, a first observation is that they are transversals of open neighborhoods. This implies an incremental quasi-polynomial time algorithm for the problem using the results of Fredman and Khachiyan [21]. However, few output-polynomial time algorithms are known for specific instances of TDom-Enum. Among them, we cite the class of chordal bipartite graphs, which admits a polynomial-delay algorithm for the problem [24].

Concerning connected dominating sets, it is not known whether the problem reduces to Trans-Enum. It has in fact been conjectured that this is not the case in [6]. Rather, it has been shown in [33] that the problem reduces to the enumeration of minimal transversals of an implicit hypergraph given by the minimal separators of the graph. However, the number of such separators may be exponential, and are intractable to compute [9]. The only work known to us dealing with connected dominating sets enumeration, apart from [33], falls under the input-sensitive approach [1, 25, 26] where one aims at small exponential times. Still, we note that some essential questions concerning output-sensitive enumeration were addressed in [1] such as the NP-hardness of the associated extension problem, i.e., deciding if a given set $U \subseteq V(G)$ is contained in some minimal connected dominating set.

Note that, in the literature, particular attention has been put into characterizing the status of these problems on bipartite, split, and cobipartite graphs. This is because such instances contain the incidence graph of any hypergraph, which is fundamentally challenging and makes interesting links with Trans-Enum. However, and in contrast to the split and cobipartite cases, the case of bipartite graphs has been particularly challenging with the question of the (in)tractability of these problems raised in various works [24,36]. Only recently has an output-polynomial time algorithm been provided for Dom-Enum in a generalization of bipartite graphs [7]. Still the problem is not known to admit a polynomial-delay algorithm. Concerning TDom-Enum and CDom-Enum, it is also open, to the best of our knowledge, whether they can be solved with polynomial delay, or even in output-polynomial time, in bipartite graphs. Among natural subclasses of bipartite graphs, the case of chordal bipartite graphs has been studied as it admits nice structural properties analogous to chordal graphs, such as perfect elimination orderings and linearly many minimal separators. We note, however, that even in this particular class, the existence of a polynomial-delay algorithm for Dom-Enum seems open, and has been left open in [24].

The following two natural questions arise from the literature.

▶ **Question 1.** *Can the minimal dominating sets of a chordal bipartite graph be enumerated with polynomial delay?*

▶ **Question 2.** *Can the minimal connected dominating sets of a chordal bipartite graph be enumerated in output-polynomial time?*

Only the case of minimal total dominating sets has been settled for chordal bipartite graphs, with the following result.

▶ **Theorem 1** ([24])**.** *The minimal total dominating sets of a chordal bipartite graph can be enumerated with polynomial delay.*

Our results and techniques. This paper is devoted to providing a positive answer to Questions 1 and 2, almost completing the picture for chordal bipartite graphs. Collaterally, we provide an alternative proof of Theorem 1 involving small modifications of our algorithm for minimal dominating sets; we note that the obtained algorithm has similar time bounds as the one in [24].

Our algorithms for Dom-Enum (Theorem 18) and TDom-Enum (Theorem 1) are based on the *ordered generation* framework (also known as *sequential method* in various works) which has been introduced in [17] as a generalization of the algorithm in [45] (see also [31] for a concise description). This approach has since then proved to be fruitful in various contexts related to domination and minimal transversals [7, 10, 14]. Roughly, the idea is to decompose the instance with respect to an ordering of its elements, and, given the partial solutions of the instance induced by the i first elements, to extend them to the next element of the instance. To achieve polynomial delay, a parent-child relation on partial solutions is defined. Then, the resulting tree is traversed in a DFS manner, and only the leaves (which correspond to actual solutions) are output. Overall, the technique allows to reduce the general enumeration to the enumeration of the children of a partial solution, and is particularly suitable for instances admitting good elimination orderings. In our case, we use the weak-elimination ordering of chordal bipartite graphs (see Section 2) to solve children generation in polynomial time, yielding a polynomial-delay algorithm for Dom-Enum and TDom-Enum in that class.

Concerning CDom-Enum, we show that the minimal separators of a chordal bipartite graph define a hypergraph of bounded conformality (Lemma 20), a parameter introduced by Berge [3] and successfully used in the design of tractable algorithms for domination [32] and minimal transversals enumeration [38]. This, combined with the fact that chordal bipartite graphs have polynomially many minimal separators, which can be listed in polynomial time (see Section 2), allows us to obtain an incremental-polynomial time algorithm for CDom-Enum in that class (Theorem 21).

Additionally, we give evidence that the sequential method cannot be directly used to generalize our results to bipartite graphs. More specifically, we show that the problem of generating the children of a given partial solution is intractable for Dom-Enum (Lemma 22) and TDom-Enum. For CDom-Enum we make a stronger statement (Theorem 24): the problem is at least as hard as Trans-Enum, which was not known to the best of our knowledge, despite the reduction being straightforward.

Organization of the paper. We introduce preliminary notions and properties dealing with chordal bipartite graphs and minimal separators in Section 2. We describe the sequential approach in Section 3, and provide our algorithms for Dom-Enum and TDom-Enum in Section 4. The arguments for an incremental-polynomial time algorithm for CDom-Enum in chordal bipartite graphs are given in Section 5. Limitations of the sequential method in bipartite graphs are discussed in Section 6, and the reduction from Trans-Enum to CDom-Enum in bipartite graphs is given in Section 7. We discuss open questions in Section 8.

2 Preliminary notions

We assume familiarity with standard terminology from graph and hypergraph theory, and refer to [3, 15] for notations not included below.

Graphs. All graphs considered in this paper are simple and finite. Given $X \subseteq V(G)$, $G[X] := (X, \{uv \in E(G) \mid u, v \in X\})$ is the subgraph of G *induced* by X. Given $v \in V(G)$, we denote by $N(v)$ its open neighborhood, by $N[v]$ its closed neighborhood, and define its *distance-2 neighborhood* as $N^2(v) := N[N[v]] \setminus N[v]$. The first two notions are naturally extended to subsets by $N[S] := \bigcup_{v \in S} N[S]$ and $N(S) := (\bigcup_{v \in S} N[S]) \setminus S$. An *independent set* is a set of pairwise non-adjacent vertices. A graph is *bipartite* if its vertex set $V(G)$ can be partitioned into two independent sets X, Y. It is called *bipartite chain* if, in addition, there

exist orderings $x_1, \ldots, x_p$ and $y_1, \ldots, y_q$ of X and Y, respectively, such that $N(x_i) \subseteq N(x_j)$ for all $1 \leq i \leq j \leq p$, and $N(y_i) \supseteq N(y_j)$ for all $1 \leq i \leq j \leq q$. Note that bipartite chain graphs are closed under taking induced subgraphs.

Hypergraphs. Let $\mathcal{H}$ be a hypergraph. Given a vertex $v \in V(\mathcal{H})$, the set of edges that contain v, also called edges *incident* to v, is denoted by $\mathsf{inc}(v)$. The hypergraph *induced* by a set $X \subseteq V(\mathcal{H})$ of vertices is defined as $\mathcal{H}[X] = (X, \{E \in E(\mathcal{H}) : E \subseteq X\})$. A *transversal* of $\mathcal{H}$ is a subset $T \subseteq V(\mathcal{H})$ such that $E \cap T \neq \emptyset$ for all $E \in E(\mathcal{H})$, i.e., every edge is incident to (or *hit* by) a vertex of T. It is *minimal* if it is minimal by inclusion, or equivalently, if $T \setminus \{v\}$ is not a transversal for any $v \in T$. The set of minimal transversals of $\mathcal{H}$ is denoted by $Tr(\mathcal{H})$. The *private edges* of an element v in a transversal T are those edges of $\mathcal{H}$ which are only hit by v in T; we denote the set of such edges by $\mathsf{priv}(T, v)$. It is well-known (see e.g., [3]) that a set $T \subseteq V(\mathcal{H})$ is a minimal transversal of $\mathcal{H}$ if and only if T is a transversal and every v in T has a private edge. We formally state the problem of enumerating minimal transversals of a hypergraph as follows.

Minimal Transversals Enumeration (Trans-Enum)
Input: A hypergraph $\mathcal{H}$.
Output: The set $Tr(\mathcal{H})$.

Note that in the problem above, $\mathcal{H}$ can be considered inclusion-wise minimal (or *Sperner*) as this does not change its set of minimal transversals, and removing non-minimal hyperedges can be performed in polynomial time [18].

Domination. We recall the different notions of domination given in the introduction. A *dominating set* (or DS for short) in a graph G is a set $D \subseteq V(G)$ such that $V(G) = N[D]$. It is called *connected* if the graph $G[D]$ is connected. A *total dominating set* in G is a set $D \subseteq V(G)$ such that $V(G) = \bigcup_{v \in D} N(v)$. A (resp. connected, total) dominating set is said to be *minimal* if it is inclusion-wise minimal. We denote by $\mathrm{MDS}(G)$ (resp. $\mathrm{MCDS}(G)$, $\mathrm{MTDS}(G)$) the families of minimal such sets in a graph G. The problems of enumerating these families are our main interest in this paper.

We recall that the precise complexity status of each of the problems above is widely open, the first one admitting a quasi-polynomial time algorithm [21, 33], while the third is conjectured to be intractable [6]. Their study has been initiated in [33], where it is noted that each of the problems is a particular instance of Trans-Enum: using the family of (closed and open) neighborhoods for the first two, and a more involved hypergraph for the third one, as formalized in Propositions 2–4. See also [3].

▶ **Proposition 2.** *For every graph G,* $\mathrm{MDS}(G) = Tr(\{N[v] : v \in V(G)\})$.

▶ **Proposition 3.** *For every graph G,* $\mathrm{MTDS}(G) = Tr(\{N(v) : v \in V(G)\})$.

Minimal separators. A *separator* in a graph G is a subset $S \subseteq V(G)$ such that $G - S$ is disconnected. In the following, let $\mathcal{S}(G) := \{S \subseteq V : G - S \text{ is not connected and } \forall S' \subset S, G - S' \text{ is connected}\}$ denote the set of (inclusion-wise) minimal separators of G. Then we have the following.

▶ **Proposition 4** ([33])**.** *For every graph G,* $\mathrm{MCDS}(G) = Tr(\mathcal{S}(G))$.

Minimal separators are not to be confused with the slightly different notion of *minimal a-b separators* which are minimal sets S such that a and b lie in different components of $G-S$. Indeed, while every minimal separator is a minimal a-b separator, the converse is not always true: minimal a-b separators may contain another c-d separator as a subset; see [28] for an early mention of this observation, and [9] for a thorough discussion on this matter. A simple example is the cycle on four vertices with a pendant edge [9]. Moreover, while enumerating a-b minimal separators can be done with polynomial delay [4, 40, 44], enumerating minimal separators is a hard task [9]. However, and as is noted in [33], CDom-Enum reduces to Trans-Enum for graph classes admitting polynomially many a-b separators.[1] Let $\mathcal{G}$ be the hypergraph where each hyperedge is an a-b minimal separator of G; using the above-cited algorithms for enumerating a-b minimal separators, we can construct $\mathcal{G}$ in polynomial time on $|\mathcal{G}| + |V(G)| + |E(G)|$; moreover, note that $\mathcal{S}(G) = \text{Min}_{\subseteq} \mathcal{G}$. Indeed, recall that $Tr(\mathcal{H}) = Tr(\mathcal{G})$ for any pair of hypergraphs $\mathcal{H}, \mathcal{G}$ such that for any $E \in E(\mathcal{G})$ there exists $E' \in E(\mathcal{H})$ with $E' \subseteq E$. In other words, $Tr(\mathcal{H}) = Tr(\text{Min}_{\subseteq} \mathcal{H})$ for any hypergraph $\mathcal{H}$ [3].

Chordal bipartite graphs. A bipartite graph is *chordal bipartite* if it does not contain induced cycles of length greater than four [28]. Note that chordal bipartite graphs are *not* chordal, as they may contain the induced cycle on four vertices. This terminology should be regarded as an analogue to bipartite graphs of chordality, as bicliques are to cliques.

A vertex v in a graph G is called *weak-simplicial* if $N(v)$ is an independent set, and for any two $x, y \in N(v)$, we have $N(x) \subseteq N(y)$ or $N(y) \subseteq N(x)$; two such vertices x, y are said to be *comparable*. Note that this is precisely to say that the graph $G[N(v) \cup N^2(v)]$ induced by the vertices at distance at most 2 from v is a bipartite chain; see [43, Lemma 3]. A *weak-simplicial elimination ordering* of G is an ordering $v_1, \ldots, v_n$ of its vertices such that, for all $1 \leq i \leq n$, v_i is weak-simplicial in $G_i = G[\{v_1, \ldots, v_i\}]$. The following characterization is due to [43] and may be regarded as a simplification of an earlier characterization and recognition algorithm by Uehara [46].

▶ **Proposition 5** ([43, 46]). *A graph is chordal bipartite if and only if it admits a weak-simplicial elimination ordering, which can be computed in polynomial time.*

Another characterization of chordal bipartite graphs based on separators is given in [29], where it is proved that a bipartite graph is chordal bipartite if and only if every minimal *edge* separator induces a complete bipartite subgraph. In [41], the following analogue of the forward implication is given for minimal a-b separators (hence for minimal separators).

▶ **Proposition 6** ([41]). *If G is chordal bipartite and S is an a-b minimal separator then $G[S]$ is complete bipartite.*

In the same paper, the authors show that chordal bipartite graphs have at most $O(n+m)$ minimal a-b separators [41]. Since they can be listed with polynomial delay [40], we derive a polynomial time procedure listing a-b minimal separators and filtering out the non-minimal ones. We formalize this in the next statement that will be of use in Section 5.

▶ **Proposition 7.** *Let G be an n-vertex chordal bipartite graph. Then its set $\mathcal{S}(G)$ of minimal separators can be enumerated in polynomial time in n.*

[1] It should be noted that the statement [33, Corollary 33] presents an inaccuracy where by *minimal separators* it should be understood *minimal a-b separators*. Indeed, listing minimal separators is hard [9] while enumerating minimal a-b separators is tractable [40]. The present inaccuracy comes from an ambiguity and common confusion in the literature around this terminology [41, 44].

Finally, we will need two extra properties of minimal separators in general and in chordal bipartite graphs. Let S be a separator of G and C be a component of $G - S$. We say that C is *close* to S if every vertex in S has at least one neighbor in C. It is folklore that minimal a-b separators S are characterized by the existence of two close components in $G - S$; see [28, 41, 44] or [39] for a proof. We derive one implication for minimal separators since, in particular, they are a-b minimal separators for some specific pairs a, b of vertices. Note however that the converse is not true.

▶ **Proposition 8.** *Let G be a graph and S be a minimal separator of G. Then there exist two components C_1, C_2 in $G - S$ that are close to S.*

Finally, the following will be of use in Section 5. We state it for minimal separators but this more generally holds for minimal a-b separators [41].

▶ **Proposition 9** ([41])**.** *Let G be chordal bipartite with bipartition (A, B). Let S be a minimal separator of G and C be a component of $G - S$ that is close to S. If $S \cap A \neq \emptyset$ then there exists a vertex x in C with $N(x) \cap S = S \cap A$.*

3 The sequential method

We describe the *sequential method*. As by Propositions 2–4, the problems Dom-Enum, TDom-Enum, and CDom-Enum can be seen as particular instances of Trans-Enum we choose to follow the presentation of [2], and refer the reader to [7, 17, 19] for more details on this approach.

Let $\mathcal{H} \subseteq 2^V$ be a hypergraph on vertex set $V = \{v_1, \ldots, v_n\}$. For $1 \leq i \leq n$ we denote by $V_i := \{v_1, \ldots, v_i\}$ the set of its first i vertices, by $\mathcal{H}_i := \mathcal{H}[V_i]$ the hypergraph induced by V_i, and set $V_0 := \mathcal{H}_0 := Tr(\mathcal{H}_0) := \emptyset$. In addition, for $S \subseteq V_i$ and $v \in S$, we note $\mathsf{priv}_i(S, v)$ the set of private edges of v with respect to S in $\mathcal{H}_i$. Given a minimal transversal $T \in Tr(\mathcal{H}_{i+1})$ for $i < n$, we denote by $\mathsf{parent}(T, i+1)$ the *parent of T with respect to $i+1$* obtained after repeatedly removing from T the vertex of smallest index v such that $\mathsf{priv}_i(T, v) = \emptyset$. By definition, every T has a unique parent. Lastly, we denote by $\mathsf{children}(T^\star, i)$, for $T^\star \in Tr(\mathcal{H}_i)$, the family of $T \in Tr(\mathcal{H}_{i+1})$ such that $T^\star = \mathsf{parent}(T, i+1)$.

The next two lemmas are central to the description of the algorithm: they define a tree structure where nodes are pairs (T, i) with $T \in Tr(\mathcal{H}_i)$ and that will be traversed in a DFS manner in order to produce every leaf $T \in Tr(\mathcal{H}) = Tr(\mathcal{H}_n)$ as a solution. We refer to [2] for the proofs of these statements, which follow from the definition of the parent relation.

▶ **Lemma 10.** *Let $0 \leq i \leq n-1$ and $T \in Tr(\mathcal{H}_{i+1})$, then $\mathsf{parent}(T, i+1) \in Tr(\mathcal{H}_i)$.*

▶ **Lemma 11.** *Let $0 \leq i \leq n-1$ and $T^\star \in Tr(\mathcal{H}_i)$, then either:*
- *$T^\star \in Tr(\mathcal{H}_{i+1})$ and consequently $\mathsf{parent}(T^\star, i+1) = T^\star$; or*
- *$T^\star \cup \{v_{i+1}\} \in Tr(\mathcal{H}_{i+1})$ and $\mathsf{parent}(T^\star \cup \{v_{i+1}\}, i+1) = T^\star$.*

Finally, the whole framework culminates in the following statement that reduces Trans-Enum to the enumeration of the children of a given pair (T, i) with $T \in Tr(\mathcal{H}_i)$, $i < n$, while preserving polynomial delay and space.

▶ **Theorem 12.** *Let $f, s : \mathbb{N}^2 \to \mathbb{Z}_+$ be two functions. If there is an algorithm that, given a hypergraph $\mathcal{H}$ on vertex set $V = \{v_1, \ldots, v_n\}$ and m edges, an integer $i \in \{1, \ldots, n-1\}$, and $T^\star \in Tr(\mathcal{H}_i)$, enumerates $\mathsf{children}(T^\star, i)$ with delay $f(n, m)$ and using $s(n, m)$ space, then there is an algorithm that enumerates $Tr(\mathcal{H})$ with $O(nf(n, m))$ delay and $O(ns(n, m))$ space.*

In the following, given $T^\star$ and i as above, let

$$\Delta_{i+1} := \{E \in \mathcal{H}_{i+1} : E \cap T^\star = \emptyset\} \subseteq \mathsf{inc}_{i+1}(v_{i+1})$$

be the family of hyperedges of $\mathcal{H}_{i+1}$ that are not hit by $T^\star$. The following shows that children may be found by considering minimal extensions to $\mathcal{H}_{i+1}$ of minimal transversals of $\mathcal{H}_i$.

▶ **Lemma 13.** *If $T \in \mathsf{children}(T^\star, i)$ then $T = T^\star \cup X$ for some $X \in Tr(\Delta_{i+1})$.*

Proof. Let $T \in \mathsf{children}(T^\star, i)$. Then $T^\star$ is obtained from T by greedily removing elements that have no private edge in $\mathcal{H}_i$. Let X be the set of removed elements. Note that since $T^\star$ is transversal of $\mathcal{H}_i$, X may only have private edges among those $E \in \mathcal{H}_{i+1}$ that are not intersected by $T^\star$, i.e., among Δ_{i+1}. Then, either $\Delta_{i+1} = \emptyset$ in which case $T = T^\star$ by Lemma 11, or X is non empty and intersects each edge in Δ_{i+1}. By minimality of T, we derive that X is minimal with this property, as desired. ◀

As a corollary, in order to enumerate $\mathsf{children}(T^\star, i)$ for some $i \in \{1, \ldots, n-1\}$ and $T^\star \in Tr(\mathcal{H}_i)$, one can simply enumerate all the minimal transversals of Δ_{i+1} and filter out those that do not provide children of $T^\star$. In the following, we call such sets *minimal extensions* of $(T^\star, i)$. Note, however that the number of minimal transversal of Δ_{i+1} may be greater (typically exponential) than the number of actual children, hence that this approach fails to efficiently produce children in general. Moreover, this approach is intractable in general, with instances admitting non-trivial children if and only if NP-complete problems admit a positive answer; see [2] or Section 6. We nevertheless have the following which says that the approach is still tractable if we can bound the number of minimal extension by a polynomial in the input size, and which is in essence what is done in [17].

▶ **Corollary 14.** *If there is an algorithm that, given $T^\star \in Tr(\mathcal{H}_i)$, $i < n$, enumerates all $X \in Tr(\Delta_{i+1})$ in time and space which are polynomial in $n + m$, then there is one that lists $Tr(\mathcal{H})$ with polynomial delay and space.*

Note that, if $\mathcal{H}$ is d-degenerate and $v_1, \ldots, v_n$ is an optimal degeneracy ordering, then minimal extensions have size at most d. In [17], the authors use this fact to provide a polynomial delay algorithm that lists minimal transversals in $O(1)$-degenerate hypergraphs relying on Corollary 14. See [2, 17] for the definition of these notions. In this paper, we combine this corollary together with the elimination ordering of chordal bipartite graphs provided by Proposition 5 to show that Dom-Enum and TDom-Enum can be solved with polynomial delay as well in this class.

4 Minimal extension for (total) dominating sets

In this section, we apply the sequential method to Trans-Enum restricted to hypergraphs of closed neighborhoods of chordal bipartite graphs to show that Question 1 admits a positive answer. More specifically, we use the weak-simplicial elimination ordering of chordal bipartite graphs to restrict the instance of minimal extensions enumeration to a variant of domination in bipartite chain graphs, which is tractable. By Corollary 14, this allows us to obtain Theorem 18.

In the following, let G be a chordal bipartite graph on n vertices and $v_1, \ldots, v_n$ be a weak-simplicial ordering of its vertices as provided by Proposition 5. Let $\mathcal{H}$ be its hypergraph of closed neighborhoods, $i \in \{1, \ldots, n-1\}$ be an integer and T be a minimal transversal of $\mathcal{H}_i$ as in the statement of Corollary 14. We will show how to enumerate the minimal transversals of Δ_{i+1} in polynomial time and space, to derive the desired result.

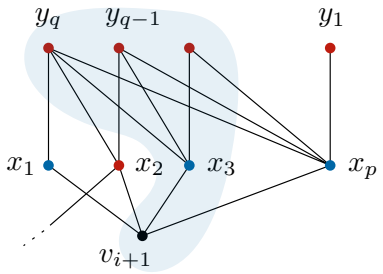


Figure 1 The bipartite chain instance induced by X and Y with v_{i+1} depicted as well. Blue vertices are those whose closed neighborhoods are incident to v_{i+1} *and* included in V_{i+1} as a subset. Red vertices in X have neighbors out of V_i.

First note that since the edges of Δ_{i+1} belong to $\mathcal{H}_{i+1}$ and are incident to v_{i+1}, they may be equal to $N[v_{i+1}]$ if $N[v_{i+1}] \subseteq V_{i+1}$, or to $N[u]$ for some neighbor u of v_{i+1} if $N[u] \subseteq V_{i+1}$. Note that in the later case, the set $N[u]$ may contain vertices at distance 2 from v_{i+1}. From now on, let B be the set of vertices $u \in N(v_{i+1})$ such that $N[u] \in \Delta_{i+1}$, and let

$$R := \left(\bigcup\{E : E \in \Delta_{i+1}\}\right) \setminus B$$

be the remaining vertices that lie in some edge of Δ_{i+1}. Consequently $B \subseteq N(v_{i+1})$ and $R \subseteq N(v_{i+1}) \cup N^2(v_{i+1})$. Thus the graph $H = G_i[R \cup B]$ is an induced subgraph of $G_i[N(v_{i+1}) \cup N^2(v_{i+1})]$ which, by Proposition 5, is a bipartite chain. Hence H is a bipartite chain. We denote by (X, Y) the bipartition of H where $B \subseteq X \subseteq N(v_{i+1})$, and by $x_1, \ldots, x_p$ and $y_1, \ldots, y_q$ the respective orderings of X and Y satisfying $N(x_j) \cap Y \subseteq N(x_\ell) \cap Y$ and $N(y_\ell) \cap X \subseteq N(y_j) \cap X$ for all $j < \ell$; see Section 2 and Figure 1 for an illustration of the situation.

In what remains we shall call *blue* the vertices in B, *red* those in R, and characterize minimal transversals of Δ_{i+1} as particular sets analogous to red-blue dominating sets where red and blue vertices may be selected in order to dominate blue vertices.

▶ **Proposition 15.** *Let $Z \in Tr(\Delta_{i+1})$. Then*

- $|Z \cap R \cap N(v_{i+1})| \leq 1$*; and*
- $|Z \cap N^2(v_{i+1})| \leq 1$.

Proof. First, note that if x belongs to $Z \cap R \cap N(v_{i+1})$ then it is incident to at most one edge in Δ_{i+1} (as H is bipartite), namely $N[v_{i+1}]$, in which case $N[v_{i+1}]$ is a private edge of x. Hence, no two distinct vertices x, y may lie in $Z \cap R \cap N(v_{i+1})$ by minimality of Z, yielding the first inequality.

Now, recall that $N^2(v_{i+1}) \subseteq Y$ hence that one of $N(y) \cap X \subseteq N(y') \cap X$ or $N(y) \cap X \supseteq N(y') \cap X$ holds for any pair $y, y' \in N^2(v_{i+1})$. Since edges of Δ_{i+1} are of the form $N[x]$ for some $x \in X$, no two distinct y, y' can have private edges in Δ_{i+1} simultaneously. Hence, $|Z \cap N^2(v_{i+1})| \leq 1$ as claimed. ◀

▶ **Lemma 16.** *Let $Z \in Tr(\Delta_{i+1})$. Then exactly one of the following holds:*

1. $Z = \{v_{i+1}\}$*;*
2. $Z \subseteq B$ *in which case* $Z = B$*;*
3. $Z \subseteq R$ *in which case* $|Z| \leq 2$*; or*
4. $Z = \{r\} \cup (B \setminus N(r))$ *for some* $r \in N^2(v_{i+1})$.

Proof. First, note that all the cases are mutually distinct. Hence, we may consider some $Z \in Tr(\Delta_{i+1})$, assume that $Z \neq \{v_{i+1}\}$, and prove the conclusion for each of Items 2–4 Note that each $x \in X$ hits a single edge $N[x]$ in Δ_{i+1}, as X is an independent set. Hence, if $Z \subseteq B$ then $Z = B$ concluding Item 2. If $Z \subseteq R$ then we have that $|Z| \leq 2$ by Proposition 15 deriving Item 3. We are thus left with proving Item 4. Suppose that $Z \cap B \neq \emptyset$ and $Z \cap R \neq \emptyset$. Note that if $x \in X \cap R$, then it may be incident to at most one edge in Δ_{i+1}, namely $N[v_{i+1}]$. However, since $Z \cap B \neq \emptyset$ by assumption of this last case, and since each element in B is adjacent to v_{i+1}, the edge $N[v_{i+1}]$ may not be the private edge of such an x in Z. Hence $Z \cap X \cap R = \emptyset$, that is, $Z \cap R \subseteq Y$. By Proposition 15 there is exactly one vertex, call it y, belonging to $Z \cap Y$. Finally, note that in addition to $N[v_{i+1}]$, each $b \in B$ hits a single edge $N[b] \in \Delta_{i+1}$ that only b can hit in X. Hence $B \setminus N(r) \subseteq Z$ and Item 4 follows. ◀

Note that the conditions of the items in Lemma 16 can be checked in polynomial time, and provide an upper bound of $O(n^2)$ on the number of minimal transversals of Δ_{i+1}. Clearly these minimal transversals can be constructed within the same time and using polynomial space. We formalize this as follows.

▶ **Corollary 17.** *The set $Tr(\Delta_{i+1})$ can be listed in polynomial time and space.*

We conclude the following by combining Corollaries 14 and 17. This answers positively Question 1.

▶ **Theorem 18.** *There is a polynomial delay and space algorithm enumerating minimal dominating sets in chordal bipartite graphs.*

Let us now argue that with small modifications we can get a polynomial-delay algorithm for TDom-Enum on the same class. Note that the minimal extension problem is almost identical, with the caveat that the hyperedges we must hit are now of the form $N(x)$ for $x = v_{i+1}$ or $x \in X$, i.e., we are now dealing with open instead of closed neighborhoods. A similar analysis as conducted in the proof of Lemma 16 shows that minimal extensions are red subsets dominating blue elements plus v_{i+1} if $N(v_{i+1}) \subseteq V_{i+1}$. We note that this becomes an instance of proper red-blue domination (where only red elements may be picked) which is precisely what is solved in [24] for chordal bipartite graphs. However in our case, due to the instance being a bipartite chain graph, the problem is trivial. Indeed, it may be seen that solutions have size at most two, intersecting X and Y on at most one vertex. As such, we have an alternative proof of Theorem 1, which was already known by [24].

▶ **Theorem 1** ([24])**.** *The minimal total dominating sets of a chordal bipartite graph can be enumerated with polynomial delay.*

5 Minimal connected dominating sets

Recall that CDom-Enum amounts to list the minimal transversals of the (implicit) hypergraph of minimal separators of a given graph G, i.e., Proposition 4. By Proposition 7, this hypergraph can be constructed in polynomial time if G is chordal bipartite. In this section, we moreover show that this hypergraph has bounded conformality in that case. As a corollary, we can use the algorithm of Khachiyan et al. in [38] to list minimal connected dominating sets in incremental-polynomial time in chordal bipartite graphs, providing a positive answer to Question 2.

Let us define the notion of conformality introduced by Berge in [3]. We say that a hypergraph $\mathcal{H}$ is of *conformality* c if the following property holds for every subset $X \subseteq V(\mathcal{H})$: X is contained (as a subset) in a hyperedge of $\mathcal{H}$ whenever each subset of X of cardinality

at most c is contained (as a subset) in a hyperedge of $\mathcal{H}$. See also [3, 38] for other equivalent characterizations of bounded conformality. We recall that a hypergraph $\mathcal{H}$ is called Sperner if $E_1 \not\subseteq E_2$ for any two distinct hyperedges E_1, E_2 in $\mathcal{H}$.

Khachiyan, Boros, Elbassioni, and Gurvich proved the following.

▶ **Theorem 19** ([38]). *The minimal transversals of a Sperner hypergraph of bounded conformality can be enumerated in incremental-polynomial time.*

Using the properties of minimal separators of chordal bipartite graphs given in Section 2, we obtain the following, whose proof is moved to appendix, and derive the desired algorithm as a corollary of Theorem 19 and Lemma 20.

▶ **Lemma 20.** *If G is chordal bipartite then $\mathcal{S}(G)$ has conformality 5.*

▶ **Theorem 21.** *The minimal connected dominating sets of a chordal bipartite graph can be listed in incremental polynomial time.*

6 Limitations of the sequential method

It is already known that the sequential method may not directly provide output-polynomial time algorithms for Trans-Enum in general; see e.g. [2] for explicit statements. We adapt the construction in [2] to show that this still holds for hypergraphs of (open or closed) neighborhoods of bipartite graphs, i.e., that we may not expect to get tractable algorithms for TDom-Enum and Dom-Enum using this technique.

We begin with Dom-Enum and later detail the modifications to be performed for the statement to hold for TDom-Enum. In the following by $\mathcal{N}(G)$ we mean the hypergraph of closed neighborhoods of G, whose minimal transversals are precisely the minimal dominating sets of G; see Section 2.

Recall that in Multicolored Independent Set problem, we are given a graph G, an integer k, and a partition $\mathcal{V} = (V_1, V_2, \ldots, V_k)$ of $V(G)$ such that V_i is an independent set for every $1 \leq i \leq k$. Sets $V_1, \ldots, V_k$ are usually referred to as color classes. The goal is then to decide whether there exists an independent set in G containing precisely one vertex from each V_i, i.e., that is multicolored. See [12] for more details on this problem.

Due to space constraints, we omit the formal description of the reduction and only include the statement we derive. We refer the reader to Figure 2 for an idea of the construction, and to the appendix for its formal description and proof. The idea is to show that a minimal transversal $T^\star$ of $\mathcal{H}$ induced by all but two of its vertices can be extended into a non-trivial child if and only if an instance G of multicolored independent set admits a solution.

▶ **Lemma 22.** *Let $(G, \mathcal{V})$ be an instance of Multicolored Independent Set. Then there exists a bipartite graph H, an ordering $v_1, \ldots, v_n$ of its vertices, and a minimal transversal $T^\star$ of $\mathcal{N}(H)_{n-2}$, such that the sets in $\mathsf{children}(T^\star, n-2)$ are all multicolored independent sets of G except for one.*

A consequence of Lemma 22 is that the sequential approach may not be directly applied to solve Dom-Enum, even in bipartite graphs. Moreover, in our construction, no vertex v of $T^\star$ is made self-private, i.e., no v has its closed neighborhood as a private edge. Furthermore, the elements in the color classes are dominated by $T^\star$, and their role is thus limited to dominate the c_i's in an extension. Hence, by keeping the same construction, all the arguments of the proof hold for the hypergraph of *open* neighborhoods of G. We derive that the sequential approach may not be directly be used for TDom-Enum neither.

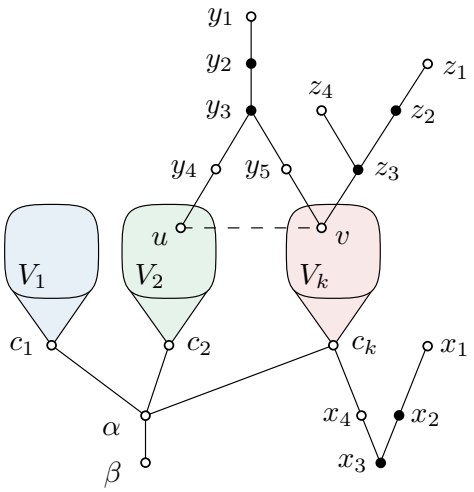


Figure 2 The construction of H, where vertices are ordered arbitrarily except for α and β which are put last, in this order. Color classes $V_1, \ldots, V_k$ induce independent sets in H. The dashed edge denote an edge of G that is not part of H. Black vertices represent $T^\star$. Note that there is one copy of the depicted gadgets for each color class, vertex, and edge of G.

7 Trans-Enum and bipartite graphs[2]

Our final result is a proof that Trans-Enum reduces to CDom-Enum on bipartite graphs, which we are not aware of being previously known or directly implied by other results. In fact, by Proposition 4, if we can construct a graph where (almost) every minimal separator is in a one to one correspondence to our hyperedges, we are done.

Let $\mathcal{H}$ be the input hypergraph to Trans-Enum with $E(\mathcal{H}) = \{E_1, \ldots, E_m\}$ and $V(\mathcal{H}) = \{u_1, \ldots, u_n\}$ and assume w.l.o.g. that $\mathcal{H}$ is a Sperner hypergraph. To obtain G, the input to CDom-Enum, we start with the incidence bipartite graph of $\mathcal{H}$, that is, $\{v_1, \ldots, v_n\}$, $\{e_1, \ldots, e_m\} \subset V(G)$, where v_i corresponds to $u_i \in V(\mathcal{H})$ and e_j corresponds to $E_j \in E(\mathcal{H})$, and $v_i e_j \in E(G)$ if and only if $u_i \in E_j$. To conclude the construction of G, we add two vertices v' and $v^\star$, the edge $v'v^\star$, and the edges $v^\star v_i$ for all $1 \leq i \leq n$. Due to space constraints the proofs of the following are omitted as well.

▶ **Lemma 23.** *The minimal separators of G are $N(e_1), \ldots, N(e_m)$ and $\{v^\star\}$.*

▶ **Theorem 24.** *CDom-Enum on bipartite graphs is at least as hard as Trans-Enum.*

8 Discussion

In this paper, we investigated the complexity of Dom-Enum and its two variants TDom-Enum and CDom-Enum in chordal bipartite graphs. We obtained polynomial-delay algorithms for the first two problems, and an incremental-polynomial time algorithm for the last problem. Then we gave evidence that the techniques used for Dom-Enum and TDom-Enum may not be pushed directly to bipartite graphs. As of CDom-Enum, we proved a stronger statement that CDom-Enum is as hard as Trans-Enum in bipartite graphs, with the later problem being open since more than forty years [16].

[2] After the submission of this work, it has come to our attention that there is a recent stronger result on the subject [42]. We nevertheless leave the proof as is since it is simple and for self-containment.

We leave open the existence of polynomial-delay algorithms for Dom-Enum and TDom-Enum in bipartite graphs, and for CDom-Enum in chordal bipartite graphs. Concerning this last case, we note that obtaining polynomial delay is open for Trans-Enum on hypergraph of bounded dimension, which define a proper subclass of hypergraphs of bounded conformality. Hence, in order to improve Theorem 21, one either has to find another strategy, or should aim at improving the results in [16, 38] first.

References

1 Faisal N. Abu-Khzam, Henning Fernau, Benjamin Gras, Mathieu Liedloff, and Kevin Mann. Enumerating minimal connected dominating sets. In *30th Annual European Symposium on Algorithms (ESA 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.1.

2 Valentin Bartier, Oscar Defrain, and Fionn Mc Inerney. Hypergraph dualization with FPT-delay parameterized by the degeneracy and dimension. In *International Workshop on Combinatorial Algorithms*, pages 111–125. Springer, 2024. doi:10.1007/978-3-031-63021-7_9.

3 Claude Berge. *Hypergraphs: combinatorics of finite sets*, volume 45. Elsevier, 1984.

4 Anne Berry, Jean-Paul Bordat, and Olivier Cogis. Generating all the minimal separators of a graph. *International Journal of Foundations of Computer Science*, 11(03):397–403, 2000. doi:10.1142/S0129054100000211.

5 Thomas Bläsius, Tobias Friedrich, Julius Lischeid, Kitty Meeks, and Martin Schirneck. Efficiently enumerating hitting sets of hypergraphs arising in data profiling. *Journal of Computer and System Sciences*, 124:192–213, 2022. doi:10.1016/J.JCSS.2021.10.002.

6 Hans Bodlaender, Endre Boros, Pinar Heggernes, and Dieter Kratsch. *Open Problems of the Lorentz Workshop, "Enumeration Algorithms using Structure"*. Department of Information and Computing Sciences Utrecht University, Utrecht, The Netherlands, 2015.

7 Marthe Bonamy, Oscar Defrain, Marc Heinrich, Michał Pilipczuk, and Jean-Florent Raymond. Enumerating minimal dominating sets in K_t-free graphs and variants. *ACM Transactions on Algorithms (TALG)*, 16(3):1–23, 2020. doi:10.1145/3386686.

8 Marthe Bonamy, Oscar Defrain, Piotr Micek, and Lhouari Nourine. Enumerating minimal dominating sets in the (in)comparability graphs of bounded dimension posets. *arXiv preprint arXiv:2004.07214*, 2020. arXiv:2004.07214.

9 Caroline Brosse, Oscar Defrain, Kazuhiro Kurita, Vincent Limouzy, Takeaki Uno, and Kunihiro Wasa. On the hardness of inclusion-wise minimal separators enumeration. *Information Processing Letters*, 185:106469, 2024. doi:10.1016/J.IPL.2023.106469.

10 Alessio Conte, Mamadou M. Kanté, Andrea Marino, and Takeaki Uno. Maximal irredundant set enumeration in bounded-degeneracy and bounded-degree hypergraphs. In *International Workshop on Combinatorial Algorithms*, pages 148–159. Springer, 2019.

11 Bruno Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics*, 157(12):2675–2700, 2009. doi:10.1016/J.DAM.2008.08.021.

12 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.

13 Oscar Defrain and Lhouari Nourine. Neighborhood inclusions for minimal dominating sets enumeration: Linear and polynomial delay algorithms in P_7-free and P_8-free chordal graphs. In *30th International Symposium on Algorithms and Computation (ISAAC 2019)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ISAAC.2019.63.

14 Oscar Defrain, Lhouari Nourine, and Simon Vilmin. Translating between the representations of a ranked convex geometry. *Discrete Mathematics*, 344(7):112399, 2021. doi:10.1016/J.DISC.2021.112399.

15 Reinhard Diestel. *Graph Theory*. Springer Berlin, Heidelberg, 2012.

16 Thomas Eiter and Georg Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995. doi:10.1137/S0097539793250299.

17 Thomas Eiter, Georg Gottlob, and Kazuhisa Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM Journal on Computing*, 32(2):514–537, 2003. doi:10.1137/S009753970240639X.

18 Thomas Eiter, Kazuhisa Makino, and Georg Gottlob. Computational aspects of monotone dualization: A brief survey. *Discrete Applied Mathematics*, 156(11):2035–2049, 2008. doi:10.1016/J.DAM.2007.04.017.

19 Khaled Elbassioni, Matthias Hagen, and Imran Rauf. Some fixed-parameter tractable classes of hypergraph duality and related problems. In *Parameterized and Exact Computation: Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings 3*, pages 91–102. Springer, 2008. doi:10.1007/978-3-540-79723-4_10.

20 Khaled Elbassioni, Imran Rauf, and Saurabh Ray. A global parallel algorithm for enumerating minimal transversals of geometric hypergraphs. *Theoretical Computer Science*, 767:26–33, 2019. doi:10.1016/J.TCS.2018.09.027.

21 Michael L. Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, 1996. doi:10.1006/JAGM.1996.0062.

22 Andrew Gainer-Dewar and Paola Vera-Licona. The minimal hitting set generation problem: algorithms and computation. *SIAM Journal on Discrete Mathematics*, 31(1):63–100, 2017. doi:10.1137/15M1055024.

23 Petr A. Golovach, Pinar Heggernes, Mamadou M. Kanté, Dieter Kratsch, Sigve H. Sæther, and Yngve Villanger. Output-polynomial enumeration on graphs of bounded (local) linear MIM-width. *Algorithmica*, 80(2):714–741, 2018. doi:10.1007/S00453-017-0289-1.

24 Petr A. Golovach, Pinar Heggernes, Mamadou M. Kanté, Dieter Kratsch, and Yngve Villanger. Enumerating minimal dominating sets in chordal bipartite graphs. *Discrete Applied Mathematics*, 199:30–36, 2016. doi:10.1016/J.DAM.2014.12.010.

25 Petr A. Golovach, Pinar Heggernes, and Dieter Kratsch. Enumerating minimal connected dominating sets in graphs of bounded chordality. *Theoretical Computer Science*, 630:63–75, 2016. doi:10.1016/J.TCS.2016.03.026.

26 Petr A. Golovach, Pinar Heggernes, Dieter Kratsch, and Reza Saei. Enumeration of minimal connected dominating sets for chordal graphs. *Discrete Applied Mathematics*, 278:3–11, 2020. doi:10.1016/J.DAM.2019.07.015.

27 Petr A. Golovach, Pinar Heggernes, Dieter Kratsch, and Yngve Villanger. An incremental polynomial time algorithm to enumerate all minimal edge dominating sets. *Algorithmica*, 72(3):836–859, 2015. doi:10.1007/S00453-014-9875-7.

28 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.

29 Martin Charles Golumbic and Clinton F. Goss. Perfect elimination and chordal bipartite graphs. *Journal of Graph Theory*, 2(2):155–163, 1978. doi:10.1002/JGT.3190020209.

30 Dimitrios Gunopulos, Heikki Mannila, Roni Khardon, and Hannu Toivonen. Data mining, hypergraph transversals, and machine learning. In *PODS*, pages 209–216. ACM, 1997. doi:10.1145/263661.263684.

31 David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988. doi:10.1016/0020-0190(88)90065-8.

32 Mamadou M. Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the neighbourhood helly of some graph classes and applications to the enumeration of minimal dominating sets. In *International Symposium on Algorithms and Computation*, pages 289–298. Springer, 2012.

33 Mamadou M. Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM Journal on Discrete Mathematics*, 28(4):1916–1929, 2014.

34 Mamadou M. Kanté, Vincent Limouzy, Arnaud Mary, Lhouari Nourine, and Takeaki Uno. A polynomial delay algorithm for enumerating minimal dominating sets in chordal graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 138–153. Springer, 2015.

35 Mamadou M. Kanté, Vincent Limouzy, Arnaud Mary, Lhouari Nourine, and Takeaki Uno. Polynomial delay algorithm for listing minimal edge dominating sets in graphs. In *Workshop on Algorithms and Data Structures*, pages 446–457. Springer, 2015.

36 Mamadou M. Kanté and Lhouari Nourine. Minimal dominating set enumeration. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 1–5. Springer US, Boston, MA, 2014. doi:10.1007/978-3-642-27848-8_721-1.

37 Dimitris Kavvadias, Christos H. Papadimitriou, and Martha Sideri. On Horn envelopes and hypergraph transversals. In *International Symposium on Algorithms and Computation*, pages 399–405. Springer, 1993. doi:10.1007/3-540-57568-5_271.

38 Leonid Khachiyan, Endre Boros, Khaled Elbassioni, and Vladimir Gurvich. On the dualization of hypergraphs with bounded edge-intersections and other related classes of hypergraphs. *Theoretical Computer Science*, 382(2):139–150, 2007. doi:10.1016/J.TCS.2007.03.005.

39 Ton Kloks and Dieter Kratsch. Finding all minimal separators of a graph. In *STACS 94: 11th Annual Symposium on Theoretical Aspects of Computer Science Caen, France, February 24–26, 1994 Proceedings 11*, pages 759–768. Springer, 1994. doi:10.1007/3-540-57785-8_188.

40 Ton Kloks and Dieter Kratsch. Listing all minimal separators of a graph. *SIAM Journal on Computing*, 27(3):605–613, 1998. doi:10.1137/S009753979427087X.

41 Ton Kloks, Ching-Hao Liu, and Sheung-Hung Poon. Feedback vertex set on chordal bipartite graphs. *arXiv preprint*, 2011. arXiv:1104.3915.

42 Yasuaki Kobayashi, Kazuhiro Kurita, Yasuko Matsui, and Hirotaka Ono. Enumerating minimal vertex covers and dominating sets with capacity and/or connectivity constraints. In Adele Anna Rescigno and Ugo Vaccaro, editors, *Combinatorial Algorithms*, pages 232–246, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-63021-7_18.

43 Kazuhiro Kurita, Kunihiro Wasa, Takeaki Uno, and Hiroki Arimura. An efficient algorithm for enumerating chordal bipartite induced subgraphs in sparse graphs. In *International Workshop on Combinatorial Algorithms*, pages 339–351. Springer, 2019. doi:10.1007/978-3-030-25005-8_28.

44 Hong Shen and Weifa Liang. Efficient enumeration of all minimal separators in a graph. *Theoretical computer science*, 180(1-2):169–180, 1997. doi:10.1016/S0304-3975(97)83809-1.

45 Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977. doi:10.1137/0206036.

46 Ryuhei Uehara. Linear time algorithms on chordal bipartite and strongly chordal graphs. In *Automata, Languages and Programming: 29th International Colloquium, ICALP 2002 Málaga, Spain, July 8–13, 2002 Proceedings 29*, pages 993–1004. Springer, 2002. doi:10.1007/3-540-45465-9_85.

An Improved Guillotine Cut for Squares

Parinya Chalermsook
University of Sheffield, England

Axel Kugelmann
PSL Research University, Paris, France

Ly Orgo
Aalto University, Finland

Sumedha Uniyal
Aalto University, Finland

Minoo Zarsav
Aalto University, Finland

Abstract

Given a set of n non-overlapping geometric objects, can we separate a constant fraction of them using straight-line cuts that extend from edge to edge? In 1996, Urrutia posed this question for compact convex objects. Pach and Tardos later refuted it for general line segments by constructing a family where any separable subfamily has size at most $O\left(n^{\log_3 2}\right)$. However, for axis-parallel rectangles, they provided positive evidence, showing that an $\Omega(1/\log n)$-fraction can be separated.

This problem naturally arises in geometric approximation algorithms. In particular, when restricting cuts to only orthogonal straight lines, known as a *guillotine cut sequence*, any bound on the separability ratio directly translates into a clean and simple dynamic programming for computing a maximum independent set of geometric objects.

This paper focuses on the case when the objects are squares. For squares of arbitrary sizes, an $\Omega(1)$-fraction can be separated (Abed et al., APPROX 2015), recently improved to $1/40$ (and $1/160 \approx 0.62\%$ for the weighted case) (Khan and Pittu, APPROX 2020). We further improve this bound, showing that a $9/256 \approx 3.51\%$ can be separated for the weighted case. This result significantly narrows the possible range for squares to $[3.51\%, 50\%]$. The key to our improvement is a refined analysis of the existing framework.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Computational geometry

Keywords and phrases Guillotine cuts, Geometric Approximation Algorithms, Rectangles, Squares

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.16

Funding This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 759557).

Acknowledgements In memory of our friend, Sumedha Uniyal.

1 Introduction

In this paper, we investigate the following combinatorial question, posed initially by Urrutia in 1996 [17]: Given a collection of n disjoint compact convex objects, what is the largest subset that a sequence of straight-line cuts can separate? This combinatorial geometric question naturally arises in two distinct scenarios:

- **Cutting Stock Problems:** The cutting stock problem involves cutting a standard-sized large piece of material – such as a roll of paper, a sheet of glass, or a wooden plank – into smaller specified pieces while minimizing waste [5]. This class of problems has inspired numerous challenges in operations research, combinatorics, and approximation algorithms (see, e.g., [9, 13, 15]).

© Parinya Chalermsook, Axel Kugelmann, Ly Orgo, Sumedha Uniyal, and Minoo Zarsav;
licensed under Creative Commons License CC-BY 4.0

- **Simple Algorithms for Geometric Packing:** In geometric packing, we seek to compute the maximum independent set in the intersection graph of geometric objects, i.e., given a collection of convex polygons, find the largest subset of non-overlapping polygons. These problems have been extensively studied, with a focus on approximation algorithms (due to their NP-hardness even for the simplest objects like disks or squares) [8, 2, 4, 3, 6, 10, 14, 7].

While Urrutia's question was refuted by Pach and Tardos [16] – who constructed a family of line segments where any separable subset has size at most $O(n^{\log_3 2})$ – there remains hope for axis-parallel rectangles:

- For any family of rectangles, there exists a separable set of size $\Omega(n/\log n)$ [16].
- For squares, a constant fraction of any given family is separable, thus affirmatively answering Urrutia's original question for square objects [1].

Both [16] and [1] (and a recent follow-up [12]) rely only on a sequence of orthogonal straight-line cuts (known as *guillotine cuts*) to separate the rectangles. Guillotine cuts are particularly appealing from an approximation algorithms perspective, as they lead to clean and simple combinatorial techniques. Specifically, any non-constructive result that guarantees a guillotine separable γ fraction of objects immediately implies a simple DP-based $(1/\gamma)$-approximation algorithm for geometric packing of the respective object families.

This connection, in fact, holds in a more general setting when the geometric objects have weights, i.e., we are looking for a separable set that retains γ fraction of total weights. For squares, the current best known factor is $1/40$ (and $1/160 \approx 0.62\%$ for the weighted setting) [12], while obtaining a factor better than $1/2$ is not possible [1].

1.1 Our Contributions

Our main contribution is summarized in the following theorem.

▶ **Theorem 1.** *For all families of non-overlapping weighted squares, there exists a separable subset that retains at least* $9/256 \approx 3.51\%$ *of the total weight.*

So our result improves upon the previous result by a factor of 5.6 in the weighted setting.

Broader connections to the DP framework. Our results relate to a dynamic programming framework for geometric packing, initiated by Adamaszek and Wiese [3, 2]. Their framework is parametrized by (α, t), where a geometric object family is called (α, t)-good if it contains a subset of at least an α-fraction of objects that can be separated using polygonal cuts that always create pieces with at most t edges. If a family is (α, t)-good, the framework yields a $(1/\alpha)$-approximation algorithm with running time $n^{O(t)}$. In Adamaszek, Har-Peled, and Wiese's elegant result [2], they show that every family of polygons in 2D is $((1-\epsilon), \mathit{poly}\log n)$-good, implying a quasi-polynomial time approximation scheme (QPTAS) for the independent set of polygons.

Since the existence of a γ-fraction guillotine separable subset implies that the family is $(\gamma, 4)$-good, our results directly contribute to a better understanding of this framework for squares. The recent breakthrough by Mitchell [14] show that rectangles are $(1/10, 5)$-good, and further refinements [11] prove that they are $(\frac{1}{2} - \epsilon, O(1/\epsilon))$-good, leading to the best known $(2+\epsilon)$-approximation algorithm for the maximum independent set of rectangles.

1.2 Overview of Techniques

This paper builds on the techniques from [1]. There, the guillotine cut sequence is created in two phases. In the first phase, which we call the **grid drawing phase**, the procedure draws hierarchical grid lines so that each square is assigned its own cell roughly the size of its own. Those squares that do not fit this criterion would be removed from the first phase. We call the squares that remain after this phase the surviving squares. Denote by γ_1 the surviving ratio of the first phase. In the second phase, called **conflict removal phase**, we define a conflict graph H where the vertices correspond to the surviving squares and the edges connect the squares that are "difficult" to save simultaneously. The graph H is defined so that every independent set in this graph corresponds to squares that can be guillotine-separated. Therefore, the crux of this approach is to ensure that the conflict graph is sufficiently sparse, so that the graph is k-colorable for some small value of k. This will imply a guillotine-separable set of size γ_1/k.

Using this terminology, the best known factor of 160 [12] ensures that the conflict graph is 5-colorable and loses the factor of 32 in the grid drawing phase [12]; the earlier work by Abed et al. [1] also aims for a simple conflict removal phase, and therefore losing a relatively large factor in the grid drawing phase. Our improvements are obtained by looking at this differently. We ensure that the first phase has a good surviving rate (i.e., our $\gamma_1 = 9/16$), thus naturally complicating the second phase. Our conflict graph is more sophisticated and is no longer 5-colorable. We, nevertheless, show that the graph is 16-colorable, which implies the factor of $9/256$ as claimed.

The paper is written so that the ideas are introduced one by one. We first present the proof where $\gamma_1 = 1/4$ and $k = 15$ (so the factor is $1/60$) and improve k to 9 and 8 respectively (while keeping $\gamma_1 = 1/4$). Finally, we present the main result with $\gamma_1 = 9/16$ and $k = 16$.

2 Preliminaries

Assume we are given a set of non-overlapping axis-parallel squares of arbitrary sizes. We aim to separate as many of them as possible with a sequence of axis-parallel cuts, defined as follows.

Guillotine cuts are a series of horizontal and vertical cuts separating objects on a plane, such that each cut is performed on a previously separated part. For example, if the first vertical cut separates the plane into two, the next two horizontal cuts can separate the two pieces independently.

Given a collection of non-overlapping squares $\mathcal{R}$, $|\mathcal{R}| = n$ on a plane, the problem is to find the largest set $R \subseteq \mathcal{R}$ separable by Guillotine cuts.

2.1 Grid construction

In this section, we give the random hierarchical grid construction used in all the separation algorithms. Let M be the maximum width/height of a square in $\mathcal{R}$. Instead of width/height, we will say the **size** of a square.

We define a hierarchical grid structure similar to Abed et al. [1], which will act as a scaffolding for our instance $\mathcal{R}$. In our cutting strategy, we will perform cuts only along these lines.

We choose two integers $a, b \in [0, M)$ uniformly at random, corresponding to the random *horizontal* and *vertical* shift, respectively. The positive integer r will be a *"scaling factor"*.

Algorithm 1 Grid drawing.

foreach $i \in [1, 2, \ldots]$ **do**
  Draw vertical grid lines at x coordinates $a + \frac{rM}{2^i} \cdot k$ for every integer k
  Draw horizontal grid lines at y coordinates $b + \frac{rM}{2^i} \cdot k$ for every integer k

We say that a grid line is **level-i** if it is drawn by Algorithm 1 in iteration i. Note that all level-i grid lines are also level-$(i+1)$ grid lines, etc.

▶ **Definition 2** (Cell levels). *Any two consecutive horizontal and vertical grid lines on level-i form a unique square, called a level-i cell.*

▶ **Definition 3** (Square levels). *We say that $S \in \mathcal{R}$ is a level-i square, if its size is within $\left(\frac{M}{2^{i+1}}, \frac{M}{2^i}\right]$.*

▶ **Fact 4.** *A level-i cell is at least r times (and strictly less than $2r$ times) larger than a level-i square.*

▶ **Proposition 5.** *Grid cells satisfy the following properties:*

- *A level-i cell has a size of $\frac{rM}{2^i}$.*
- *Two cells of the same level do not intersect.*
- *For two cells that intersect, one must contain the other.*

We say that C is a **subcell** of D if D contains C, and $D \neq C$ (they are not the same cell). We use $\mathsf{size}(C)$ to refer to the size of cell C and similarly for squares.

2.2 Surviving set $\mathcal{R}_1$

▶ **Definition 6** (Original cells). *Let S be any square and i be the level of S. If S does not intersect a grid line of level i, we define the original cell of S to be the level-i cell that contains S. We denote it as $\mathcal{OC}(S)$.*

Note that $\mathcal{OC}(S)$ is not defined for every square S. The set of squares for which original cells are defined is called the **surviving** set $\mathcal{R}_1$.

▶ **Theorem 7** (Claim 2.1, [1]). *The horizontal and vertical shifts $a, b \in [0, M)$ can be chosen such that the surviving set $\mathcal{R}_1$ is of size at least $(1 - \frac{1}{r})^2|\mathcal{R}|$.*

The proof can be found in Appendix A.

3 Algorithmic Results

3.1 Overview

We will introduce multiple strategies for separating a set of squares by performing guillotine cuts. All of these strategies will follow the basic structure outlined here.

Algorithm 2 Cutting framework.

Choose r.
Choose horizontal and vertical shifts $a, b \in [0, M)$ such that the surviving set $\mathcal{R}_1$ is at least $(1 - \frac{1}{r})^2|\mathcal{R}|$.
For the surviving set of squares, define "a conflicting pair".
Based on the definition:
  Construct a conflict graph $G = (\mathcal{R}_1, E)$, where for all squares $S_1, S_2 \in \mathcal{R}_1$, $\{S_1, S_2\} \in E$, if and only if S_1 and S_2 conflict.
  Color G with β colors, such that each color (independent set) contains no conflicting pairs.
  Show that any independent set of G can be the result of a sequence of guillotine cuts.

This strategy leads to a set of size at least $(1 - \frac{1}{r})^2 \cdot \frac{1}{\beta} \cdot |\mathcal{R}|$. The specific ratio depends on r and, most importantly, on how we define a conflict, which will gradually be improved in each section, resulting in increasingly larger sets.

3.2 A direct cutting strategy to get 1/60-separation

In this subsection, we fix $r = 2$, define the conflict graph G_1 on $\mathcal{R}_1$, and show that it is 15-colorable. Thus, the cutting sequence separating any independent set of G_1 will imply a 1/60-separation.

We say that two squares $S_1, S_2 \in \mathcal{R}_1$ *conflict* if the original cells of S_1 and S_2 are the same, i.e., $\mathcal{OC}(S_1) = \mathcal{OC}(S_2)$, or we have the case that $\mathcal{OC}(S_1)$ contains $\mathcal{OC}(S_2)$ and S_1 intersects $\mathcal{OC}(S_2)$. Note that S_1 and S_2 do not conflict if $\mathcal{OC}(S_1)$ and $\mathcal{OC}(S_2)$ are of the same level but not the same cells. Figure 1 shows examples of conflicting and non-conflicting squares.

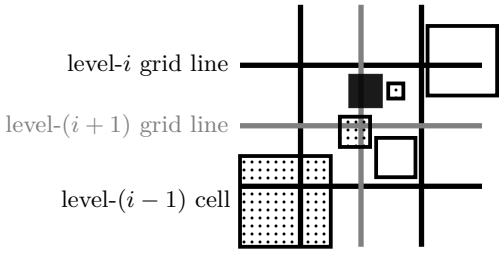


Figure 1 Dot-filled squares conflict with the black square, but the empty squares do not. The original cell of the black square is the black level-i cell.

Now, we define the *conflict graph* G_1, so that the vertices are the squares in $\mathcal{R}$ and two vertices are adjacent if they conflict.

For any cell C, we denote by $\mathsf{Squares}_{\mathcal{OC}}(C)$ the set of all squares $S \in \mathcal{R}_1$, such that $\mathcal{OC}(S) = C$. Note that some higher- or lower-level squares might be drawn inside C but not contained in $\mathsf{Squares}_{\mathcal{OC}}(C)$.

▶ **Lemma 8.** *The conflict graph G_1 is 15-colorable.*

Proof. We prove it by induction on the size of G_1. If G_1 is empty, it is obviously 15-colorable. Otherwise, we take the smallest non-empty cell C_1 such that $\mathsf{Squares}_{\mathcal{OC}}(C_1) \neq \emptyset$, i.e. there exists a square $S \in \mathcal{R}_1$ such that $\mathcal{OC}(S) = C_1$, and for any cell C_2 contained in C_1, we have $\mathsf{Squares}_{\mathcal{OC}}(C_2) = \emptyset$. Let i be the level of cell C_1.

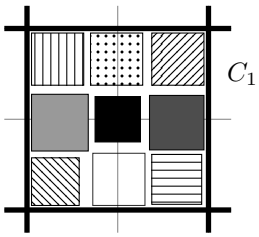


Figure 2 We color nine squares contained in C_1 using different colors.

According to the induction hypothesis, $G_1 \setminus \mathsf{Squares}_{\mathcal{OC}}(C_1)$ is 15-colorable, so let us assume it is already colored. We show that we can color all the squares in $\mathsf{Squares}_{\mathcal{OC}}(C_1)$ to get a 15 coloring for G_1. These squares all have to have a level at most i; otherwise, their original cell would have been smaller, and we wouldn't have chosen C_1. According to Fact 4, the size of level-i squares is strictly bigger than $\frac{1}{2r} = \frac{1}{4}$ of the size of C_1, and therefore at most 3 squares of level i fit into a cell of level i horizontally as well as vertically. Since squares of smaller levels are larger, it follows that at most 9 squares can be inside C_1 (Figure 2). If C_1 contains squares of smaller levels, they will already be colored by the induction hypothesis, and these colors will not be changed. We can color all the squares in $\mathsf{Squares}_{\mathcal{OC}}(C_1)$ with at most nine different colors (see Figure 2), avoiding the already used colors inside C_1.

Now, we bound the number of squares that conflict with any square in $\mathsf{Squares}_{\mathcal{OC}}(C_1)$ but are not contained in C_1. Since C_1 is the smallest non-empty cell, any square $S \in \mathcal{R}_1$ must cross the boundary of C_1. Since C_1 is a level-i cell, then $\mathcal{OC}(S)$ is a level-j cell for $j < i$. We bound the number of squares that can conflict with any square in $\mathsf{Squares}_{\mathcal{OC}}(C_1)$ based on the way they intersect C_1 (see Figure 3a): First, there can be at most four squares containing one of the corners of C_1. Next, let S be a square intersecting the side of C_1 without containing any corner. Using Fact 4, we know that the size of S is strictly bigger than half of the size of C_1. Thus, there can be at most one square in this position for each side of C_1, which adds up to four squares. We can further reduce this number because two sides of any level-i cell are also level-$(i-1)$ grid lines. Then, there cannot be any intersecting square on those sides, implying that at most two squares intersecting only sides of C_1 (see Figure 3b).

(a) Without considering the geometry of the cells.

(b) Such squares shouldn't cross the level-$(i-1)$ grid lines.

Figure 3 The dot-filled squares show the possible positions of squares not contained in cell C (darkened area) and conflict with the black square in C.

We showed that at most six squares can conflict with squares in $\mathsf{Squares}_{\mathcal{OC}}(C_1)$ by intersecting the boundary of C_1 and at most 9 colors can be required to color the squares inside C_1, leading to a valid 15 coloring for G_1. ◀

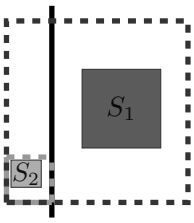


Figure 4 The dashed squares are $\mathcal{OC}(S_1)$ and $\mathcal{OC}(S_2)$, and black-line cut separates S_1 and S_2.

Figure 5 For the black square $\mathcal{FC}(S) = \mathcal{OC}(S)$, whereas for the grey square $\mathcal{FC}(S) \neq \mathcal{OC}(S)$.

▶ **Lemma 9.** *Any independent set of G_1 is guillotine separable.*

Proof. Let $\mathcal{I} \subseteq \mathcal{R}_1$ be an independent set of G_1. First, we can separate the two squares with the "biggest" original cells without cutting through any other squares. Let $S_1, S_2 \in \mathcal{I}$ be the squares such that $\mathsf{size}(\mathcal{OC}(S_1)) \geq \mathsf{size}(\mathcal{OC}(S_2))$ and for any other square $S \in \mathcal{I}$, $\mathsf{size}(\mathcal{OC}(S_2)) \geq \mathsf{size}(\mathcal{OC}(S))$. Note that since S_1 and S_2 cannot conflict, then $\mathcal{OC}(S_1) \neq \mathcal{OC}(S_2)$.

Since the grid lines defining $\mathcal{OC}(S_2)$ cannot go through any cell smaller than itself, we can safely cut along these grid lines and do not touch any other squares in $\mathcal{I}$ except for S_1. Now, we claim that one of the four grid lines defining $\mathcal{OC}(S_2)$ separates S_1 and S_2 (see Figure 4). Since S_1 cannot intersect $\mathcal{OC}(S_2)$, it must be entirely on the left, right, top, or bottom of $\mathcal{OC}(S_2)$. The grid line of the corresponding side then separates S_1 from S_2 without cutting any square in $\mathcal{I}$. We can separate all the squares by recursively and independently applying this process to the two resulting sub-instances. ◀

3.3 Fitting Cells to get $1/44$-separation

In this section, we improve coloring by using the additional concept of fitting cells. The grid's scaling factor remains 2.

Note that the grid structure defined in Section 2 forms a **laminar family** of cells. Let the set of these cells be $\mathcal{L}_1$. For any laminar family of cells, we define the notion of fitting cells as follows.

▶ **Definition 10** (Fitting cells). *Given a laminar family of cells $\mathcal{L}_1$ and a surviving set $\mathcal{R}_1$, the fitting cell $\mathcal{FC}(S)$ of any square $S \in \mathcal{R}_1$ is the smallest cell in $\mathcal{L}_1$ containing S.*

The fitting and original cells of square $S \in \mathcal{R}_1$ can differ when the former is too small to qualify as an original cell because of the scaling factor $r = 2$ (Figure 5).

▶ **Fact 11.** *For a level-i square S, the fitting cell $\mathcal{FC}(S)$ is always of level j, for $i \leq j \leq i+1$.*

Proof. The bound $j \leq i + 1$ is a direct result of the scaling factor: for a cell C of level i, $\mathsf{size}(S) > \frac{1}{4}\,\mathsf{size}(C)$ (Fact 4), therefore for a cell C_1 of level $i + 1$, we would have $\mathsf{size}(S) > \frac{1}{2}\,\mathsf{size}(C_1)$ and for a level-$(i + 2)$ cell C_2, $\mathsf{size}(S) > \mathsf{size}(C_2)$. This implies that C_2 cannot be a fitting cell of S, bounding j at most $i + 1$. The first part of the inequality, $i \leq j$, is easy to see since every original cell must contain S, so the smallest cell that contains S cannot be larger than $\mathcal{OC}(S)$. ◀

Conflict graph G_2. We define the new conflict graph G_2 based on the definition of conflicts below.

▶ **Definition 12** (Conflicts). *Let S_1 and S_2 be distinct squares in $\mathcal{R}_1$. We say that S_1 and S_2 conflict if one of the following conditions is true: (i) $\mathcal{FC}(S_1) = \mathcal{FC}(S_2)$, or (ii)* $\mathsf{size}(\mathcal{FC}(S_1)) > \mathsf{size}(\mathcal{FC}(S_2))$ *and S_1 intersects $\mathcal{FC}(S_2)$.*

Figure 6 shows examples of conflicting and non-conflicting squares.

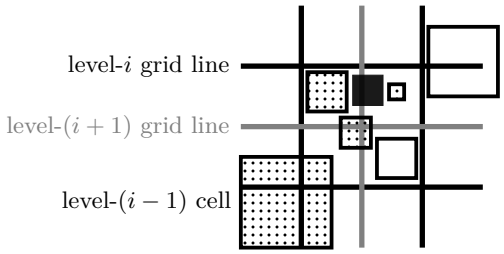


Figure 6 Dot-filled squares conflict with the black square, and empty squares do not. The fitting cell of the black square is the black level-i cell.

Analogously to $\mathsf{Squares}_{\mathcal{OC}}$ for original cells, we define $\mathsf{Squares}_{\mathcal{FC}}$ for fitting cells.

▶ **Definition 13.** *Let C be a cell of $\mathcal{L}_1$. We use $\mathsf{Squares}_{\mathcal{FC}}(C)$ to denote the set of all squares $S \in \mathcal{R}_1$, such that $\mathcal{FC}(S) = C$.*

▶ **Lemma 14.** *The conflict graph G_2 is 11-colorable.*

The proof can be found in Appendix A.

▶ **Lemma 15.** *Any independent set of G_2 is guillotine-separable.*

The proof can be seen by replacing the concept of original cells in Lemma 9 with fitting cells.

4 Further Improvements for cutting Squares

4.1 Twin squares to reach 1/36-separation

In this subsection, we introduce the notion of *twin squares*. The idea is to identify pairs of squares (belonging to the same fitting cell) that can be separated using the next level's vertical or horizontal grid line. We will get a sparser conflict graph G_3 on $\mathcal{R}_1$ by removing conflicts between such squares. We show that the new conflict graph G_3 is 9-colorable, and any independent set of this graph is guillotine separable, implying a 1/36-separation for $\mathcal{R}$. We start with the same set $\mathcal{R}_1$, laminar family $\mathcal{L}_1$, and the fitting cells defined in Section 3.3.

▶ **Definition 16** (Twin squares). *For a laminar family $\mathcal{L}_1$, Let S_1 and S_2 be two squares with the same level-i fitting cell, i.e. $\mathcal{FC}(S_1) = \mathcal{FC}(S_2)$. We say S_1 and S_2 are* twin squares *if they can be separated by the level-$(i+1)$ horizontal or vertical grid line cutting through $\mathcal{FC}(S_1)$ (see Figure 7).*

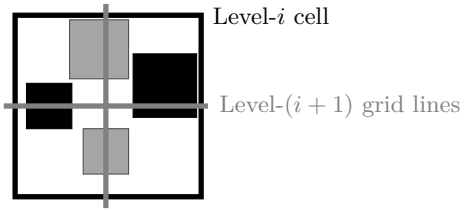


Figure 7 The two gray and two black squares all have a common fitting cell.

Conflict graph G_3. We define the new conflict graph G_3 based on the definition of conflicts below.

▶ **Definition 17** (Conflicts). *Let S_1 and S_2 be two squares in $\mathcal{R}_1$. We say that S_1 and S_2 conflict (up to renaming) if one of the following conditions is true:*

- *$\mathcal{FC}(S_1) = \mathcal{FC}(S_2)$, and S_1 and S_2 are not twin squares.*
- *$\mathcal{FC}(S_1)$ contains $\mathcal{FC}(S_2)$, and S_1 intersects $\mathcal{FC}(S_2)$.*

▶ **Lemma 18.** *The conflict graph G_3 is 9-colorable.*

The proof can be found in Appendix A.

▶ **Lemma 19.** *Any independent set of G_3 is guillotine separable.*

Proof. Similarly to Lemma 9, we show a recursive cutting strategy. Starting with an independent set $\mathcal{R}_2$ of G_3, we first separate two squares with the "biggest" fitting cell. Let $S_1, S_2 \in \mathcal{R}_2$ be the squares such that $\mathsf{size}(\mathcal{FC}(S_1)) \geq \mathsf{size}(\mathcal{FC}(S_2))$, and for any other square $S \in \mathcal{R}_2$, $\mathsf{size}(\mathcal{FC}(S_2)) \geq \mathsf{size}(\mathcal{FC}(S))$. If $\mathcal{FC}(S_1) \neq \mathcal{FC}(S_2)$, then S_1 and S_2 can be separated using the proof of Lemma 9 (see Figure 4). Otherwise, S_1, S_2 must be twin squares with a common fitting cell (let $C = \mathcal{FC}(S_1)$), since they do not conflict. Then, we can cut along the grid lines of C (see Figure 8). Since for any square $S \in \mathcal{R}_2$, $\mathsf{size}(C) \geq \mathsf{size}(\mathcal{FC}(S))$, we can safely perform these cuts without intersecting any square in $\mathcal{R}_2$.

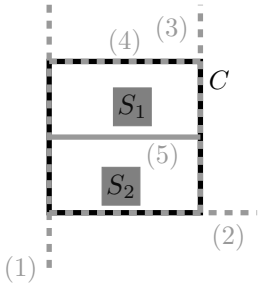


Figure 8 Cutting to separate twin squares; we first cut the dashed grey lines 1 to 4 to isolate C. We cut along the 5^{th} light grey line to separate S_1 and S_2. No square can intersect the last cut.

Let i be the level of C. Since we have isolated C with the first 4 cuts, further cuts of C can be done independently from the rest of the graph. We use the horizontal (or vertical) level-$(i+1)$ grid line, which certifies that S_1 and S_2 are twins (see Definition 16), as a cut to separate S_1 from S_2 (see Figure 8). Let $S \in \mathcal{R}_2$ be any square inside C intersecting the cut. Since S is contained in C then $\mathsf{size}(\mathcal{FC}(S)) \leq \mathsf{size}(C)$. S also cannot have a level-$(i+1)$ (or smaller) cell as its fitting cell because of intersecting the level-$(i+1)$ grid line. Therefore $\mathcal{FC}(S) = C$. Since twin squares S_1 and S_2 intersect the vertical level-$(i+1)$ grid line and

their fitting cell is also C, then S conflicts with them. This implies that $S \notin \mathcal{R}_2$ and the cut is safe to perform. Similarly to the proof of Lemma 9, we recursively apply this process to separate $\mathcal{R}_2$. ◀

4.2 Half-Cells for a 1/32-separation

In this section, we redefine the notion of cells to include rectangular cells.

▶ **Definition 20** (Rectangular Cells). *Any two consecutive horizontal and vertical grid lines of level i and j form a unique rectangle, called a level-(i, j) cell.*

Let *square-cells* be all level-(i, i) cells for some i (our old notion of cells), and *half-cells* be all level-$(i, i+1)$ cells for some i. Note that half-cells of level $(i, i+1)$ can be obtained by splitting square-cells of level (i, i) with the level-$(i+1)$ vertical line within them.

We will use $\mathsf{height}(C)$ and $\mathsf{width}(C)$ to refer to the height and width of a cell C, respectively. For any pair of cells C_1, C_2, we will say $\mathsf{size}(C_1) \leq \mathsf{size}(C_2)$, if $\mathsf{height}(C_1) \leq \mathsf{height}(C_2)$ and $\mathsf{width}(C_1) \leq \mathsf{width}(C_2)$. Note that among square-cells and half-cells, any pair of cells C_1, C_2 is comparable (either $\mathsf{size}(C_1) \leq \mathsf{size}(C_2)$ or $\mathsf{size}(C_2) \leq \mathsf{size}(C_1)$), since we are only considering the tall halves. Adding half-cells to the laminar family $\mathcal{L}_1$, the resulting family $\mathcal{L}_2$ remains laminar. In this section, let fitting cells be defined over the laminar family $\mathcal{L}_2$ (Definition 10), while the scaling factor r remains 2. Examples of the new fitting cells are depicted on Figure 9.

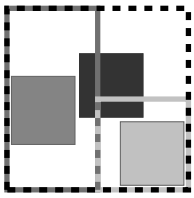


Figure 9 The black dashed square-cell is the original cell of all the squares. The fitting cell of the black square is its original cell. The dark and light gray squares have a half-cell and a higher-level square-cell as their fitting cells, respectively.

▶ **Fact 21.** *For a level-i square S, the fitting cell $\mathcal{FC}(S)$ is*

- *A square-cell of level j, for $i \leq j \leq i+1$, or*
- *A half-cell of level $(i, i+1)$*

Proof. The first part follows directly from Fact 11. Let us consider the second part. When we define fitting cells over $\mathcal{L}_2$, the surviving set remains the same; therefore, when the fitting cell is a half-cell, it is always two times narrower than if we defined fitting cells over $\mathcal{L}$. Similarly to the proof of Fact 11, a half-cell C of level $(i, i+1)$ has $\mathsf{width}(S) > \frac{1}{2}\,\mathsf{width}(C)$, therefore a half-cell C_1 of level $(i+1, i+2)$ would not be able to contain S. ◀

Twin squares. Similarly to the original definition Definition 16, we define twin squares $S_1, S_2 \in \mathcal{R}_1$ to be a pair of squares in the same fitting cell C_1 (which is a level-i square-cell) that are separated by a level-$(i+1)$ grid line.

If their fitting cell is a half-cell, we do not consider $S_1, S_2 \in \mathcal{R}_1$ twin squares. Additionally, twin squares $S_1, S_2 \in \mathcal{R}_1$ cannot be separated by a vertical grid line, otherwise their fitting cells would be separate half-cells.

Conflict graph G_4. We define the new conflict graph G_4 based on the same Definition 17 of conflicts with respect to the new fitting cells of $\mathcal{L}_2$. See Figure 10 for illustration.

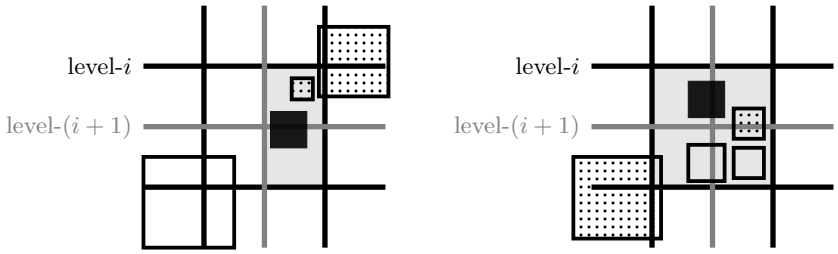


Figure 10 The dot-filled squares conflict with the black square, and the empty squares do not. The fitting cell of the black square is the shaded half-cell on the left and the shaded square-cell on the right.

▶ **Lemma 22.** *The conflict graph G_4 is 8-colorable.*

Proof. This proof is easier to show by using a stronger statement. We will show that there exists an 8-coloring of G_4, such that for each cell $C \in \mathcal{L}_2$, the squares $\mathsf{Squares}_{\mathcal{FC}}(C)$ are colored with at most 2 colors.

We will see this proof by induction over the number of squares ($|V(G_4)|$). The base case holds trivially. We again pick the smallest non-empty cell $C_1 \in \mathcal{L}_2$ for induction. Then $\mathsf{Squares}_{\mathcal{FC}}(C_1) \neq \emptyset$ and for any cell C_2 smaller than C_1 ($\mathsf{size}(C_2) \leq \mathsf{size}(C_1)$), we have $\mathsf{Squares}_{\mathcal{FC}}(C_2) = \emptyset$.

We assume that the induction hypothesis holds for $G_4 \setminus \mathsf{Squares}_{\mathcal{FC}}(C_1)$ and show that we can use 2 colors to color $\mathsf{Squares}_{\mathcal{FC}}(C_1)$, with at most eight colors in total. We will consider the cases where C_1 is a square-cell or a half-cell separately.

Firstly, consider that C_1 is a square-cell of level i. Equivalently to Lemma 8, it can be seen that there are at most 6 squares that can conflict with any square in $\mathsf{Squares}_{\mathcal{FC}}(C_1)$. Since these squares use at most 6 colors, at least 2 colors remain to color $\mathsf{Squares}_{\mathcal{FC}}(C_1)$. We will count the number of squares C_1 contains, which, similarly to previous proofs, is the same as $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ due to how C_1 was chosen. All squares in C_1 must intersect the vertical grid line of level $i+1$, splitting C_1; otherwise, those squares would be contained in a subcell of C_1. According to Fact 4, these squares can be at most 3. These squares can be colored with 2 colors since the squares above and below the horizontal grid line of the level $i+1$ are twin squares and can share a color (Figure 11).

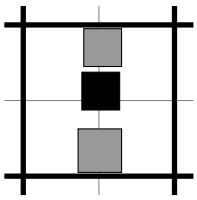


Figure 11 Two colors (black and gray) suffice to color all squares contained in C_1 since if squares are above and below the horizontal grid line crossing C_1, these are twin squares.

Secondly, we consider that C_1 is a level-$(i, i+1)$ half-cell. Again, the squares that C_1 contains among $\mathcal{R}_1$ are exactly $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ due to how C_1 was chosen. These squares must intersect the horizontal grid line of level $i+1$ within C_1; otherwise, they would fit

into a subcell of C_1. The squares in $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ must be of level i to have survived, and the number of level-i squares intersecting the level-$(i+1)$ grid line in C_1 can be at most 1 (Fact 21, Fact 4). Since $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ cannot be empty, it must contain exactly one square. Let S be that square.

It remains to bound the number of squares conflicting with S. Note that all the conflicting squares must intersect the boundary of C_1 (Definition 17). Let C_0 be the level-i square-cell such that C_1 is a subcell of C_0.

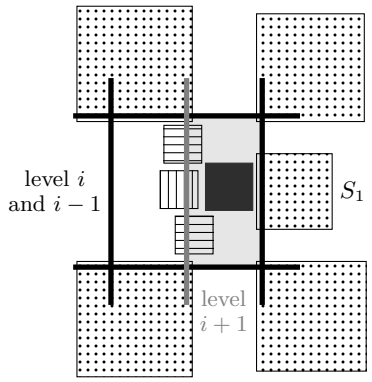


Figure 12 Cell C_1 is shaded, and cell C_0 is bordered by the black grid lines of level i. We depict that the left vertical grid line of level i is also of level $i-1$.

At most, 4 squares can intersect with the corners of cell C_1. Next, consider the squares intersecting the level-i horizontal grid lines bordering C_1. For these squares to survive, their level must be at most $i-1$. This implies they are too large to intersect only the level-i horizontal grid line without intersecting a corner (Figure 12). Next, let us consider the level-i vertical grid lines bordering C_0. Similarly to Lemma 8, Figure 3b, one of those grid lines must also be of level $i-1$ and therefore cannot have a surviving square intersecting it without also intersecting a corner. This leaves us with one square S_1 as depicted on Figure 12. Finally, let us consider the squares intersecting the level-$(i+1)$ vertical grid line bordering C_1 (Figure 12). Since we already counted the squares intersecting the borders of C_0 in addition to C_1, these squares must be in $\mathsf{Squares}_{\mathcal{FC}}(C_0)$. By induction hypothesis, they can be colored with 2 colors. If the colors of the corner squares and S_1 are different from these two colors, we have at most 7 colors used for the set of squares that conflict with S. By coloring S with a different color, we have a valid 8-coloring with at most 2 colors used for $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ as stated. ◀

▶ **Lemma 23.** *Any independent set of G_4 is guillotine separable.*

Proof. Similarly to Lemma 19, we show a recursive cutting strategy. Starting with an independent set $\mathcal{R}_2$ of G_4, we first separate two squares with the "biggest" fitting cell. Let $S_1, S_2 \in \mathcal{R}_2$ be the squares such that $\mathsf{size}(\mathcal{FC}(S_1)) \geq \mathsf{size}(\mathcal{FC}(S_2))$, and for any other square $S \in \mathcal{R}_2$, $\mathsf{size}(\mathcal{FC}(S_2)) \geq \mathsf{size}(\mathcal{FC}(S))$. If $\mathcal{FC}(S_1) \neq \mathcal{FC}(S_2)$, then S_1 and S_2 can be separated using the proof of Lemma 9 (see Figure 4). Next, we consider the case that $\mathcal{FC}(S_1) = \mathcal{FC}(S_2)$, e.g S_1 and S_2 share the same fitting cell. Let this fitting cell be C and its level be i.

First, consider the case that C is a half-cell. This implies that S_1 and S_2 intersect the same horizontal grid line of level $i+1$. Otherwise, they would fit in smaller subcells. Since there are no twin squares in half-cells, S_1 and S_2 conflict, which contradicts them being in the same independent set $\mathcal{R}_2$.

Secondly, consider the case that C is a square-cell. Similarly to the proof of Lemma 19, the square-cell can be isolated from the rest of the graph by the first 4 cuts as depicted on Figure 8. Again, these cuts will not intersect any squares outside of C, because the fitting cells for other squares must be smaller than or equal to the size of C (due to our choice of S_1 and S_2). Similarly to Lemma 19, the fifth cut is a level-$(i+1)$ horizontal grid line if i is the level of C. Let us now consider the squares inside C since the fifth cut will be performed only on C. For $\mathsf{Squares}_{\mathcal{FC}}(C)$, we can see similarly to Lemma 19 that they are not cut. For a square S whose fitting cell is a half-cell subcell of C, there is a conflict between S and S_i, because $\mathcal{FC}(S_1) \geq \mathcal{FC}(S)$ and S_1 intersects $\mathcal{FC}(S)$. This means that S cannot exist in the independent set $\mathcal{R}_2$, and the cutting strategy doesn't cut any squares.

Similarly to the proof of Lemma 9, we recursively apply this process to separate $\mathcal{R}_2$. ◀

5 Using Coarser Grid ($r = 4$) for Squares to get $\frac{9}{256}$-separation

We consider a set $\mathcal{R}$ containing n non-overlapping axis-parallel squares. In Theorem 27, we show that we can separate at least $\frac{9}{256}$ fraction of the squares with guillotine cuts.

For the coarser grid, we set $r = 4$, and get $\frac{9}{16}$ fraction of surviving squares $\mathcal{R}_1 \in \mathcal{R}$ by applying Theorem 7. Then, we introduce a conflict graph G_5, showing that it is 16-colorable. Finally, we show that any independent set of this graph is Guillotine separable, leading to a 9/256-separation.

Similar to Section 4.2, we consider the laminar family $\mathcal{L}_3$ containing all the level-i cells together with all the level-$(i, i+1)$ half-cells for any $i \geq 0$. For each square S, we define the fitting cell $\mathcal{FC}(S)$ over the laminar family $\mathcal{L}_3$ (Definition 10).

Since, in this case, $r = 4$, the number of squares associated with a fitting cell increases. In this section, let the surviving set $\mathcal{R}_1$ be redefined to be the surviving set of the new grid structure.

▶ **Fact 24.** *For any square $S \in \mathcal{R}_1$, one of the following must be true:*

- *The fitting cell $\mathcal{FC}(S)$ is a level-i square-cell for some i. Then S intersects the level-$(i+1)$ vertical grid line, cutting $\mathcal{FC}(S)$ in half, and $\mathcal{FC}(S)$ can be a fitting cell for at most seven squares (see Figure 13a).*
- *The fitting cell $\mathcal{FC}(S)$ is a level-$(i, i+1)$ half-cell for some i. Then S intersects the level-$(i+1)$ horizontal grid line, cutting $\mathcal{FC}(S)$ in half, and $\mathcal{FC}(S)$ can be a fitting cell for at most three squares (see Figure 13b).*

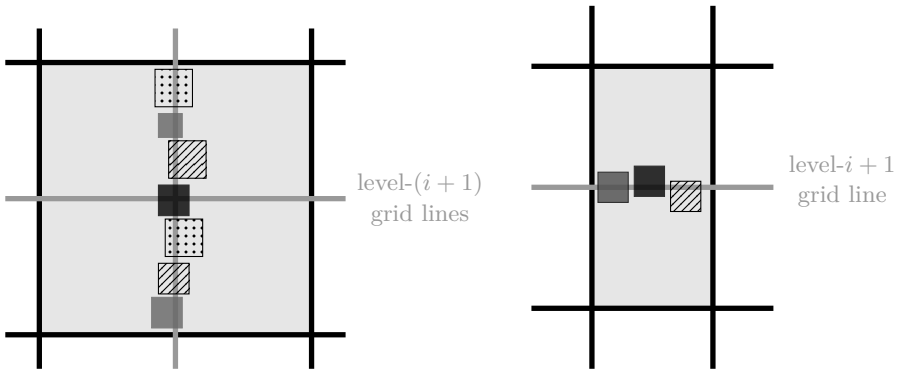
Proof. If the intersection property did not hold for either of the cases above, the square S would fit into some smaller cell in $\mathcal{L}_3$, leading to a contradiction.

Since S intersects a level-$(i+1)$ grid line in both cases, S must be level-j for $j \leq i$. Also, $\mathsf{size}(S)$ must be strictly more than $\frac{1}{8}$ fraction of the height of any level-i cell, because of the Fact 4.

In case $\mathcal{FC}(S)$ is of level i, any such square must intersect the level-$(i+1)$ vertical grid line cutting $\mathcal{FC}(S)$ in half. Hence, at most seven squares can fit into $\mathcal{FC}(S)$.

Otherwise, if $\mathcal{FC}(S)$ is a level-$(i, i+1)$ cell, the width of $\mathcal{FC}(S)$ is half the width of any level-i cell. Since S must intersect the level-$(i+1)$ horizontal grid line cutting $\mathcal{FC}(S)$ into half, then at most three squares can fit into $\mathcal{FC}(S)$. ◀

Conflict Graph G_5. We use the same Definition 17 of conflicting squares to define our conflict graph G_5 on $\mathcal{R}_1$. Similar to Section 4.2, there are no twin squares separated by the vertical line cutting through the cell; otherwise, the two squares would have different fitting



(a) Square-cell. **(b)** Half-cell.

Figure 13 A coloring for the squares in $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ when C_1 is a level-i square-cell or level-$(i, i+1)$ half-cell on the left and right respectively. C_1 is shaded.

cells. The following lemma shows the coloring of G_5, similar to the proof of Lemma 22, involving more counting arguments. Since we have $r = 4$ instead of $r = 2$, this grid is coarser, leading to more squares inside and intersecting the boundary of any cell C in $\mathcal{L}_3$.

▶ **Lemma 25.** *The graph G_5 is 16-colorable.*

The proof can be found in Appendix A.

▶ **Lemma 26.** *Any independent set of G_5 is guillotine separable.*

The proof of this lemma is similar to the proof of Lemma 23 in Section 4.

▶ **Theorem 27.** *For an n-size set $\mathcal{R}$ of non-overlapping axis-parallel squares, there exists a $\frac{9}{256}$-separation with guillotine cuts.*

Proof. Using Theorem 7, we can place a grid on $\mathcal{R}$ and define the original cells for a subset $\mathcal{R}_1 \subset \mathcal{R}$ containing at least $\frac{9n}{16}$ squares, with $r = 4$. Then, Lemma 25 ensures that the conflict graph G_5 for the squares is 16-colorable. Thus, we can extract a subset $\mathcal{R}_2 \subset \mathcal{R}_1$ containing at least $\frac{9n}{256}$ squares, which do not conflict with any other square in $\mathcal{R}_2$.

Finally, Lemma 26 provides a cutting strategy for $\mathcal{R}_2$, showing that $\mathcal{R}_2$ is guillotine separable. ◀

References

1 Fidaa Abed, Parinya Chalermsook, José Correa, Andreas Karrenbauer, Pablo Pérez-Lantero, José A. Soto, and Andreas Wiese. On Guillotine Cutting Sequences. In Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, volume 40 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1–19, Dagstuhl, Germany, 2015. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.APPROX-RANDOM.2015.1.

2 Anna Adamaszek, Sariel Har-Peled, and Andreas Wiese. Approximation schemes for independent set and sparse subsets of polygons. *Journal of the ACM (JACM)*, 66(4):29, 2019.

3 Anna Adamaszek and Andreas Wiese. Approximation schemes for maximum weight independent set of rectangles. In *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, FOCS '13, pages 400–409. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.50.

4 Anna Adamaszek and Andreas Wiese. *A QPTAS for Maximum Weight Independent Set of Polygons with Polylogarithmically Many Vertices*, pages 645–656. Society for Industrial and Applied Mathematics, 2014. doi:10.1137/1.9781611973402.49.

5 Mir-Bahador Aryanezhad, Nima Fakhim Hashemi, Ahmad Makui, and Hasan Javanshir. A simple approach to the two-dimensional guillotine cutting stock problem. *Journal of Industrial Engineering International*, 8(1):21, September 2012.

6 Parinya Chalermsook and Julia Chuzhoy. Maximum independent set of rectangles. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 892–901. Society for Industrial and Applied Mathematics, 2009. doi:10.1137/1.9781611973068.97.

7 Parinya Chalermsook and Bartosz Walczak. Coloring and maximum weight independent set of rectangles. In *Proceedings of the 2021 ACM-SIAM symposium on discrete algorithms (SODA)*, pages 860–868. SIAM, 2021. doi:10.1137/1.9781611976465.54.

8 Timothy M Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012. doi:10.1007/S00454-012-9417-5.

9 CH Cheng, BR Feiring, and TCE Cheng. The cutting stock problem – A survey. *International Journal of Production Economics*, 36(3):291–305, 1994.

10 Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM Journal on Computing*, 34(6):1302–1323, 2005. doi:10.1137/S0097539702402676.

11 Waldo Gálvez, Fabrizio Grandoni, Arindam Khan, Diego Ramírez-Romero, and Andreas Wiese. Improved approximation algorithms for 2-dimensional knapsack: Packing into multiple l-shapes, spirals, and more. *arXiv preprint arXiv:2103.10406*, 2021. arXiv:2103.10406.

12 Arindam Khan and Madhusudhan Reddy Pittu. On Guillotine Separability of Squares and Rectangles. In Jaroslaw Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*, volume 176 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:22, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.APPROX/RANDOM.2020.47.

13 Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems: A survey. *European journal of operational research*, 141(2):241–252, 2002. doi:10.1016/S0377-2217(02)00123-6.

14 Joseph Mitchell. Approximating maximum independent set for rectangles in the plane. *SIAM Journal on Computing*, 0(0):FOCS21–299–FOCS21–322, 0. doi:10.1137/22M1475521.

15 Nthabiseng Ntene and Jan H van Vuuren. A survey and comparison of guillotine heuristics for the 2d oriented offline strip packing problem. *Discrete Optimization*, 6(2):174–188, 2009. doi:10.1016/J.DISOPT.2008.11.002.

16 János Pach and Gábor Tardos. Cutting glass. In *Proceedings of the Sixteenth Annual Symposium on Computational Geometry*, SoCG '00, pages 360–369. ACM, 2000. doi:10.1145/336154.336223.

17 Jorge Urrutia. Problem presented at ACCOTA'96: Combinatorial and computational aspects of optimization. *Topology, and Algebra, Taxco, Mexico*, 1996.

A Deferred proofs

▶ **Theorem 7** (Claim 2.1, [1]). *The horizontal and vertical shifts $a, b \in [0, M)$ can be chosen such that the surviving set $\mathcal{R}_1$ is of size at least $(1 - \frac{1}{r})^2|\mathcal{R}|$.*

Proof. We will show that this result can be reached in expectation by sampling $a, b \in [0, M)$ independently and uniformly at random.

Let $S \in \mathcal{R}$ be any square of level i. Let P_h and P_v be the probabilities that a level-i horizontal or vertical grid line is drawn on S, respectively. By using Fact 4, we get that $P_h, P_v \leq \frac{1}{r}$. Thus, the probability that S is in the surviving set (survives), is $(1-P_h)(1-P_v) \geq (1-\frac{1}{r})^2$. In expectation, $(1-\frac{1}{r})^2|\mathcal{R}|$ rectangles survive. This implies that there must exist a and b such that at least $(1-\frac{1}{r})^2|\mathcal{R}|$ squares survive. ◀

▶ **Lemma 14.** *The conflict graph G_2 is 11-colorable.*

Proof. Similar to Lemma 8, we will prove this by induction on the number of squares ($|V(G_2)|$). The base case is trivial. For the inductive step, consider the smallest cell C_1 that contains a square ($\mathsf{Squares}_{\mathcal{FC}}(C_1) \neq \emptyset$). This implies that for any cell C_2 smaller than C_1, $\mathsf{Squares}_{\mathcal{FC}}(C_2) = \emptyset$. By induction hypothesis, $G_2 \setminus \mathsf{Squares}_{\mathcal{FC}}(C_1)$ is 11-colorable, and we will show that we can color the squares of $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ while maintaining a valid 11-coloring of G_2.

First, we bound the number of squares contained within C_1. Let i be the level of C_1. Since C_1 is the smallest cell containing any square, any square S within C_1 must cross the level-$(i+1)$ grid lines within C_1, or otherwise S would be contained in a smaller level-$(i+1)$ cell. According to Fact 11 and Fact 4, at most 3 consecutive squares can fit into C_1 width- or height-wise. Since all contained squares must intersect the level-$(i+1)$ grid lines, it is easy to verify that the number of these squares can be at most 5 in total (Figure 14). Note that this set of squares equals $\mathsf{Squares}_{\mathcal{FC}}(C_1)$.

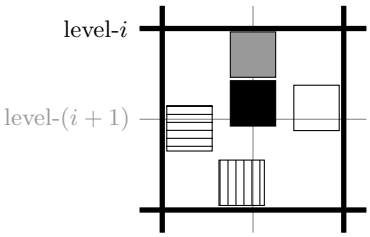


Figure 14 We color five squares of $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ using different colors as shown in the figure.

We count all conflicts involving $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ for coloring. The number of colors that cannot be used for coloring $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ based on the second condition of Definition 12 can be counted equivalently to Lemma 8, resulting in 6 colors that $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ must avoid. Since the counted set of squares contained within C_1 is equal to $\mathsf{Squares}_{\mathcal{FC}}(C_1)$, five additional colors suffice for coloring $\mathsf{Squares}_{\mathcal{FC}}(C_1)$. This gives us a valid 11-coloring of G_2. ◀

▶ **Lemma 18.** *The conflict graph G_3 is 9-colorable.*

Proof. This proof follows a similar inductive argument on the size of G_3 with an improvement on the proof of Lemma 14, as the definition of conflict in G_3 is relaxed.

The base case is trivially true, and for the inductive step, we again pick the smallest non-empty cell C_1, such that $\mathsf{Squares}_{\mathcal{FC}}(C_1) \neq \emptyset$ and for any cell C_2 contained in C_1, $\mathsf{Squares}_{\mathcal{FC}}(C_2) = \emptyset$. Let i be the level of C_1. By the induction hypothesis, $G_2 \setminus \mathsf{Squares}_{\mathcal{FC}}(C_1)$ is 11-colorable, and we show that we can color the squares of $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ while maintaining a valid 11-coloring of G_2. By induction hypothesis, $G_3 \setminus \mathsf{Squares}_{\mathcal{FC}}(C_1)$ is 9-colorable, and we show that we can color the squares of $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ while maintaining this property. The proof of Lemma 14, implies that $|\mathsf{Squares}_{\mathcal{FC}}(C_1)| \leq 5$. However, using Definition 17, there is at most one square on each side of the vertical and horizontal level-$(i+1)$ line cutting

through the center of C_1. Hence, it is always possible to pair up the twin squares by giving them the same color, such that we can color the graph $G_3[\mathsf{Squares}_{\mathcal{FC}}(C_1)]$ using at most three colors as shown in Figure 15.

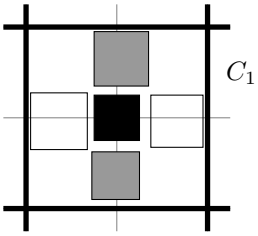


Figure 15 A 3-coloring for $G_3[\mathsf{Squares}_{\mathcal{FC}}(C_1)]$ using the same color for twin squares.

Using the bound of six (from proof of Lemma 8) on the number of squares in $G_3 \setminus \mathsf{Squares}_{\mathcal{FC}}(C_1)$ conflicting with any square of $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ leaves three out of nine used to color $G_3 \setminus \mathsf{Squares}_{\mathcal{FC}}(C_1)$ to color $\mathsf{Squares}_{\mathcal{FC}}(C_1)$. Eventually, we have a 9-coloring for G_3. ◀

▶ **Lemma 25.** *The graph G_5 is 16-colorable.*

Proof. We show a 16-coloring of G_5 by induction on the size of the graph with the property that for any cell $C \in \mathcal{L}_3$, all the squares in $\mathsf{Squares}_{\mathcal{FC}}(C)$ are colored with at most four colors.

The base case is trivial. Let us consider the smallest cell C_1 so that $\mathsf{Squares}_{\mathcal{FC}}(C_1) \neq \emptyset$, and for any cell C_2 contained in C_1 we have $\mathsf{Squares}_{\mathcal{FC}}(C_2) = \emptyset$. By induction hypothesis, $G \setminus \mathsf{Squares}_{\mathcal{FC}}(C_1)$ is 16-colorable. We show that we can color the squares of $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ while maintaining the mentioned property. There are two possible cases:

Case 1. Consider C_1 to be a level-i square-cell. Like Lemma 22, we start by counting the number of colors needed for the squares inside C_1, which are precisely $\mathsf{Squares}_{\mathcal{FC}}(C_1)$. According to Fact 24, at most seven such squares can intersect the level-$(i+1)$ vertical grid line cutting through C_1. Since there can be at most three squares above and below the level-$(i+1)$ horizontal grid line, we can pair up the twin squares by giving them the same color, such that we can color the graph $G[\mathsf{Squares}_{\mathcal{FC}}(C_1)]$ with at most four colors (see Figure 13a for illustration). Now, we consider the squares not in $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ that conflict with any square in $\mathsf{Squares}_{\mathcal{FC}}(C_1)$. Any such square must intersect the boundary of the cell C_1 (see Figure 16).

- There can be at most four squares that contain a corner of C_1.
- We now look at the squares that do not contain any corner of C_1. For the two level-$(i-1)$ grid lines, there can be only one square intersecting each line without intersecting the corner of the cell: Squares crossing a grid line of level $(i-1)$ must be of level-j for some $j \leq i-2$, and according to Fact 4, their size is strictly bigger than half of the size of a level-i cell (specifically, C_1) with $r = 4$. Thus, at most, two additional conflicting squares can intersect the level-$(i-1)$ grid lines.
- Let us consider the two other sides of C_1 formed by the level-i grid lines. There are at most three squares intersecting those sides but not the corners, since the size of those squares is strictly more than $\frac{1}{4}$ of the size of C_1. In total, there are at most six additional conflicting squares intersecting the level-i grid lines.

We have shown that the squares in $G_5 \setminus \mathsf{Squares}_{\mathcal{FC}}(C_1)$ conflicting with $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ can be colored by at most twelve colors. Considering the additional four colors needed to color $\mathsf{Squares}_{\mathcal{FC}}(C_1)$, the graph is 16-colorable.

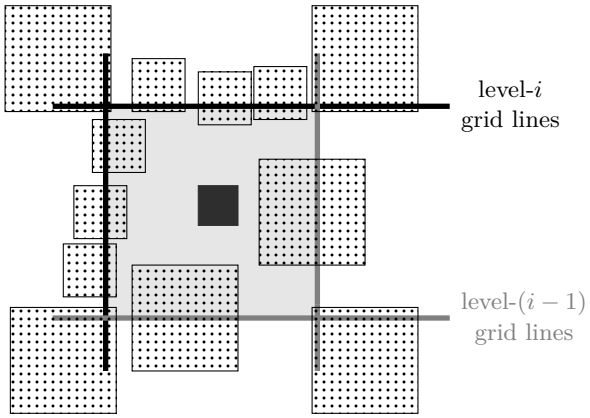


Figure 16 The black square is a square of $\mathsf{Squares}_{\mathcal{FC}}(C_1)$, where C_1 (shaded) is a level-i square-cell. The dot-filled squares are the squares of $G \setminus \mathsf{Squares}_{\mathcal{FC}}(C_1)$ that can conflict with any square in $\mathsf{Squares}_{\mathcal{FC}}(C_1)$.

Case 2. Consider C_1 to be a level-$(i, i+1)$ half-cell. The squares contained by C_1 are precisely the squares in $\mathsf{Squares}_{\mathcal{FC}}(C_1)$. According to Fact 24, at most three squares can intersect the level-$(i+1)$ horizontal grid line cutting through C_1, and they can be colored using three colors as shown in Figure 13b. Now, we bound the number of colors needed to color the squares not in $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ conflicting with any square in $\mathsf{Squares}_{\mathcal{FC}}(C_1)$. Note that those squares must intersect the boundary of C_1 (see Figure 16).

- There are at most four squares covering a corner of C_1.
- Now, consider the squares not containing any of the corners of C_1. Let C_0 be the smallest level-i square-cell containing C_1. Up to three squares can intersect the vertical level-i grid line that C_1 shares with C_0, without intersecting a corner of C_1 (Figure 17). Since the width of C_1 is half that of its height, there can be at most one square intersecting the level-i grid lines at the top and the bottom of C_1 without intersecting a corner. Hence, at most, two squares intersect the top or bottom grid lines of C_1 without intersecting its corners.

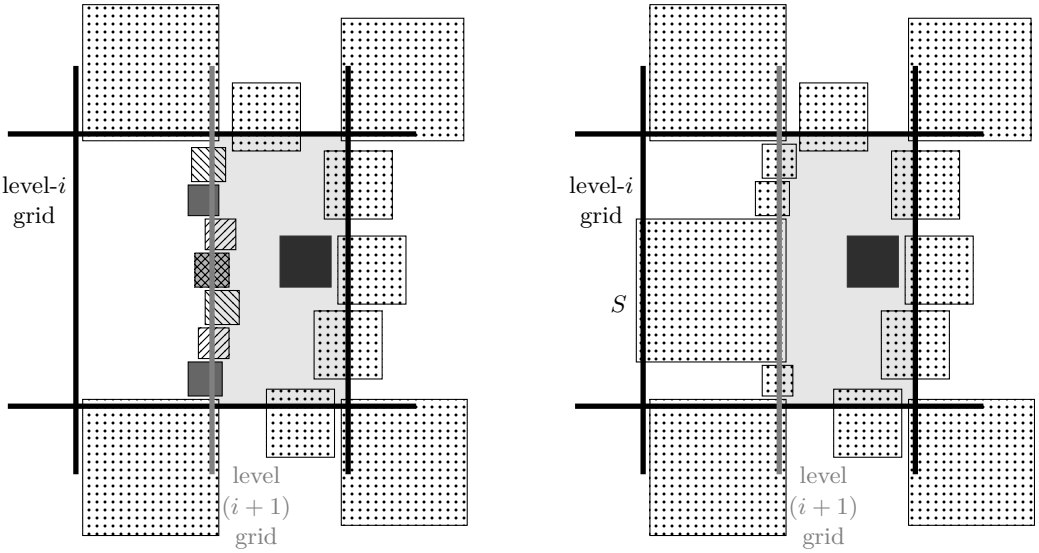
  Next, we examine the fourth side of C_1, which is formed by a level-$(i+1)$ grid line (the gray lines on Figure 17). We consider two possible scenarios depending on whether all conflicting squares intersecting the level-$(i+1)$ grid line have C_0 as their fitting cell.

(a) If all these squares belong to $\mathsf{Squares}_{\mathcal{FC}}(C_0)$, then by the induction hypothesis, they are colored with at most four colors (see Figure 17a).

(b) Otherwise, there exists a square S that also intersects a vertical side of C_0 (and not a corner of C_1) such that C_0 is not its fitting cell (see Figure 17b). Then, the size of S is bigger than half of $\mathsf{height}(C_1)$. Moreover, the size of the other squares crossing this side of C_1 (a level-$(i+1)$ grid line) is strictly more than $\frac{1}{8}$ of $\mathsf{height}(C_1)$. Thus, at most three more squares can fit into this space.

In both scenarios, the squares intersecting the fourth side of C_1 can be colored with at most four colors.

We showed that all the squares in $G_5 \setminus \mathsf{Squares}_{\mathcal{FC}}(C_1)$ conflicting with any square in $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ can be colored using at most thirteen colors. Since we need at most three more colors for the squares in $\mathsf{Squares}_{\mathcal{FC}}(C_1)$ (Figure 13b), then G_5 is 16-colorable in this case as well. ◀



(a) All the squares intersecting the gray level-$(i+1)$ grid line and not a corner are in $\mathsf{Squares}_{\mathcal{GC}}(C_0)$.

(b) The square S, intersecting the level-$(i+1)$ grid line and not a corner, is not in $\mathsf{Squares}_{\mathcal{GC}}(C_0)$.

Figure 17 The black square is a square in $\mathsf{Squares}_{\mathcal{GC}}(C_1)$, and C_1 is a level-$(i, i+1)$ cell (shaded). The cell formed by the bold black lines is C_0. The remaining squares are the squares in $G_5 \setminus \mathsf{Squares}_{\mathcal{GC}}(C_1)$ that conflict with any element of $\mathsf{Squares}_{\mathcal{GC}}(C_1)$. They can all have different colors in G_5.

Dynamic Streaming Algorithms for Geometric Independent Set

Timothy M. Chan
Siebel School of Computing and Data Science, University of Illinois Urbana-Champaign, IL, USA

Yuancheng Yu
Siebel School of Computing and Data Science, University of Illinois Urbana-Champaign, IL, USA

Abstract

We present the first space-efficient, fully dynamic streaming algorithm for computing a constant-factor approximation of the maximum independent set size of n axis-aligned rectangles in two dimensions. For an arbitrarily small constant $\delta > 0$, our algorithm obtains an $O((1/\delta)^2)$ approximation and requires $O(U^\delta \operatorname{polylog} n)$ space and update time with high probability, assuming that coordinates are integers bounded by U. We also obtain a similar result for fat objects in any constant dimension. This extends recent non-streaming algorithms by Bhore and Chan from SODA'25, and also greatly extends previous streaming results, which were limited to special types of geometric objects such as one-dimensional intervals and unit disks.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Geometric Independent Set, Dynamic Streaming Algorithms

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.17

1 Introduction

In this paper, we study dynamic streaming algorithms for a fundamental geometric optimization problem: *maximum independent set* (MIS) of geometric intersection graphs. Given a set S of n objects in d dimensions, the problem is to compute a maximum-cardinality subset $I \subseteq S$ of *independent* (i.e., pairwise-disjoint) objects. Exact geometric MIS is NP-hard even for unit squares[1] in 2D, and there has been a rich body of work on approximation algorithms [33, 25, 26, 16, 19, 27, 22]. The dynamic version of the problem (maintaining an approximate solution under insertions and deletions of objects in S) has also gained much attention in the last few years [32, 6, 10, 13, 8].

In particular, MIS for rectangles in 2D has been extensively studied. Early algorithms, based on simple divide-and-conquer, achieved logarithmic approximation ratios [2, 37, 41, 17, 5]. For $(1+\varepsilon)$-approximation, Adamaszek, Har-Peled, and Weise [1] obtained a quasi-PTAS with running time $n^{\operatorname{polylog} n}$, which was subsequently improved to $n^{\operatorname{polyloglog} n}$ by Chuzhoy and Ene [21]. For polynomial-time results, Chalermsook and Chuzhoy [14] obtained an $O(\log\log n)$-approximation algorithm via LP rounding, and in a major breakthrough, Mitchell [39] obtained the first $O(1)$-approximation algorithm, running in $O(n^{21})$ time, via dynamic programming (DP). The approximation ratio was subsequently improved to $2+\varepsilon$ for any constant $\varepsilon > 0$ by Gálvez, Khan, Mari, Mömke, Pittu, and Wiese [31, 30]. Recently, the running time has also been improved to $O(n^{1+\delta})$ for any constant $\delta > 0$ by Bhore and Chan [8], with approximation ratio dependent on $1/\delta$ (polynomially); if only an approximate size is desired, the time bound can be further improved to $O(n \operatorname{polylog} n)$. With the static time bound reduced to near linear, one can next turn to the question of efficient dynamic

[1] Throughout this paper, all squares and rectangles are axis-aligned by default.

© Timothy M. Chan and Yuancheng Yu;
licensed under Creative Commons License CC-BY 4.0

algorithms. Indeed, Bhore and Chan showed that their static algorithm can be dynamized to obtain the first fully dynamic, constant-approximation algorithm, with $O(n^\delta)$ amortized update time.

For fat objects in any constant dimension d, PTASes were known [33, 16, 25], but the running time is high ($n^{\Omega(1/\varepsilon)}$). For faster algorithms, a simple greedy strategy gives an $O(1)$-approximation in quadratic time [25], and Bhore and Chan [7] recently obtained an improved $O(n \log n)$-time, $O(1)$-approximation algorithm. Bhore and Chan [8] also obtained fully dynamic $O(1)$-approximation algorithms with $O(n^\delta)$ amortized update time. For disks in $\mathbb{R}^2$ specifically, polylogarithmic amortized update time is also known [6, 7].

Meanwhile, *streaming* algorithms for geometric optimization problems also have a rich history. The dynamic streaming model, allowing streams of updates (both insertions and deletions), is also known as the *turnstile* model (viewing insertions/deletions as incrementing/decrementing the multiplicity of an element – we assume that the multiplicity of each element is nonnegative). We seek algorithms that use one pass and small (sublinear) space. Typically, one needs to assume that input points lie in a bounded integer universe[2] $[U]^d$, and allow (Monte Carlo) randomization. For the streaming version of MIS, it is more natural to consider outputting an approximate size rather than the solution itself (since the MIS itself may have linear size).

Indyk [35] initiated the study of dynamic streaming algorithms for geometric problems, and considered basic problems such as Euclidean minimum spanning tree (EMST), min-weight matching, facility location, and k-median. Follow-up works investigated k-clustering [11, 29, 34], facility location [24], maximum cut [29, 38, 20], EMST [28], geometric Steiner forest [23], width and ε-kernels [3, 18], etc.

Streaming algorithms for geometric MIS, however, have received relatively limited success. Cabello and Pérez-Lantero [12] obtained tight approximation ratio for MIS of 1D intervals in insertion-only streams. Bakshi, Chepurko, and Woodruff [4] extended their result to fully dynamic (turnstile) streams, and also gave a dynamic streaming algorithm with polylogarithmic space for MIS of weighted unit disks in 2D. Bhore, Klute, and Oostveen [9] gave a 4-approximate insertion-only streaming algorithm for axis-aligned unit-height rectangles using $O(\mathrm{OPT} \operatorname{polylog} n)$ space, where OPT denotes the size of the MIS (which could be large). They also showed that for line segments, constant-approximation algorithms require linear space even for insertion-only streams. The above lines of work raise the following natural question:

"Do there exist fully dynamic streaming $O(1)$-approximation algorithms with sublinear space and update time for MIS for rectangles, or other types of geometric objects (more general than 1D intervals, 2D unit-height rectangles, and 2D unit disks)?"

New results. We answer this question in the affirmative by presenting the first space-efficient, fully dynamic streaming $O(1)$-approximation algorithm for MIS for rectangles in the plane. Specifically, our algorithm computes an $O((1/\delta)^2)$-approximation of the optimal size with $O(U^\delta \operatorname{polylog} n)$ space and update time w.h.p.,[3] where coordinates are in $[U]^2$. In particular, for polynomially bounded universe size $U = n^{O(1)}$, the space and update time is $O(n^\delta)$ (after adjusting δ by a constant factor).

[2] $[U]$ denotes $\{0, 1, \ldots, U-1\}$.

[3] With high probability, i.e., probability $1 - O(1/n^c)$ for an arbitrarily large constant c.

We also obtain a similar result for (unweighted and weighted) fat objects in any constant dimension d; in particular, this is new even for arbitrary-radii disks in the plane. Here, our algorithm computes an $O((1/\delta)^2)$-approximation of the optimal size with $O(U^\delta \operatorname{polylog} n)$ space and update time w.h.p., assuming that the objects are contained in $[0, U)^d$ and have diameter $\Omega(1)$ (and weights are in $[U]$).

Techniques. Our algorithms are based on Bhore and Chan's recent dynamic algorithms in the non-streaming setting [8]. Although most dynamic data structures do not adapt well to the streaming model and inherently require linear space or worse, we show that Bhore and Chan's approach does, luckily! Their approach for MIS for rectangles can be viewed as a combination of the old divide-and-conquer method [2] with Mitchell's constant-approximation DP-based algorithm [39]. More precisely, they used a b-way divide-and-conquer to reduce the problem to special subproblems in which the input rectangles are stabbable by $O(b)$ horizontal and vertical lines. The approximation ratio increases by an $O((\log_b n)^2)$ factor, which is constant if we choose $b = n^\delta$. They observed that for these special subproblems, one can round the input to reduce the input size to $b^{O(1)}$, while increasing the approximation ratio by at most a constant; when the input size is this small, one can afford to run Mitchell's polynomial-time approximation algorithm as a black box.

It turns out that the rounding trick is a perfect fit for dynamic streaming. The divide-and-conquer part, however, requires rethinking. We provide a new interpretation of the divide-and-conquer, by directly assigning input rectangles to levels of the recursion (this is possible due to the bounded integer universe assumption, with U^δ factors instead of n^δ); this simpler interpretation actually helps understand Bhore and Chan's algorithm better. To achieve sublinear space, we cannot afford to solve all subproblems across a level. Rather, we estimate the answer by taking a small random sample of the subproblems (the idea of using random sampling to approximate a sum appeared already in one version of Bhore and Chan's *static* algorithm [8], but not dynamic). In the dynamic setting, the right sampling rate is not known ahead of time. Our idea is to try all sampling rates (powers of 2) simultaneously. However, for sampling rates that are too large, the space usage would be too large. We resolve this issue by hashing to a small universe. This ensures that space is kept small for all sampling rates, and at the same time, for the right sampling rate, collision is likely avoided. We remark that sampling and hashing are both commonly used in the streaming literature, but we will keep our presentation largely self-contained, to make it accessible to computational geometers without prior background on streaming.

MIS for fat objects can be solved in a similar way, using shifted quadtrees for the divide-and-conquer part.

2 Preliminaries on Streaming

We begin by describing a key tool on dynamic streaming that we need. We formulate it in a general form, as stated in Theorem 1 below (even though the ideas in the proof may not be new to experts on streaming, our formulation will make applications easier).

The setup is roughly the following: we have a collection of subproblems, where each input object is assigned to one subproblem (a substream). Each subproblem individually admits an efficient dynamic streaming algorithm. We want to approximate the sum of the answers to these subproblems, subject to a stream of updates (insertions and deletions of objects), assuming that answers to nonempty subproblems lie in a bounded range $[1, B]$. Obtaining sublinear space is a challenge, because we cannot afford to explicitly store the answers to all subproblems.

▶ **Theorem 1.** *Let f be any function that maps sets to $\{0\} \cup [1, B]$ and satisfies the following two properties:*

- *$f(S) = 0$ iff $S = \emptyset$;*
- *there is a dynamic streaming algorithm for maintaining $f(S)$ under insertions and deletions of objects in S with $s(n)$ space and $t(n)$ time w.h.p., where $n = |S|$.*

Then we can design a dynamic streaming algorithm for maintaining a $(1\pm\varepsilon)$-approximation of $T = \sum_{i=1}^{M} f(S_i)$, for a collection of sets $S_1, \ldots, S_M$, with $O(s(n)(B/\varepsilon)^{O(1)} \log^{O(1)}(nM))$ space and $O(t(n)(B/\varepsilon)^{O(1)} \log^{O(1)}(nM))$ time per update (insertion/deletion of an object to S_i for a given i), where $n = \sum_{i\in[M]} |S_i|$, w.h.p.

Before proving the theorem, we first consider a special case stated in the lemma below: when the number of nonempty subproblems is small, we can actually maintain the sum exactly. The precise statement is a little subtle, since the number of nonempty subproblems may vary over time as objects are inserted and deleted: at times when the number is larger than the threshold τ, the maintained answer may not be correct, but when the number gets back to below τ, the maintained answer would become correct again. The lemma is proved by a simple bucketing approach using pairwise-independent hash functions, with $O(\tau^2)$ buckets.

▶ **Lemma 2.** *Let f be any function that maps sets to numbers and satisfies the following two properties:*

- *$f(S) = 0$ iff $S = \emptyset$;*
- *there is a dynamic streaming algorithm $\mathcal{A}$ for maintaining $f(S)$ under insertions and deletions of objects in S with $s(n)$ space and $t(n)$ time w.h.p., where $n = |S|$.*

Let τ be a parameter. Then we can design a dynamic streaming algorithm for maintaining $\widetilde{T} = \sum_{i=1}^{M} f(S_i)$ for a collection of sets $S_1, \ldots, S_M$, with $O(s(n)\tau^2 \log n)$ space and $O(t(n)\tau^2 \log n)$ time per update (insertion/deletion of an object to S_i for a given i), where $n = \sum_{i=1}^{M} |S_i|$. The algorithm is correct w.h.p. when $|\{i \in [M] : S_i \neq \emptyset\}| < \tau$. Furthermore, this condition can be tested w.h.p. with the same space and time.

Proof. Let $h : [M] \to [q]$ be a random function chosen from a pairwise independent hash family, for $q = 4\tau^2$; by standard constructions (e.g. [40, Theorem 8.16]), h can be represented using $O(\log m)$ space.

Our streaming algorithm is simple: for each $a \in [q]$, we use the given algorithm $\mathcal{A}$ to maintain $f(Y_a)$ for the set

$$Y_a \;=\; \bigcup_{i\in[M]:\ h(i)=a} S_i,$$

i.e., when we insert/delete an object in S_j, we compute $a = h(j)$ and then insert/delete that object in Y_a using algorithm $\mathcal{A}$. Finally, let

$$y := |\{a \in [q] : Y_a \neq \emptyset\}|,$$

and when $y < \tau$, we output

$$\widehat{T} \;=\; \sum_{a\in[q]} f(Y_a).$$

For correctness, we discuss the two cases. Suppose $|\{i \in [M] : S_i \neq \emptyset\}| < \tau$. We claim that $\widehat{T} = \widetilde{T}$ (and $y < \tau$ trivially) with probability at least $3/4$. To see this, observe that $\widehat{T} = \widetilde{T}$ when there are no *colliding pairs*, i.e., no pairs (i, j) with $i \neq j$, $S_i, S_j \neq \emptyset$, and $h(i) = h(j)$. The expected number of colliding pairs is at most $|\{i \in [M] : S_i \neq \emptyset\}|^2 \cdot 1/q < 1/4$. By Markov's inequality, the probability that there is a colliding pair is less than $1/4$.

Conversely, suppose $|\{i \in [M] : S_i \neq \emptyset\}| \geq \tau$. We claim that $y \geq \tau$ with probability at least 3/4. To see this, pick τ nonempty subsets S_i. By the same argument as above, with probability at least 3/4, there are no colliding pairs among these τ nonempty subsets S_i, implying $y \geq \tau$.

The error probability is bounded by a constant, but can be lowered by running logarithmically many independent copies of the algorithm and taking the majority of the answers. ◀

Note that alternatively, the above condition $|\{i \in [M] : S_i \neq \emptyset\}| < \tau$ can be tested by directly applying known techniques for sampling elements (an "ℓ_0-sampler") in a turnstile stream [28, 36] (viewing an insertion/deletion in S_i as incrementing/decrementing the multiplicity of $i \in [M]$). The proof above may be viewed as an adaptation or extension of such techniques.

We now reduce the general case to the special case by random sampling, using logarithmically many random subsets with different sampling rates. Chebyshev's inequality shows that the sum of a random subset approximates the overall sum, assuming pairwise independence.

▶ **Lemma 3.** *Let $T = \sum_{i=1}^m a_i$, where $a_i \in [1, B]$. Let R be a random subset of $[m]$ such that for each $i \in [m]$, $\Pr[i \in R] = \rho$, and the events "$i \in R$" over all $i \in [m]$ are pairwise independent. For $\varepsilon < 1$, if $\rho m \geq cB/\varepsilon^2$, then $(1/\rho)\sum_{i \in R} a_i$ gives a $(1 \pm \varepsilon)$-approximation of T with probability $1 - O(1/c)$.*

Proof. Let $X = \sum_{i \in R} a_i = \sum_{i=1}^m a_i \cdot [i \in R]$, where $[\cdot]$ denotes the Iverson bracket. Then $\mathbb{E}[X] = \rho T$, and by pairwise independence, $\mathrm{Var}[X] = \sum_{i=1}^m a_i^2 \rho \leq B \sum_{i=1}^m a_i \rho = B\rho T$. By Chebyshev's inequality, $\Pr[|X - \mathbb{E}[X]| \geq \varepsilon \mathbb{E}[X]] \leq \mathrm{Var}[X]/(\varepsilon^2 \mathbb{E}[X]^2) \leq B/(\varepsilon^2 \rho T) \leq B/(\varepsilon^2 \rho m) \leq 1/c$. ◀

Proof of Theorem 1. For each $\rho = 1/2, 1/4, \ldots, 1/2^{\lceil \log M \rceil}$, let R_ρ be a random subset of $[M]$ with sampling rate ρ, where the events "$i \in R_\rho$" are pairwise independent. By standard constructions (e.g. [40, Theorem 8.16]), such a subset can be represented using $O(\log m)$ space (so that we can tell if a given index i is in R_ρ in $O(1)$ time).

We will apply the algorithm in Lemma 2 to try to maintain

$$\widetilde{T}_\rho \;=\; \sum_{i \in R_\rho} f(S_i)\,,$$

with $\tau = cB/\varepsilon^2$ for a sufficiently large constant c. Pick the largest ρ^* for which $|\{i \in R_{\rho^*} : S_i \neq \emptyset\}| < \tau$ (recall that this condition can be tested w.h.p. by Lemma 2). Then the algorithm from Lemma 2 will succeed in computing $\widetilde{T}_{\rho^*}$ w.h.p. and we return $(1/\rho^*)\widetilde{T}_{\rho^*}$.

For the analysis, let $m = |\{i \in [M] : S_i \neq \emptyset\}|$. Let ρ_0 be a power of 2 with $\tau/4 < \rho_0 m \leq \tau/2$. Then by Chebyshev's inequality,

$$\Pr[\rho^* < \rho_0] \leq \Pr[|\{i \in R_{\rho_0} : S_i \neq \emptyset\}| \geq \tau] \leq \rho_0 m/(\tau - \rho_0 m)^2 \leq O(1/(\rho_0 m)).$$

On the other hand, for each $\rho \geq 8\rho_0$, by Chebyshev's inequality,

$$\Pr[\rho^* = \rho] \leq \Pr[|\{i \in R_\rho : S_i \neq \emptyset\}| < \tau] \leq \rho m/(\rho m - \tau)^2 \leq O(1/(\rho m)).$$

By a geometric series over all $\rho \geq 8\rho_0$, $\Pr[\rho^* \geq 8\rho_0] \leq O(1/(\rho_0 m))$. Thus, with probability $1 - O(1/c)$, ρ^* must be a value among $\{\rho_0, 2\rho_0, 4\rho_0\}$. For each such value ρ, the probability that $\rho^* = \rho$ but $(1/\rho)\widetilde{T}_\rho$ is not a $(1 \pm \varepsilon)$-approximation of T is at most $O(1/c)$, by applying Lemma 3 to the sum $T = \sum_{i \in [M] : S_i \neq \emptyset} f(S_i)$, which has m nonempty terms.

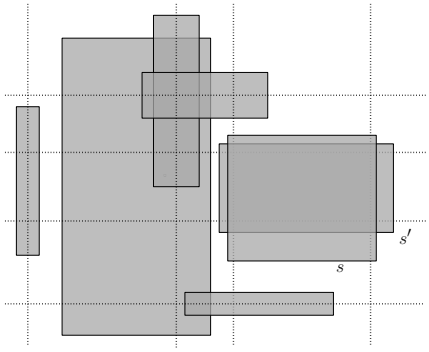


Figure 1 Proof of Lemma 4. Each rectangle is stabbed by at least one horizontal line and at least one vertical line of Γ (shown as dotted lines). The rectangles s and s' belong to the same class.

The error probability is bounded by a constant, but can again be lowered by running logarithmically many independent copies of the algorithm and taking the majority of the answers. (For simplicity, we have not attempted to optimize the $(B/\varepsilon)^{O(1)} \log^{O(1)}(nM)$ factors.) ◀

3 Maximum Independent Set for Rectangles

We now present our dynamic streaming algorithm for MIS for rectangles. We begin by considering a special case in which the input rectangles are stabbable by a small number of horizontal as well as vertical lines. Bhore and Chan [8] solved this case by a simple rounding trick, which we observe works well in the dynamic streaming setting:

▶ **Lemma 4.** *Let Γ be a set of $O(b)$ horizontal/vertical lines in 2D. Let S be a set of n axis-aligned rectangles in 2D with the property that each rectangle in S is stabbed by at least one horizontal line and at least one vertical line in Γ. Then we can design a dynamic streaming algorithm for maintaining an $O(1)$-approximation of the maximum independent set for S with $O(b^{O(1)} \log^{O(1)} n)$ space and time w.h.p.*

Proof. The lines in Γ form a (non-uniform) grid with $O(b^2)$ grid cells. (See Figure 1.) Place two rectangles of S in the same *class* if they intersect the same subset of grid cells. There are $O(b^4)$ classes. Let $\hat{S}$ be a subset of S where we keep one "representative" element from each class. Then $|\hat{S}| = O(b^4)$. We apply Mitchell's algorithm [39] (or its subsequent improvement [31, 30]) to compute an $O(1)$-approximation of the maximum independent set for the rectangles in $\hat{S}$. This takes time polynomial in $|\hat{S}|$, i.e., $b^{O(1)}$ time (note that the optimal size must be at most b^2 under the input assumption). Bhore and Chan [8] proved that this is an $O(1)$-approximation of the maximum independent set for S.

To implement the algorithm in the dynamic streaming model, it suffices to maintain one representative element per class, since we can re-run Mitchell's algorithm from scratch. Frahling, Indyk, and Sohler [28] (see also [36]) showed how to maintain a random element of a set in the dynamic streaming model; we can just apply their algorithm to each class and use the random element as the representative. ◀

We now solve the main problem by reinterpreting the divide-and-conquer approach by Bhore and Chan [8] and incorporating Theorem 1.

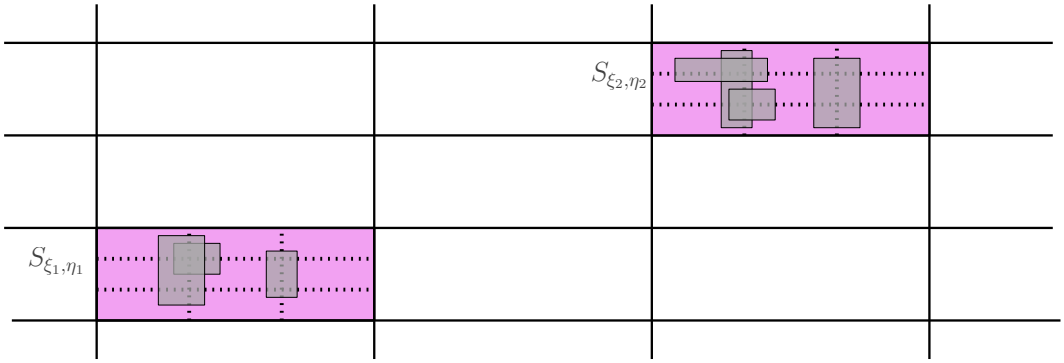


Figure 2 Proof of Theorem 5. The picture shows two different sets S_{ξ_1,η_1} and S_{ξ_2,η_2} from a common collection $S_{k,\ell}$.

▶ **Theorem 5.** *Given a set S of n axis-aligned rectangles in 2D with coordinates in $[U]$ as a dynamic geometric stream, we can compute an $O((1/\delta)^2)$-approximation of the maximum independent set size for S using $O(U^{O(\delta)} \log^{O(1)} n)$ space and update time w.h.p.*

Proof. Write each coordinate value in base b; the number of digits in each value is $\lceil \log_b U \rceil$.

Consider a rectangle $s = [x, x'] \times [y, y']$ in S. Suppose that the most significant digit which x and x' differ in is the k-th most significant digit, and the most significant digit which y and y' differ in is the ℓ-th most significant digit. Let ξ be the $k-1$ most significant digits of x and x'. Let η be the $\ell-1$ most significant digits of y and y'. We place s in a set $S_{\xi,\eta}$; in turn, we place the set $S_{\xi,\eta}$ in a collection $\mathcal{S}_{k,\ell}$.[4] (See Figure 2.)

For each $k, \ell \in [\lceil \log_b U \rceil]$, we maintain an $O(1)$-approximation of

$$T_{k,\ell} = \sum_{S_{\xi,\eta} \in \mathcal{S}_{k,\ell}} f_{k,\ell}(S_{\xi,\eta})$$

using Theorem 1. Here, $f_{k,\ell}(S)$ is defined as the maximum independent set size for $S^{\#} = \{s^{\#} : s \in S\}$, where $s^{\#}$ denotes the rectangle obtained from $s = [x, x'] \times [y, y']$ by removing the $k-1$ most significant digits from x and x' and the $\ell-1$ most significant digits from y and y', assuming that x and x' differ in the k-th most significant digit and y and y' differ in the ℓ-th most significant digit (if the assumption is not satisfied, make $s^{\#}$ undefined). Note that for an individual set $S_{\xi,\eta} \in \mathcal{S}_{k,\ell}$, $f(S_{\xi,\eta})$ is the same as the maximum independent set size for $S_{\xi,\eta}$ (although this may not be true for a general set S). Also note that every rectangle $s^{\#}$ is stabbed by one of the b vertical lines at $x = b^{\lceil \log_b U \rceil - k} i$ for some $i \in [b]$, and one of the b horizontal lines at $y = b^{\lceil \log_b U \rceil - \ell} j$ for some $j \in [b]$, and so we can maintain $f_{k,\ell}(S)$ using Lemma 4 (note that $f_{k,\ell}(S) \le b^2$). The conditions of Theorem 1 are fulfilled with $B = b^2$ and $s(n), t(n) = O(b^{O(1)} \log^{O(1)} n)$. We return the maximum of the approximations of $T_{k,\ell}$ over all $k, \ell \in [\lceil \log_b U \rceil]$.

Observe that $T_{k,\ell}$ is just the maximum independent set size for $\bigcup_{S_{\xi,\eta} \in \mathcal{S}_{k,\ell}} S_{\xi,\eta}$, because rectangles in S_{ξ_1,η_1} do not overlap with rectangles in S_{ξ_2,η_2} for any two different sets $S_{\xi_1,\eta_1}, S_{\xi_2,\eta_2} \in \mathcal{S}_{k,\ell}$. Thus, OPT (the maximum independent set size for S) is at most $\sum_{k,\ell} T_{k,\ell} \le O(\log_b U)^2 \cdot \max_{k,\ell} T_{k,\ell}$. It follows that the returned value is an $O((\log_b U)^2)$-approximation of OPT. Finally, we set $b = U^{\delta}$. ◀

[4] Roughly, Bhore and Chan's recursion [8] generates a primary tree (essentially a b-ary interval tree in x), where each node stores a secondary tree (a b-ary interval tree in y). Each set $S_{\xi,\eta}$ corresponds to a node of a secondary tree. Each collection $\mathcal{S}_{k,\ell}$ corresponds to a "level" of the whole structure (namely, the ℓ-th level of all nodes at the k-th level of the primary tree).

Although the preceding proof is compact, the details are subtle. For example, some readers may wonder why we didn't define $f_{k,\ell}(S)$ more simply as the maximum independent set size for S, bypassing the definition of $s^{\#}$. The reason is that Theorem 1 requires working with $f_{k,\ell}(S)$ for general sets S, not just the $S_{\xi,\eta}$'s: indeed, the proof of Lemma 2 maintains intermediate sets that are *unions* of $S_{\xi,\eta}$'s (from a common bucket); without replacing s with $s^{\#}$, the rectangles in such composite sets may not be stabbable by b vertical/horizontal lines. Admittedly, defining $f_{k,\ell}(S)$ as the maximum independent set for $S^{\#}$ is somewhat counterintuitive, as its value seems irrelevant or meaningless when S is a union of two or more sets $S_{\xi,\eta}$'s. (To picture such a set $S^{\#}$, imagine overlaying the two violet cells in Figure 2 on top of each other.) However, under the "right" circumstances, when we don't have collisions, as in the proof of Lemma 2, S would revert to a single $S_{\xi,\eta}$, and $f_{k,\ell}(S)$ would become meaningful again.

▶ Remark 6. Curiously, the approach of Theorem 5 extends to yield a space-efficient, $O(1)$-approximation streaming algorithm of boxes (hyperrectangles) in any constant dimension beyond 2. The catch is that the update time is prohibitively large ($2^{U^{O(\delta)}}$), as Mitchell's algorithm works only in 2D. (Obtaining a polynomial-time $O(1)$-approximation algorithm for boxes in 3D is a major open problem even in the non-streaming setting.)

As in Bhore and Chan's paper [8], our approach can also solve the *minimum piercing set* problem for 2D rectangles in the dynamic streaming model, but the approximation ratio increases to $O(\log\log n)$ with $O(U^{O(1)}\log n)$ space and update time.

4 Maximum Independent Set for Fat Objects

We now consider the case of fat objects, again adapting Bhore and Chan's non-streaming algorithm [8]. Our algorithm extends to the weighted version of independent set.

In the following, we adapt notation and definitions from [8]. Let $\mathrm{diam}(s)$ denote the diameter of an object s, where diameter refers to L_∞-diameter. A collection $\mathcal{C}$ of objects is *c-fat* if for every hypercube γ, there exist c points piercing all objects in $\mathcal{C}$ that intersect γ and have diameter at least $\mathrm{diam}(\gamma)$. A *quadtree box* γ is a hypercube of the form $[i_1 2^\ell, (i_1+1)2^\ell) \times \cdots \times [i_d 2^\ell, (i_d+1)2^\ell)$ for integers $i_1, \ldots, i_d, \ell$.

An object s is *c_0-good* if it is contained in a quadtree box with diameter at most $c_0\mathrm{diam}(s)$.

▶ **Fact 7** (Shifting Lemma [15, 16]). *Suppose d is even. Let $v_j = (\frac{jU}{d+1}, \ldots, \frac{jU}{d+1}) \in \mathbb{R}^d$. For every object $s \subset [0,U)^d$, there exists $j \in \{0,\ldots,d\}$ such that $s + v_j$ is $O(d)$-good.*

Bhore and Chan [8] solved the following special case by rounding, which we observe works well in the dynamic streaming setting:

▶ **Lemma 8.** *Let d, c, c_0 be constants. Let Γ be b disjoint quadtree boxes. Let S be a set of n c_0-good objects in $\mathbb{R}^d$ of constant description complexity from a c-fat collection $\mathcal{C}$, with the property that each object in S intersects the boundary of at least one quadtree box of Γ. Each object is given a weight in $[U]$. Then we can design a dynamic streaming algorithm for maintaining an $O(1)$-approximation of the maximum-weight independent set for S with $O(b^{O(1)}\log^{O(1)}(nU))$ space and time w.h.p.*

Proof. First round the weights to powers of 2; the approximation ratio increases by at most a factor of 2.

Place two objects of S in the same *class* if they intersect the same subset of cells in Γ. There are at most $b^{O(1)}$ classes (as noted in [8]). Let $\hat{S}$ be a subset of S where we keep one "representative" element from each class, namely, a largest-weight element from the class.

Then $|\hat{S}| \leq b^{O(1)}$. We apply a known algorithm (e.g., [16]) to compute an $O(1)$-approximation to the maximum-weight independent set for the fat objects in $\hat{S}$. This takes time polynomial in $|\hat{S}|$, i.e., $b^{O(1)}$ time. Bhore and Chan [8] proved that this is an $O(1)$-approximation of the maximum independent set for S (using the goodness assumption).

To implement the algorithm in the dynamic streaming model, it suffices to maintain a representative element per class. Frahling, Indyk, and Sohler [28] (see also [36]) showed how to maintain a random element of a set in the dynamic streaming model; we can just apply their algorithm to each class restricted to elements of each weight, and use the random element as the representative for the largest weight whose set is nonempty. There are only $O(\log U)$ possible weights. ◀

We now solve the main problem, via quadtree-based divide-and-conquer, as in [8],[5] but in combination with Theorem 1.

▶ **Theorem 9.** *For any constant dimension d, given a set S of n c-fat objects in $[0, U)^d$ as a dynamic geometric stream, where the diameter of each object is at least 1 and each object has a weight in $[U]$, we can compute an $O((1/\delta)^2)$-approximation of the maximum independent set weight for S using $O(U^{O(\delta)} \log^{O(1)} n)$ space and update time w.h.p.*

Proof. We assume that all objects of S are $O(d)$-good. This is without loss of generality by the shifting lemma (Fact 7): for each of the $d+1$ shifts v_j ($j \in \{0, \ldots, d\}$), we can solve the problem for the good objects of $S + v_j$ in parallel, and return the maximum of the answers. The approximation ratio increases by a factor of $d + 1 = O(1)$.

Furthermore, we assume that all objects have weights in a common interval $[b^k, b^{k+1})$. This is without loss of generality, if we increase the approximation ratio by an extra factor of $O(\log_b U)$ (since there are $O(\log_b U)$ choices for k). By rescaling, we may now assume that all objects have weights in $[1, b)$.

Consider an object $s \in S$. Let γ be the smallest quadtree box containing s such that $\mathrm{diam}(\gamma)$ is a power of b (the parameter b will be a power of 2). Suppose that $\mathrm{diam}(\gamma) = b^\ell$. We place s in a set S_γ; in turn, we place the set S_γ in a collection $\mathcal{S}_\ell$.[6]

For each $\ell \in [\log_b U]$, we maintain an $O(1)$-approximation of $T_\ell = \sum_{S_\gamma \in \mathcal{S}_\ell} f_\ell(S_\gamma)$ using Theorem 1. Here, $f_\ell(S)$ is defined as the maximum independent set weight for $S^\# = \{s^\# : s \in S\}$, where $s^\#$ denotes the object s shifted by $-p_\gamma$, where γ is the quadtree box containing s with $\mathrm{diam}(\gamma) = b^\ell$, and p_γ is the lexicograpically smallest vertex of γ, assuming that s is not contained in any quadtree box with diameter $b^{\ell-1}$ (if the assumption is not satisfied, make $s^\#$ undefined). Note that for an individual set $S_\gamma \in \mathcal{S}_\ell$, $f(S_\gamma)$ is the same as the maximum independent set weight for S_ℓ. Also note that every object $s^\#$ intersects the boundary of one of the b^d quadtree boxes of diameter $b^{\ell-1}$ contained in $[0, b^\ell)^d$, and so we can maintain $f_\ell(S)$ using Lemma 4 after replacing b with b^d (note that $f_\ell(S) \leq b^{O(1)}$, since all objects have weights in $[1, b)$). The conditions of Theorem 1 are fulfilled with $B = b^{O(1)}$ and $s(n), t(n) = O(b^{O(1)} \log^{O(1)}(nU))$. We return the maximum of the approximations of T_ℓ over all $\ell \in [\log_b U]$.

Observe that T_ℓ is just the maximum independent set weight for $\bigcup_{S_\gamma \in \mathcal{S}_\ell} S_\gamma$, because rectangles in S_{γ_1} do not overlap with rectangles in S_{γ_2} for any two different sets $S_{\gamma_1}, S_{\gamma_2} \in \mathcal{S}_\ell$. Thus, OPT (the maximum independent set weight for S) is at most $\sum_\ell T_\ell \leq O(\log_b U) \cdot$

[5] Bhore and Chan [8] needed a balanced version of quadtrees (requiring cells that are differences of two quadtree boxes), but we are able to simplify the divide-and-conquer because of the bounded universe assumption.

[6] Each S_γ corresponds to a node in the b-ary variant of the quadtree. Each $\mathcal{S}_\ell$ corresponds to one level of the quadtree.

$\max_\ell T_\ell$. It follows that the returned value is an $O(\log_b U)$-approximation of OPT, under the assumption that objects have weights in $[1, b)$. We thus obtain approximation ratio $O((\log_b U)^2)$ for the general problem. Finally, we set $b = U^\delta$. ◀

▶ Remark 10. Without weights, the approximation ratio is $O(1/\delta)$. If the ratio of the maximum to the minimum diameter of the objects and the ratio of the maximum to the minimum weight of the objects are bounded by Δ, then the U^δ factors can be reduced to Δ^δ.

As in Bhore and Chan's paper [8], the approach also works for the minimum piercing set problem for fat objects.

References

1 Anna Adamaszek, Sariel Har-Peled, and Andreas Wiese. Approximation schemes for independent set and sparse subsets of polygons. *J. ACM*, 66(4):29:1–29:40, 2019. doi:10.1145/3326122.

2 Pankaj K. Agarwal, Marc J. van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. *Comput. Geom.*, 11(3-4):209–218, 1998. doi:10.1016/S0925-7721(98)00028-5.

3 Alexandr Andoni and Huy L. Nguyên. Width of points in the streaming model. *ACM Trans. Algorithms*, 12(1):5:1–5:10, 2016. doi:10.1145/2847259.

4 Ainesh Bakshi, Nadiia Chepurko, and David P. Woodruff. Weighted maximum independent set of geometric objects in turnstile streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 176 of *LIPIcs*, pages 64:1–64:22, 2020. doi:10.4230/LIPICS.APPROX/RANDOM.2020.64.

5 Piotr Berman, Bhaskar DasGupta, S. Muthukrishnan, and Suneeta Ramaswami. Improved approximation algorithms for rectangle tiling and packing. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 427–436, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365496.

6 Sujoy Bhore, Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Dynamic geometric independent set. In *Abstracts of 23rd Thailand-Japan Conference on Discrete and Computational Geometry, Graphs, and Games (TJDCG)*, 2021. arXiv:2007.08643.

7 Sujoy Bhore and Timothy M. Chan. Dynamic independent set of disks (and hypercubes) made easier. In *Proceedings of the 8th SIAM Symposium on Simplicity in Algorithms (SOSA)*, pages 485–495, 2025. doi:10.1137/1.9781611978315.36.

8 Sujoy Bhore and Timothy M. Chan. Fast static and dynamic approximation algorithms for geometric optimization problems: Piercing, independent set, vertex cover, and matching. In *Proceedings of the 36th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2357–2386, 2025. doi:10.1137/1.9781611978322.79.

9 Sujoy Bhore, Fabian Klute, and Jelle J. Oostveen. On streaming algorithms for geometric independent set and clique. In *Proceedings of the 20th International Workshop on Approximation and Online Algorithms (WAOA)*, volume 13538 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 2022. doi:10.1007/978-3-031-18367-6_11.

10 Sujoy Bhore, Martin Nöllenburg, Csaba D. Tóth, and Jules Wulms. Fully dynamic maximum independent sets of disks in polylogarithmic update time. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG)*, volume 293 of *LIPIcs*, pages 19:1–19:16, 2024. doi:10.4230/LIPICS.SOCG.2024.19.

11 Vladimir Braverman, Gereon Frahling, Harry Lang, Christian Sohler, and Lin F. Yang. Clustering high dimensional dynamic data streams. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 576–585, 2017. URL: http://proceedings.mlr.press/v70/braverman17a.html.

12 Sergio Cabello and Pablo Pérez-Lantero. Interval selection in the streaming model. *Theor. Comput. Sci.*, 702:77–96, 2017. doi:10.1016/J.TCS.2017.08.015.

13 Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Worst-case efficient dynamic geometric independent set. In *Proceedings of the 29th Annual European Symposium on Algorithms (ESA)*, volume 204 of *LIPIcs*, pages 25:1–25:15, 2021. doi:10.4230/LIPICS.ESA.2021.25.

14 Parinya Chalermsook and Julia Chuzhoy. Maximum independent set of rectangles. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 892–901. SIAM, 2009. doi:10.1137/1.9781611973068.97.

15 Timothy M. Chan. Approximate nearest neighbor queries revisited. *Discret. Comput. Geom.*, 20(3):359–373, 1998. doi:10.1007/PL00009390.

16 Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46(2):178–189, 2003. doi:10.1016/S0196-6774(02)00294-8.

17 Timothy M. Chan. A note on maximum independent sets in rectangle intersection graphs. *Inf. Process. Lett.*, 89(1):19–23, 2004. doi:10.1016/J.IPL.2003.09.019.

18 Timothy M. Chan. Dynamic streaming algorithms for epsilon-kernels. In *Proceedings of the 32nd International Symposium on Computational Geometry (SoCG)*, volume 51 of *LIPIcs*, pages 27:1–27:11, 2016. doi:10.4230/LIPICS.SOCG.2016.27.

19 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.

20 Xiaoyu Chen, Shaofeng H.-C. Jiang, and Robert Krauthgamer. Streaming Euclidean max-cut: Dimension vs data reduction. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 170–182, 2023. doi:10.1145/3564246.3585170.

21 Julia Chuzhoy and Alina Ene. On approximating maximum independent set of rectangles. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 820–829, 2016. doi:10.1109/FOCS.2016.92.

22 Jana Cslovjecsek, Michał Pilipczuk, and Karol Wegrzycki. A polynomial-time $\mathrm{OPT}^{\varepsilon}$-approximation algorithm for maximum independent set of connected subgraphs in a planar graph. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 625–638, 2024. doi:10.1137/1.9781611977912.23.

23 Artur Czumaj, Shaofeng H.-C. Jiang, Robert Krauthgamer, and Pavel Veselý. Streaming algorithms for geometric Steiner forest. *ACM Trans. Algorithms*, 20(4):28:1–28:38, 2024. doi:10.1145/3663666.

24 Artur Czumaj, Christiane Lammersen, Morteza Monemizadeh, and Christian Sohler. $(1+\epsilon)$-approximation for facility location in data streams. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1710–1728, 2013. doi:10.1137/1.9781611973105.123.

25 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.

26 Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM Journal on Computing*, 34(6):1302–1323, 2005. doi:10.1137/S0097539702402676.

27 Jacob Fox and János Pach. Computing the independence number of intersection graphs. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1161–1165, 2011. doi:10.1137/1.9781611973082.87.

28 Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. *Int. J. Comput. Geom. Appl.*, 18(1/2):3–28, 2008. doi:10.1142/S0218195908002520.

29 Gereon Frahling and Christian Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 209–217. ACM, 2005. doi:10.1145/1060590.1060622.

30 Waldo Gálvez, Arindam Khan, Mathieu Mari, Tobias Mömke, Madhusudhan Reddy Pittu, and Andreas Wiese. A $(2+\varepsilon)$-approximation algorithm for maximum independent set of rectangles. *CoRR*, abs/2106.00623, 2021. arXiv:2106.00623.

31 Waldo Gálvez, Arindam Khan, Mathieu Mari, Tobias Mömke, Madhusudhan Reddy Pittu, and Andreas Wiese. A 3-approximation algorithm for maximum independent set of rectangles. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 894–905, 2022. doi:10.1137/1.9781611977073.38.

32 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic approximate maximum independent set of intervals, hypercubes and hyperrectangles. In *Proceedings of the 36th International Symposium on Computational Geometry (SoCG)*, volume 164 of *LIPIcs*, pages 51:1–51:14, 2020. doi:10.4230/LIPICS.SOCG.2020.51.

33 Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, 1985. doi:10.1145/2455.214106.

34 Wei Hu, Zhao Song, Lin F Yang, and Peilin Zhong. Nearly optimal dynamic k-means clustering for high-dimensional data. *arXiv preprint*, 2018. arXiv:1802.00459.

35 Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 373–380, 2004. doi:10.1145/1007352.1007413.

36 Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for L_p samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 49–58, 2011. doi:10.1145/1989284.1989289.

37 Sanjeev Khanna, S. Muthukrishnan, and Mike Paterson. On approximating rectangle tiling and packing. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 384–393, 1998. URL: http://dl.acm.org/citation.cfm?id=314613.314768.

38 Christiane Lammersen, Anastasios Sidiropoulos, and Christian Sohler. Streaming embeddings with slack. In *Proceedings of the 11th International Symposium on Algorithms and Data Structures (WADS)*, volume 5664 of *Lecture Notes in Computer Science*, pages 483–494. Springer, 2009. doi:10.1007/978-3-642-03367-4_42.

39 Joseph S. B. Mitchell. Approximating maximum independent set for rectangles in the plane. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 339–350, 2022. doi:10.1109/FOCS52979.2021.00042.

40 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. doi:10.1017/CBO9780511814075.

41 Frank Nielsen. Fast stabbing of boxes in high dimensions. *Theor. Comput. Sci.*, 246(1-2):53–72, 2000. doi:10.1016/S0304-3975(98)00336-3.

Parameterized Streaming Algorithms for Topological Sorting

Ho-Lin Chen
National Taiwan University, Taipei, Taiwan

Peng-Ting Lin
Academia Sinica, Taipei, Taiwan

Meng-Tsung Tsai
Academia Sinica, Taipei, Taiwan

Abstract

Computing a topological ordering for an n-node directed acyclic graph (DAG) G is computationally challenging in streaming models. Chakrabarti et al. [SODA 2020] showed that in the insertion-only streaming model, every single-pass algorithm requires $\Omega(n^2)$ space, and every k-pass algorithm requires $n^{1+\Omega(1/k)}$ space for any constant $k \geq 1$.

We study the parameterized complexity of streaming algorithms for topological sorting, considering two parameters: the independence number α and the maximum displacement δ. Our results include an $O(1/\varepsilon)$-pass $O(\alpha n^{1+\varepsilon})$-space streaming algorithm and an $O(n^{1/2})$-pass $O(n+\delta^2)$-space streaming algorithm. For dense random DAGs, both α and δ are small, allowing us to improve the state-of-the-art for topological sorting in random DAGs.

As applications, we show that strongly connected components (SCC) decomposition and 2-satisfiability (2-SAT) can be solved in $O(1/\varepsilon)$ passes using $O(\alpha n^{1+\varepsilon})$ space and $O(\alpha_I n^{1+\varepsilon})$ space, respectively, where α_I denotes the independence number of the implication graph induced by the input 2-SAT instance.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Independence Number, Chain Cover, SCC Decomposition, 2-Satisfiability

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.18

Funding This research was supported in part by the National Science and Technology Council under contract NSTC 114-2221-E-001-023 and 113-2221-E-002-204-MY3.

Acknowledgements We want to thank the anonymous reviewers for their valuable comments.

1 Introduction

We study the problem of computing a topological ordering of a given n-node directed acyclic graph (DAG) $G = (V, E)$ using space subquadratic in the number of nodes. Specifically, the goal is to output an ordering $(v_1, v_2, \ldots, v_n)$ of the nodes in V using $O(n^{2-\varepsilon})$ space for some constant $\varepsilon > 0$, such that for every $i, j \in [n]$[1] with $j > i$, there is no directed path in G from v_j to v_i. This problem has been studied by Chakrabarti et al. [5], who showed that in the insertion-only streaming model, every single-pass algorithm requires $\Omega(n^2)$ space, and every k-pass algorithm requires $n^{1+\Omega(1/k)}$ space for any constant $k \geq 1$.

For problems that require substantial space when computed with few passes in the streaming model, a natural direction is to study their parameterized complexity. This approach was introduced by Fafianie and Kratsch for edge dominating set [10] and by

[1] We define $[n] \coloneqq \{1, 2, \ldots, n\}$.

© Ho-Lin Chen, Peng-Ting Lin, and Meng-Tsung Tsai;
licensed under Creative Commons License CC-BY 4.0

Chitnis et al. for maximal matching and vertex cover [7]. Since then, the parameterized complexity of various problems has been explored. Chitnis and Cormode studied treewidth and dominating set [6]. Agrawal et al. studied Min-Ones SAT [1]. Oostveen and van Leeuwen studied diameter and connectivity [21]. Lokshtanov et al. studied feedback vertex set and multiway cut [17].

Our model of computation follows the *graph streaming model* [20, 18]. The edges of the input graph $G = (V, E)$ are presented in an order determined by an adversary who has access to the algorithm but not its random seed. The algorithm processes the edges sequentially, making up to p passes over the input while using at most s space. During each pass, the algorithm can only read the stream forward and cannot backtrack. We refer to such an algorithm as a p-pass s-space streaming algorithm. If the edges are static, meaning only edge insertions occur in the stream, the model is called the *insertion-only model*. In contrast, if both edge insertions and deletions are allowed, where deletions can only remove previously inserted edges, the model is referred to as the *turnstile model*.

Our first main result shows that any n-node directed acyclic graph G with independence number α can be topologically sorted in $O(\log n)$ passes using $O(\alpha n)$ space. A directed graph G has *independence number* α if there exists a set of α nodes in G such that no two nodes in the set are joined by an edge in G, and no larger set satisfies this property. The number of passes can be further reduced to $O(1/\varepsilon)$ for every $\varepsilon > 0$ if $O(\alpha n^{1+\varepsilon})$ space is available. Additionally, we provide a smooth tradeoff between passes and space. Formally, we have:

▶ **Theorem 1.** *Let G be an n-node directed acyclic graph with independence number α. For every $\varepsilon > 0$, there exists an $O(1/\varepsilon)$-pass $O(\alpha n^{1+\varepsilon})$-space deterministic streaming algorithm to topologically sort G in the insertion-only model. Moreover, there is a smooth tradeoff between passes and space: for every constant $r \in (0, 1)$, there exists an $O(n^r)$-pass $O(n + \alpha n^{1-r/2} \log n)$-space streaming algorithm with the same functionality with probability $1 - 1/n^{\Omega(1)}$.*

Another main result is that any n-node directed acyclic graph G with an advice A of maximum displacement δ can be topologically sorted in $O(n^{1/2})$ passes using $O(n + \delta^2)$ space. For an n-node acyclic graph $G = (V, E)$, a function $A : V \to \mathbb{Z}$ is an *advice* of G with *maximum displacement* δ if G admits a topological ordering $(v_1, v_2, \ldots, v_n)$ such that for every $i \in [n]$, the displacement satisfies $|i - A(v_i)| \leq \delta$. Note that the function A is not required to be injective. We provide also a smooth tradeoff between passes and space. Formally, we have:

▶ **Theorem 2.** *Let G be an n-node directed acyclic graph with an advice A of maximum displacement δ. For every $t \in [n]$, there exists an $O(n/t)$-pass $O(n+t\delta+\delta^2)$-space deterministic streaming algorithm in the insertion-only model to topologically sort G.*

The above two algorithms can be applied to random directed acyclic graphs (random DAGs) $G \sim \mathcal{G}(n, p)$, where each edge is included independently with probability p. In random DAGs, the independence number can be estimated quite accurately, and an advice A can be provided using the in-degree of the nodes. Using these facts, we have an $\tilde{O}(1)$-pass $\tilde{O}(np^{-1})$-space streaming algorithm and an $O(\sqrt{np})$-pass $\tilde{O}(n)$-space streaming algorithm. These algorithms have almost identical pass and space requirement as [5]. But, unlike [5] which requires a chain that goes through all nodes, our algorithms apply to random DAGs $\mathcal{G}(n, p)$.

Furthermore, both of our algorithm exhibit smooth trade-offs between pass and space. Also, both of our algorithms and their analysis directly applies to graphs of type $G \cup H$, where $G \sim \mathcal{G}(n,p)$ is a random DAG and H can be any given graph with maximum in-degree $o(\sqrt{np\log n})$ that has the same node set and the same topological ordering as G.

The final main result gives a deterministic algorithm in the turnstile model to compute a topological ordering for any directed acyclic graph, stated below:

▶ **Theorem 3.** *Let G be an n-node directed acyclic graph. For every $k \in [n]$, there exists an k-pass $O(n^2/k)$-space deterministic streaming algorithm in the turnstile model to topologically sort G.*

1.1 Technical Ingredients

If a directed acyclic graph $G = (V, E)$ has independence number α, then every node-induced subgraph of G admits a chain cover of at most α chains. This is a stronger condition than simply assuming that a largest antichain in G contains at most α nodes, a parameter studied extensively for algorithms on a RAM [16, 4], as our condition ensures a structural property that holds for all node-induced subgraphs, enabling a divide-and-conquer approach to compute a chain cover consisting of few chains. Since the chain cover consists of α chains, the set of nodes reachable from any node x in G can be succinctly represented by a tuple of α integers. Combined with a distributed variant of the Bellman-Ford algorithm, this forms the basis of our first algorithm.

Our second and third algorithms rely on the concept of a *source subgraph*, a node-induced subgraph of G defined by a set S such that no edge in G enters S from $V \setminus S$. The second algorithm leverages the maximum displacement δ to construct a subgraph H that fits in memory and contains an r-node source subgraph of G for some sufficiently large r. The third algorithm refines G by removing a majority of out-neighbors from high-degree nodes, producing a sparse subgraph (in terms of edge density) that also contains a large source subgraph of G. Consequently, topological sorting of G can be performed by iteratively peeling away a source subgraph. Since each source subgraph contains a moderate number of nodes, the number of peeling rounds remains low.

1.2 Applications

Finally, in Section 7, we show that the idea used in our topological sort algorithm can be applied to SCC decomposition, yielding an $O(1/\varepsilon)$-pass $O(\alpha n^{1+\varepsilon})$-space deterministic algorithm for any $\varepsilon > 0$, as well as a ranodmized $O(k)$-pass $O((n^2 \log n)/k)$-space. Since SCC decomposition can be directly applied to solve 2-SAT, these results also extend to the 2-SAT problem.

1.3 Paper Organization

The paper is organized as follows. In Section 2, we introduce our notation. In Section 3, we develop algorithms for topologically sorting directed acyclic graphs with independence number α, proving Theorem 1. In Section 4, we design algorithms for topologically sorting directed acyclic graphs given an advice function with maximum displacement δ, thereby proving Theorem 2. We then combine the results from Section 3 and Section 4 to obtain a result for random DAGs in Section 5. In Section 6, we introduce a deterministic algorithm in the turnstile model for topologically sorting general DAGs. Finally, in Section 7, we discuss selected applications of topological sorting, including SCC decomposition and 2-satisfiability.

2 Notation

Our input graph is an n-node directed acyclic graph $G = (V, E)$ in which each directed edge $(u, v) \in E$ is from node u to node v. Let $[n]$ denote the set $\{1, 2, \ldots, n\}$ and $[a, b]$ (resp. (a, b)) denote the closed (resp. open) interval between a and b. A *topological ordering* of G is an ordering $(v_1, v_2, \ldots, v_n)$ of the nodes in V such that there is no directed path in G from v_j to v_i for any $j > i$ with $i, j \in [n]$. We use $G[S]$ to denote the *node-induced subgraph* of G induced by the node set S; that is, $G[S] = (S, E^S)$ where all edges in E with both endnodes in S form E^S. We use $G \sim \mathcal{G}(n, p)$ to denote an n-node random directed acyclic graph G where each edge is included in G with probability p independently, and orient the directions of all edges based on a random permutation on the n nodes from an earlier node in the permutation to a later node.

For each node x in G, if there is a directed edge $(x, y) \in E$, we say that y is an *out-neighbor* of x, and x is an *in-neighbor* of y. A node x with no in-neighbors (resp. no out-neighbors) is called a *source* (resp. a *sink*). We say that a node x has *in-degree* (resp. *out-degree*) k if it has k in-neighbors (resp. out-neighbors), denoted as $d_{in}(x) = k$ (resp. $d_{out}(x) = k$).

The *transitive closure* of a directed graph $G = (V, E)$ is the graph $G^{tc} = (V, E^{tc})$, where $(u, v) \in E^{tc}$ if and only if there exists a directed path from u to v in G. A *chain* of G is a sequence of nodes $(v_1, v_2, \ldots, v_t)$ for some integer $t \geq 1$, such that $(v_1, v_2, \ldots, v_t)$ forms a directed path in the transitive closure G^{tc}. A *chain cover* of G is a collection of chains such that the nodes contained in the chains partition the node set V. An *antichain* of G is a set of nodes such that no directed path exists between any two distinct nodes u, v in the set.

We say that a directed graph G has *independence number* α if there exists a set of α nodes in G such that no two nodes in the set are joined by an edge in G, and no larger set satisfies this property.

For an n-node acyclic graph $G = (V, E)$, we say that a function $A : V \to \mathbb{Z}$ is an *advice* of G with *maximum displacement* δ if G admits a topological ordering $(v_1, v_2, \ldots, v_n)$ such that for every $i \in [n]$, the displacement satisfies $|i - A(v_i)| \leq \delta$.

In this paper, the space usage of each algorithm is measured in words, while the space lower bound of each problem is measured in bits. Therefore, if an algorithm's space usage matches the problem's space lower bound, the algorithm is considered nearly optimal in space usage, up to an $O(\log n)$ factor.

3 An $O(1/\varepsilon)$-Pass $O(\alpha n^{1+\varepsilon})$-Space Algorithm and Tradeoff

We begin with showing that the transitive closure of an n-node directed acyclic graph G with independence number α can be represented by the transitive closure of an $O(\alpha n)$-edge subgraph H of G. Since G and H have the same transitive closure, computing a topological ordering of H suffices to topologically sort G. Then, we devise a streaming algorithm in the insertion-only model to compute H, which yields an $O(1/\varepsilon)$-pass $O(\alpha n^{1+\varepsilon})$-space streaming algorithm to topologically sort any given n-node acyclic directed graph with independence number α. Moreover, there is a smooth tradeoff between passes and space: for every constant $r \in (0, 1)$, there exists an $O(n^r)$-pass $O(n + \alpha n^{1-r/2} \log n)$-space streaming algorithm with the same functionality with probability $1 - 1/n^{\Omega(1)}$. This proves Theorem 1.

▶ **Lemma 4.** *Every directed acyclic graph G with independence number α contains an $O(\alpha n)$-edge subgraph H such that G and H have the same transitive closure. Moreover, if a graph G' has the same transitive closure as G, then G' also contains an $O(\alpha n)$-edge subgraph with the same transitive closure, even if the independence number of G' exceeds α.*

Proof. Let S be an antichain in G; that is, a subset of nodes in G such that for any two nodes u and v in the subset, there exists no directed path from u to v. Thus, S is an independent set and $|S| \leq \alpha$. This property holds also for any largest antichain in G. By Dilworth's theorem [9], G has a chain cover consisting of at most α chains. For each node x in G, if x has out-degree greater than α, then by the pigeonhole principle, there exist two out-neighbors u and v of x that belong to the same chain in the chain cover. Let the chain be $a_1 \to a_2 \to \cdots \to a_t$ for some $t \geq 2$ and $u = a_i, v = a_j$ for some $i < j$. If the edge (x, v) is removed from G, then the resulting graph has the same transitive closure as G. This holds because x can still reach v via the edge (x, u) followed by a directed path P that witnesses the chain $a_i \to a_j$. Note that x is not on P, as G is acyclic, and neither is the edge (x, v). Consequently, one can iteratively remove an edge from G while retaining the transitive closure unchanged until every node in G has out-degree at most α.

The same argument works for any graph G' that has the same transitive closure as G, even if the independence number of G' exceeds α. ◀

One may find that Lemma 4 still holds if the number of nodes in a largest antichain in G is α. However, it is essential for our algorithm to have the independence number of G bounded by α. In this way, every node-induced subgraph of G (rather than only G) has an antichain consisting of at most α chains, which is the key to find the subgraph H efficiently in the streaming model.

First, we find the chain cover used in the proof of Lemma 4 as follows.

▶ **Lemma 5.** *Let G be an n-node acyclic directed graph with independence number α. For every $\varepsilon > 0$, there exists an $O(1/\varepsilon)$-pass $O(\alpha n^{1+\varepsilon})$-space streaming algorithm that outputs a chain cover of G consisting of α chains.*

Proof. We begin with an $O(\log n)$-pass $O(\alpha n)$-space streaming algorithm. First, we partition the nodes of G into subsets $\{S_{2i-1}, S_{2i} : i \in [t]\}$ arbitrarily for some integer $t = O(n)$, each containing $O(1)$ nodes. We then scan the stream of all edges in G once. For each $j \in [2t]$, we collect all edges in the node-induced subgraph $G[S_j]$ in memory. Next, we apply the approach from the proof of Lemma 4 to compute a chain cover consisting of α chains for $G[S_j]$ and trim $G[S_j]$ into a sparse subgraph with $O(\alpha|S_j|)$ edges.

We then scan the stream of all edges in G again. For each pair of consecutive node-induced subgraphs $G[S_{2i-1}]$ and $G[S_{2i}]$, we collect edges with one endnode in S_{2i-1} and the other in S_{2i} (i.e., crossing edges). Since we have a chain cover consisting of α chains for both $G[S_{2i-1}]$ and $G[S_{2i}]$, at most α essential crossing edges per node in S_{2i-1} and S_{2i} are required to preserve the transitive closure. This can be done using $O(\alpha(|S_{2i-1}| + |S_{2i}|))$ space because, for each of the α chains in the node-induced subgraph S_{2i-1} (resp. S_{2i}), each node x in S_{2i} (resp. S_{2i-1}) only needs to keep the single out-neighbor y that appears first in the chain. Finally, we replace the union of $G[S_{2i-1}]$, $G[S_{2i}]$, and their essential crossing edges with an $O(\alpha(|S_{2i-1}| + |S_{2i}|))$-edge subgraph H, ensuring that H and $G[S_{2i-1} \cup S_{2i}]$ have the same transitive closure and thus the chain cover of H consists of at most α chains. Such a sparse subgraph H can be found because the union of $G[S_{2i-1}]$, $G[S_{2i}]$, and the essential crossing edges has the same transitive closure as $G[S_{2i-1} \cup S_{2i}]$ and hence we can apply Lemma 4 though the independence number of the union may exceed α.

The above process reduces the number of subsets by half. The space usage remains $O(\alpha n)$. Repeating this process for $O(\log n)$ iterations merges all subsets into a single subgraph with a chain cover of α chains.

The above approach groups two subsets at a time. More generally, we can group n^ε subsets at a time. This increases space usage by a factor of n^ε while reducing the number of passes to

$$O(\log_{n^\varepsilon} n) = O(1/\varepsilon).$$ ◀

Then, we use the chain cover found in Lemma 5 to compute H as follows.

▶ **Corollary 6.** *Let G be an n-node acyclic directed graph with independence number α. For every $\varepsilon > 0$, there exists an $O(1/\varepsilon)$-pass $O(\alpha n^{1+\varepsilon})$-space streaming algorithm that outputs an $O(\alpha n)$-edge subgraph H such that G and H have the same transitive closure.*

Proof. By Lemma 5, we obtain a chain cover of G consisting of α chains using the claimed number of passes and space. Then, in another pass over all edges of G, for each node x, we store all of its out-neighbors in a list. If the list exceeds α out-neighbors, at least one can be discarded, as established in the proof of Lemma 4. ◀

Finally, given the subgraph H, a topological ordering of G can be computed by topologically sorting H. This gives an $O(1/\varepsilon)$-pass $O(\alpha n^{1+\varepsilon})$-space deterministic streaming algorithm to compute a topological ordering for any n-node directed acyclic graph G in the insertion-only model. This proves the few-pass algorithm stated in Theorem 1.

3.1 Tradeoff Between Passes and Space

We give a smooth tradeoff between passes and space: for every constant $r \in (0, 1)$, there exists an $O(n^r)$-pass $O(n + \alpha n^{1-r/2})$-space streaming algorithm with the same functionality. Our divide-and-conquer approach is inspired by the parallel topological sorting method of Schudy [22]. However, since reachability is computationally expensive in streaming models, we overcome this limitation by leveraging the concept of chain cover, which is a novel aspect of our approach compared to Schudy's result.

▶ **Lemma 7.** *Given an n-node directed acyclic graph G and a topological ordering of $\Omega(k \log n)$ nodes, sampled independently and uniformly at random (possibly with repetition), there exists an $O(n/k)$-pass $O(n)$-space streaming algorithm that topologically sorts G with probability $1 - 1/n^{\Omega(1)}$.*

Proof. We add a new node s to G, with a directed edge to every node in G that has in-degree 0. Let R be a set of $\Omega(k \log n)$ sampled nodes, denoted as $R = \{v_1, v_2, \ldots, v_{|R|}\}$, such that no directed path in G starts from v_i to v_j for any $i > j$, where $i, j \in [|R|]$. Define $v_0 \coloneqq s$.

For each integer $i \in [0, |R|]$, let U_i be the set of nodes in G that are reachable from v_i but not from any v_j with $j > i$. Since s can reach every node in G, the sets U_i for all integer $i \in [0, |R|]$ form a partition of V. Therefore, storing all U_i in memory needs $O(n)$ space. Given this partition, a topological ordering of G can be obtained by first sorting the nodes within each U_i independently and then concatenating their orderings.

Since R is a randomly selected subset of $\Omega(k \log n)$ nodes, the sets U_i satisfy the following property:

▷ Claim 8. For each integer $i \in [0, |R|]$ and each node $x \in U_i$, any longest path in G (with unweighted edges) from v_i to x contains at most n/k edges with probability $1 - 1/n^{\Omega(1)}$.

Proof. Let V be the set of nodes in G, including s. Consider any pair of nodes $y, z \in V$ such that z is reachable from y. Fix a longest path P_{yz} from y to z, and let $\mathcal{C}$ be the collection of such paths P_{yz} whose lengths exceed n/k. Since there are at most $\binom{n}{2}$ such pairs, we have $|\mathcal{C}| \leq \binom{n}{2}$.

The probability that none of the internal nodes in P_{yz} (excluding y and z) is included in R is

$$(1 - 1/k)^{\Omega(k \log n)} = 1/n^{\Omega(1)}.$$

Applying the union bound, we conclude that with probability at least $1 - 1/n^{\Omega(1)}$, every path in $\mathcal{C}$ contains at least one internal node in R.

Consequently, if there exists a node $x \in U_i$ such that a longest path P from v_i to x contains more than n/k edges, then with probability at least $1 - 1/n^{\Omega(1)}$, some internal node v_j in P belongs to R. Since there is a path from v_i to v_j, we have $j > i$, contradicting the assumption that x is in U_i. ◁

To match the claimed number of passes and space, we design a distributed variant of the Bellman-Ford algorithm (stated also in Algorithm 1). Assign a weight of -1 to all edges in G. Then, the shortest distance from s to any node $x \in V$ corresponds to the length of the longest path (in terms of edge count) from s to x. Sorting the nodes by their distances from s yields a topological ordering. However, computing the distances from s may require many passes or substantial space.

Instead of running the Bellman-Ford algorithm from a single source node s, we execute it in parallel from every node in the set R. If a node x is reachable from two nodes v_i and v_j with $i < j$, we discard the information from v_i, since x does not belong to U_i. In Algorithm 1, this behavior is enforced by the rank function r, which is set such that $r(v_i) > r(v_j)$ for all $i, j \in [0, |R|]$ with $i < j$, ensuring that only information from the appropriate source is retained. As a result, the space usage $O(1)$ per node and $O(n)$ in total. By Claim 8, the Bellman-Ford algorithm terminates after $O(n/k)$ iterations.

The node ordering returned by Algorithm 1 is a topological ordering of G because it topologically sorts the nodes in each U_i by their longest distances (in term of edge count) from v_i. Moreover, the concatenation of the topological orderings of nodes in U_is for all $i \in [0, |R|]$ cannot have any edge (x, y) in G for some $x \in U_j$ and $y \in U_i$ with $i < j$, as it would violate the definition of the sets U_i. ◀

▶ **Lemma 9.** *Given an n-node directed acyclic graph G, a chain cover of G consisting of β chains, and $\Omega(k \log n)$ nodes sampled uniformly at random from G independently (possibly with repetition), there exists an $O(n/k)$-pass $O(n + \beta k \log n)$-space streaming algorithm that topologically sorts these sampled nodes with probability $1 - 1/n^{\Omega(1)}$.*

Proof. We topologically sort the sampled nodes by ordering them based on the cardinalities of their reachable sets. Let R be the set of sampled nodes. For each $x \in R$, we perform a depth-$O(n/k)$ BFS rooted at x. The β chains can be stored in memory using $O(n)$ space. Each BFS computation requires an additional $O(\beta)$ space, as detailed below, and across all BFSs, the exploration of nodes at depth d is performed simultaneously. Consequently, the total number of passes is $O(n/k)$.

We execute each of these BFSs as follows. The key difference between our BFS and a standard BFS is that we define the *child nodes* of any node x to include both its out-neighbors and the out-neighbors of all descendant nodes of x along the chain to which x belongs. Since we are only interested in the set of nodes that x can reach, we do not need to maintain the

Algorithm 1 A distributed Bellman-Ford algorithm for topological sorting.

Input: A directed graph $G = (V \cup \{s\}, E)$ where s can reach all nodes and all edges have weight -1, a random subset $R \subseteq V$, and a rank function $r : R \cup \{s\} \to \mathbb{N}$ such that no directed path in G starts from a node with a smaller r-value to one with a larger r-value.
Output: A topological ordering of G.

```
d[x] ← (∞, ∞) for all x ∈ V;
d[y] ← (r(y), 0) for all y ∈ R ∪ {s};
for i = 1 to t = O(n log n/|R|) do
    foreach (u, v) ∈ E do
        if d[u] + (0, −1) < d[v] then
            // coordinate-wise additions and comparisons, prioritizing the first
               coordinate
            d[v] ← d[u] + (0, −1);
        end
    end
end
return nodes x ∈ V in non-decreasing order of d values;
```

full structure of a standard BFS. Thus, for each BFS level, we only need to store at most β nodes. This is because if two nodes belong to the same chain, where one is an ancestor of the other, then the child nodes of the ancestor form a superset of those of the descendant.

We only need to perform depth-$O(n/k)$ BFSs for all $x \in R$ to compute the reachable sets for all $x \in R$. For every pair of nodes $x, y \in G$, consider a shortest path from x to y. Let $\mathcal{C}$ be the collection of all such shortest paths of length at least n/k. For each path in $\mathcal{C}$, the probability that none of its internal nodes belong to R is

$$(1 - 1/k)^{\Omega(k \log n)} \leq 1/n^{\Omega(1)}.$$

Applying the union bound over all paths in $\mathcal{C}$, where $|\mathcal{C}| \leq \binom{n}{2}$, we conclude that with probability $1 - 1/n^{\Omega(1)}$, every path in $\mathcal{C}$ contains at least one internal node in R. Consequently, if there exists a node y that is reachable from x via a shortest path longer than n/k, then it must be visible from the $O(n/k)$-depth BFS rooted at some node in R. ◀

We are now ready to prove the tradeoff stated in Theorem 1. Let r be any constant in $(0, 1)$. First, we partition the node set of G into $n^{r/2}$ subsets arbitrarily, each containing $O(n^{1-r/2})$ nodes. For each subset S, we apply Lemma 5, setting $\varepsilon \to 1/\log n$ and $n \to n^{1-r/2}$, to obtain a chain cover of the subgraph induced by S, which consists of α chains. This holds because any node-induced subgraph of G also has independence number α. Combining these chain covers across all subsets yields a chain cover of G with $O(\alpha n^{r/2})$ chains.

Next, we use this chain cover as input for Lemma 9, setting $\beta = \alpha n^{r/2}$, and then apply Lemma 7 to compute a topological ordering of G, where the parameter k in both lemmas is set to n^{1-r}. Consequently, the number of passes is

$$O(n^{r/2} \log n + n/n^{1-r}) = O(n^r),$$

and the required space is

$$O(\alpha n^{1-r/2} + n + \alpha n^{r/2} n^{1-r} \log n) = O(n + \alpha n^{1-r/2} \log n).$$

This establishes the smooth tradeoff in Theorem 1.

3.2 Graph Classes with Small Independence Number

In what follows, we provide a brief proof (without claiming novelty) that a dense random DAG $G \sim \mathcal{G}(n,p)$ has a small independence number with high probability for every $p \in (0,1)$. Given the bounds on the number of nodes in a largest antichain in a random DAG due to Bollobás and Brightwell [3], our bound in Lemma 10 is nearly optimal, up to a logarithmic factor. We note that a result by Frieze [12] may eliminate this logarithmic factor, but it requires the assumption $p = o(1)$.

▶ **Lemma 10.** *For a random DAG $G \sim \mathcal{G}(n,p)$, where each edge is included independently with probability p, the independence number of G is $O(p^{-1}\log n)$ with probability $1 - 1/n^{\Omega(1)}$.*

Proof. Let $p = 1/k$, and define X_t as the number of independent sets of t nodes in G. Since $G \sim \mathcal{G}(n,p)$, we have

$$\mathbf{E}[X_{\lceil 3k\log n\rceil}] = \binom{n}{\lceil 3k\log n\rceil}(1-1/k)^{\binom{\lceil 3k\log n\rceil}{2}} \le n^{\lceil 3k\log n\rceil} e^{-1/k\binom{3k\log n}{2}} = e^{-\Omega(k\log^2 n)}.$$

Since X_t is non-negative, applying Markov's inequality gives

$$\Pr[X_{\lceil 3k\log n\rceil} \ge 1] \le e^{-\Omega(k\log^2 n)}.$$

Thus, with probability $1-1/n^{\Omega(1)}$, G contains no independent set of $\lceil 3k\log n\rceil = O(p^{-1}\log n)$ nodes or more. ◀

4 An $O(n^{1/2})$-Pass $O(n+\delta^2)$-Space Algorithm and Tradeoff

In this section, we present an $O(n/t)$-pass $O(n + t\delta + \delta^2)$-space topological sort algorithm assuming that an advice A with maximum displacement δ is given, where t can be any given number in $[n]$. This proves Theorem 2.

The main idea is that, given any $t \in [n]$, we devise a process that can find and sort a source subgraph of at least t nodes using a single pass and $O(t\delta + \delta^2)$ space. We say a node-induced subgraph $G[S]$ of G is a *source subgraph* if no edge in G is from a node outside S to a node in S. Our algorithm just repeats this process $O(n/t)$ times. In order to find a source subgraph, we need the following observation:

▶ **Lemma 11.** *Let G be an n-node directed acyclic graph with an advice A of maximum displacement δ. Given any $t \in [n]$. Let H_1 be the subgraph induced by all nodes v with $A(v) \le t + 3\delta$ and H_2 be the subgraph induced by all nodes v with $A(v) \le t + \delta$. If H_3 is a subgraph of H_2 and a source subgraph of H_1, then H_3 is a source subgraph of G.*

Proof. It suffices to show that no edge goes from $G \setminus H_3$ to H_3. Since H_3 is a source subgraph of H_1, we only need to prove that that no edge goes from $G \setminus H_1$ to H_3.

Let σ be a topological ordering of G such that $|\sigma(v) - A(v)| \le \delta$ for all nodes v. For any node $u_1 \in H_3$, we have $\sigma(u_1) \le A(u_1) + \delta \le t + 2\delta$. Similarly, for any node $u_2 \in G \setminus H_1$, we have $\sigma(u_2) > t + 2\delta$. Combining the two inequalities, we have $\sigma(u_1) < \sigma(u_2)$. Therefore, no edge goes from $G \setminus H_1$ to H_3, which completes the proof. ◀

Using the above lemma, to find a source subgraph in G, it suffices to remember the induced subgraph H_1 and find the maximum source subgraph H_3 that satisfies the conditions listed. This can be done by remembering the entire induced subgraph H_1 in a single pass. The following two lemmas show that keeping the entire subgraph H_1 takes $O(t\delta + \delta^2)$ space, and the resulting source subgraph has size at least t.

▶ **Lemma 12.** *Let G be an n-node directed acyclic graph with an advice A of maximum displacement δ. Given any $t \in [n]$. There are at most $t + 4\delta$ nodes v with $A(v) \leq t + 3\delta$.*

Proof. Let σ be a topological ordering of G such that $|\sigma(v) - A(v)| \leq \delta$ for all nodes v. Only the first $t + 4\delta$ nodes of σ can satisfy $A(v) \leq t + 3\delta$. ◀

Lemma 12 shows that the subgraph H_1 defined in Lemma 11 has $O(t + \delta)$ nodes. Since the maximum displacement between the advice and the true ordering is at most δ, any pair of nodes whose advice values differ by more than 2δ must appear in the same relative order as suggested by the advice. Therefore, when computing a topological ordering, it suffices to retain only the edges between nodes whose advice values differ by less than 2δ. These edges will be referred to as *relevant* edges. The subgraph H_1 can be stored in $O(t\delta + \delta^2)$ space if we only keep the relevant edges.

▶ **Lemma 13.** *Let G be an n-node directed acyclic graph with an advice A of maximum displacement δ. Given any $t \in [n]$. Let H_1 be the subgraph induced by all nodes v with $A(v) \leq t + 3\delta$ and H_2 be the subgraph induced by all nodes v with $A(v) \leq t + \delta$. There exists a subgraph H_3 of size at least t such that H_3 is a subgraph of H_2 and a source subgraph of H_1.*

Proof. Let σ be a topological ordering of G such that $|\sigma(v) - A(v)| \leq \delta$ for all nodes v. The subgraph H induced by the first t nodes in σ has $A(v) \leq t + \delta$ and thus is a subgraph of H_2. Since H is the first t nodes in a topological ordering, H has no incoming edges and thus is a source subgraph of H_1. ◀

We are now ready to present the algorithm stated in Theorem 2. Given a graph G and an advice function A, the algorithm makes a single pass over the input and stores the subgraph H_1 induced by all nodes v with $A(v) \leq t + 3\delta$, along with all relevant edges between them. It then identifies the largest source subgraph of H_1 by iteratively expanding H_3, starting from the empty set and adding nodes whose advice values are at most $t + \delta$ and whose in-neighbors have all already been included in H_3. By Lemma 13, this source subgraph, denoted H_3, contains at least t nodes. The algorithm sorts the nodes of H_3 and places them at the front of the final topological ordering. Repeating this process $O(n/t)$ times yields a topological ordering of the entire graph G.

According to Lemma 12, the space requirement for keeping the subgraph H_1 in each step is $O(t\delta + \delta^2)$. $O(n)$ space is needed to keep the final topological ordering. Therefore, the total space usage for our streaming algorithm is $O(n + t\delta + \delta^2)$. Note that when $t = n$, the algorithm just remembers all relevant edges in a single pass using space $O(n\delta)$.

5 Random DAGs

In this section, we consider the topological ordering of random directed acyclic graphs (random DAGs) defined in Section 2. We apply our algorithms described in Section 3 and Section 4 to random DAGs in the streaming model and obtain a hybrid algorithm comprised of the two. We conclude this section by comparing our results to the results in [5].

First, for dense random DAGs, we can apply our algorithm in Section 3. Lemma 10 shows that, for a random DAG $G \sim \mathcal{G}(n, p)$, where each edge is included independently with probability p, the independence number of G is $O(p^{-1} \log n)$ with probability $1 - 1/n^{\Omega(1)}$. Substituting this number into our algorithm in Section 3, we have an $O(1/\varepsilon)$-pass $O(n^{1+\varepsilon} p^{-1} \log n)$-space streaming algorithm for every $\varepsilon > 0$. Choosing ε to be $(\log n)^{-1}$, we have an $\tilde{O}(1)$-pass $\tilde{O}(np^{-1})$-space streaming algorithm.

Second, we apply our topological sort algorithm described in Section 4. In order to apply the algorithm, we must first obtain an advice A. For a random DAG $G \sim \mathcal{G}(n,p)$, we can obtain an advice with maximum displacement at most $O(\sqrt{p^{-1}n\log n})$ with probability $1-1/n^{\Omega(1)}$ using the incoming degrees of the nodes. The same technique has been used in [5] while considering a variant of random directed acyclic graphs.

▶ **Lemma 14.** *Let $G=(V,E)$ be a random DAG. For every node $v \in V$, $|d_{in}(v)/p - \sigma(v)| = O(\sqrt{p^{-1}n\log n})$ with probability $1-1/n^{\Omega(1)}$.*

Proof. For each node $v \in V$, the in-degree $d_{in}(v)$ is a random variable that follows a binomial distribution with $\sigma(v)-1$ trials and success probability p. By applying Chernoff Bound we have

$$\Pr\left[|d_{in}(v) - (\sigma(v)-1)p| = O(\sqrt{pn\log n})\right] = 1/n^{\Omega(1)}.$$ ◀

By Lemma 14, $A(v) = d_{in}(v)/p$ is an advice with maximum displacement $O(\sqrt{p^{-1}n\log n})$ with high probability. The calculation of A can be done in one pass and $O(n)$ space by counting the incoming degrees of the nodes.

Using $A(v) = d_{in}(v)/p$ as an advice, Theorem 2 directly gives an $O(n/t)$-pass $O(n+t\delta+\delta^2)$-space streaming algorithm where $\delta = O(\sqrt{p^{-1}n\log n})$. Furthermore, notice that the space requirement $O(t\delta+\delta^2)$ is for keeping the $O(\delta)$ relevant edges for each of $O(t+\delta)$ nodes in H_1. However, in a random graph $\mathcal{G}(n,p)$, the probability that all subgraphs that could potentially be chosen as H_1 have only $O(pt\delta+p\delta^2)$ edges is at least $1-1/n^{\Omega(1)}$. As a result, the algorithm described in Theorem 2 only uses $O(n/t)$-pass $O(t\sqrt{pn\log n}+n\log n)$-space with probability $1-1/n^{\Omega(1)}$. Choosing t to be $\sqrt{np^{-1}\log n}$, we have an $O(\sqrt{np/\log n})$-pass $O(n\log n)$-space streaming algorithm.

Combining the above algorithms, we have two topological sort algorithms for random DAGs with almost identical (up to poly-logarithmic improvement) pass and space requirement as [5]. Unlike [5] which requires a chain that goes through all nodes, our algorithms apply to random DAGs $\mathcal{G}(n,p)$.

Furthermore, both of our algorithm exhibit trade-offs between pass and space. Also, both of our algorithms and their analysis directly apply to graphs of type $G \cup H$, where $G \sim \mathcal{G}(n,p)$ is a random DAG and H can be any given graph with maximum in-degree $o(\sqrt{np\log n})$ that has the same node set and same node ordering as G. The independence number only decreases after adding the edge set of H to G, and the advice derived from Lemma 14 is still correct with high probability for sparse H.

6 A Deterministic Approach in the Turnstile Model

We devise a deterministic algorithm to compute a topological ordering for any directed acyclic graph in the turnstile model. This proves Theorem 3. Our technical ingredients for this deterministic result include a sparse certificate and the set reconciliation.

Let $G=(V,E)$ be an n-node directed acyclic graph. We say that a subgraph H of G is a *t-certificate* of topological sorting if for any topological ordering σ_H of H there exists a topological ordering σ_G of G so that the longest common prefix of σ_H and σ_G has length at least t. Our deterministic algorithm to compute σ_G (initialized as an empty sequence) is simply computing a topological ordering σ_H of a t-certificate H of G, appending the first t nodes in σ_H to σ_G, followed by reducing the problem to a subproblem. Note that the first t nodes in σ_H induces a source subgraph of G. See also Algorithm 2, whose correctness follows from the fact that the first t nodes in σ_H form a valid prefix of σ_G.

Algorithm 2 Deterministic topological sorting.

input : a directed acyclic graph $G = (V, E)$
output : a topological ordering σ_G of G

$\sigma_G \leftarrow$ null;
while *G is not an empty graph* **do**
  compute a t-certificate H of G;
  compute a topological ordering σ_H of H;
  append the first t nodes in σ_H to σ_G;
  let G be the subgraph of G induced by the nodes not containing in σ_G;
end
return σ_G;

In what follows, we present how to obtain a sparse t-certificate H. For each node $v \in V$, define $N^-(v)$ to be the set of edges in E that connect v's in-neighbors to v. Define further that $N^-_\ell(v)$ to be any subset of $N^-(v)$ of cardinality exactly $\min\{\ell, |N^-(v)|\}$. We show in Lemma 15 that the union of $N^-_t(v)$ for all $v \in V$ is a t-certificate of $O(tn)$ edges.

▶ **Lemma 15.** *For every integer $t \geq 1$, the subgraph*

$$H = (V, F) \text{ where } F \coloneqq \bigcup_{v \in V} N^-_t(v)$$

is a t-certificate of G consisting of $O(tn)$ edges.

Proof. Let σ_H be a topological ordering of H. Let S be the set of nodes placed at the first t positions of σ_H. Let $G[S]$ be the subgraph G induced by the node set S. It suffices to show that the concatenation of any topological ordering of $G[S]$ and $G[V \setminus S]$ is a topological ordering of G.

For any node $v \in S$, $N^-_t(v) < t$ because all in-neighbors of v in H have to precede v in σ_H. Combining this observation with the definition of $N^-_t(v)$, we have that for any node $v \in S$ $N^-_t(v) = N^-(v)$. Hence, no edges crossing $V \setminus S$ and S in G can be directed from $V \setminus S$ to S. Consequently, let σ_1 (resp. σ_2) be the topological ordering of $G[S]$ (resp. $G[V \setminus S]$), the concatenation $\sigma_1 \circ \sigma_2$ will not induce any backward edge. Because $G[S]$ and $G[V \setminus S]$ both are subgraphs of G and G is acyclic, this ensures the existence of σ_1 and σ_2. ◀

The remaining to show is how to obtain $N^-_t(v)$ for all v deterministically in the turnstile model. To achieve this, we use the characteristic polynomials introduced in the study of the set reconciliation problem [19]. The reason not to use ℓ_0-samplers [15] to obtain $N^-_t(v)$, a standard building block for the computations in the dynamic streams, is that our algorithm has to be deterministic here. Unfortunately, we have no idea how to obtain $N^-_t(v)$ for all v deterministically in the turnstile stream. To get around with this issue, we note that the nodes with $|N^-_t(v)| = t$ are not involved in the computation of the first t positions in σ_H, as shown in the proof of Lemma 15. We maintain the degree of node v using $O(1)$ space to decide whether v has $N^-_t(v) = t$ or not. If $N^-_t(v) = t$, the edges incident to v does not involve the computation of the first t positions in σ_H, so there is no need to compute $N^-_t(v)$; otherwise $N^-_t(v) < t$, we use Lemma 16 to recover $N^-_t(v)$ based on the characteristic polynomials used for the set reconciliation [19].

▶ **Lemma 16.** *For every integer $t \geq 1$, for every node $v \in V$ with $|N^-_t(v)| < t$, the set $N^-_t(v)$ can be obtained deterministically in the turnstile model using $O(t)$ space.*

Proof. Define the characteristic polynomial of v to be the univariate polynomial

$$P_v(Z) = \prod_{(u,v)\in N^-(v)} (Z-u) \pmod p \text{ for some prime } p > 2n,$$

noting that the roots of $P(Z)$ comprise the set $N^-(v)$ and the degree of $P_v(v)$ corresponds to $|N^-(v)|$. Though $|N_t^-(v)| < t$ at the end of the dynamic input stream, it can be much larger than t in the middle of processing the input. Hence, to recover $N_t^-(v)$ using $O(t)$ space we cannot directly maintain $P_v(Z)$. Instead, we maintain the values of $P_v(Z)$ at t distinct points $a_1, a_2, \ldots, a_t$ outside the range $[1, n]$. That is, for each $i \in [1, t]$, while processing an edge insertion (u, v) update $P(a_i)$ with $P(a_i) \cdot (a_i - u)$ and while processing an edge deletion (u, v) update $P(a_i)$ with $P(a_i) \cdot (a_i - u)^{-1}$, i.e. the multiplicative inverse of $(a_i - u)$. Given the point-value pairs $\{(a_i, P(a_i)) : i \in [1, t]\}$, by the standard interpolation [19], if the degree of $P_v(Z)$ less than t at the end of the dynamic input stream (i.e. the given premise $|N_t^-(Z)| < t$) $P_v(Z)$ can be recovered. ◀

We are now ready to prove Theorem 3. We use $O(t)$-certificates n/t times, each requiring a single pass and using $O(tn)$ space. Set k as n/t, so our algorithm uses $O(n^2/k)$ space as claimed.

7 Applications

In this section, we present applications of our topological sorting algorithms. In Section 7.1, we develop an $O(1/\varepsilon)$-pass $O(\alpha n^{1+\varepsilon})$-space streaming algorithm for SCC decomposition, based on the chain cover construction described in Section 3. We then present an $O(k)$-pass $O((n^2 \log n)/k)$-space streaming algorithm for SCC decomposition, using the notion of the t-certificate introduced in Section 6. Given these algorithms, the results for 2-SAT in Section 7.3 follow immediately.

7.1 An $O(1/\varepsilon)$-Pass $O(\alpha n^{1+\varepsilon})$-Space SCC decompisition Algorithm

In this subsection, we focus on the problem of SCC decomposition. Let $G = (V, E)$ be an n-node directed graph. A strongly connected component (SCC) $C \subseteq V$ is a maximal set of nodes such that, for all $u, v \in C$, there exists a directed path from u to v. The SCC decomposition of G is typically defined as a partition $T = \{C_1, C_2, \ldots\}$ of V, where each C_i is an SCC of G.

However, in many applications, it is not only the identification of SCCs that matters, but also their topological ordering. For instance, the algorithm for finding a satisfying assignment of 2-SAT [2] relies on such an ordering.

Therefore, we define an SCC decomposition to be an ordered collection of sets $C_1, C_2, \ldots, C_\ell$, where each C_i is an SCC, $\bigcup_{i\in[\ell]} C_i = V$, and there are no edges from any node in C_j to any node in C_i for all $i < j$.

We begin with showing that the transitive closure of an n-node directed graph G (possibly cyclic) with independence number α can be represented by the transitive closure of a subgraph $H \subseteq G$ containing $O(\alpha n)$ edges. We note that Lemma 17 extends Lemma 4, which applies only to DAGs.

▶ **Lemma 17.** *Every directed graph G (possibly cyclic) with independence number α contains an $O(\alpha n)$-edge subgraph H such that G and H have the same transitive closure. Moreover, if a graph G' has the same transitive closure as G, then G' also contains an $O(\alpha n)$-edge subgraph with the same transitive closure, even if the independence number of G' exceeds α.*

Proof. By the Gallai-Milgram Theorem [8], the node set of a directed graph with independence number α can be covered by at most α node-disjoint paths. The proof of Lemma 4 can be adapted to this setting by replacing the chain cover with the above α node-disjoint paths. ◀

▶ **Lemma 18.** *Let G be an n-node directed graph with independence number α. For every $\varepsilon > 0$, there exists an $O(1/\varepsilon)$-pass $O(\alpha n^{1+\varepsilon})$-space streaming algorithm that outputs a collection of α node-disjoint directed paths whose node sets partition the node set of G.*

Proof. The algorithm is almost the same as that in Lemma 5 except that we replace the chain cover used in Lemma 5 with the path cover stated in Lemma 17. ◀

▶ **Corollary 19.** *Let G be an n-node directed graph with independence number α. For every $\varepsilon > 0$, there exists an $O(1/\varepsilon)$-pass $O(\alpha n^{1+\varepsilon})$-space streaming algorithm that outputs an $O(\alpha n)$-edge subgraph H such that G and H have the same transitive closure.*

Proof. By Lemma 18, we obtain a path cover of G consisting of α directed paths using the claimed number of passes and space. Then, in another pass over all edges of G, for each node x, we store all of its out-neighbors in a list. If the list exceeds α out-neighbors, at least one can be discarded, as established in the proof of Lemma 4. ◀

Finally, given the subgraph H, the SCC decomposition of G can be computed by computing the SCC decomposition of H. This gives an $O(1/\varepsilon)$-pass $O(\alpha n^{1+\varepsilon})$-space deterministic streaming algorithm to compute the SCC decomposition for any n-node directed graph G in the insertion-only model.

7.2 An $O(k)$-Pass $O((n^2 \log n)/k)$-Space SCC Decomposition Algorithm

In this subsection we devise an SCC decomposition algorithm based on our topological sort algorithm in Section 6. We first present an algorithm that w.h.p. identifies all SCCs that are large enough. We then demonstrate that a modified version of our topological sort algorithm can produce the SCC decomposition of G when all SCCs are small. Combining the two algorithms, we obtain an algorithm for SCC decomposition in directed graphs.

For every $v \in V$, let $R_{\text{out}}(v)$ be the set of nodes reachable from v, and $R_{\text{in}}(v)$ be the set of nodes that can reach v. The intersection $R_{\text{out}}(v) \cap R_{\text{in}}(v)$ is the SCC that includes v. We use this property to devise our algorithm. Given an integer $k \in [n]$, we sample a set S of $n \log n/k$ nodes from V uniformly at random independently. If we can compute $R_{\text{in}}(v) \cap R_{\text{out}}(v)$ for each $v \in S$, then all the SCCs containing at least k nodes are identified with high probability as shown in the following lemma.

▶ **Lemma 20.** *Let $G = (V, E)$ be an n-node directed graph. Given any $k \in [n]$, let S be a set of $\Theta(n \log n/k)$ nodes sampled uniformly at random from V. Every SCC of G that has at least k nodes contains some node in S with probability $1 - 1/n^{\Omega(1)}$.*

Proof. Let C be an SCC with at least k nodes. The probability that none of the nodes of C are sampled is at most $(1 - c \log n/k)^k \leq 1/n^c$ for some constant $c > 1$. Applying a union bound over all SCCs with at least k nodes, the probability that there exists such an SCC containing no sampled nodes is $1/n^{\Omega(1)}$. ◀

In what follows, we show how to compute $R_{\text{out}}(v) \cap R_{\text{in}}(v)$ for each $v \in S$. We perform BFS from each $v \in S$ in both forward and backward directions. Besides usual BFS steps, whenever a node $u \in S$ reaches another node $v \in S$ in the i-th forward (resp. backward)

BFS step, we merge all nodes that v can reach in $i-1$ forward (resp. backward) BFS steps to u. Formally, for all $v \in S$ and a given integer $i \in [n]$, let $R^i_{\text{out}}(v)$ be the set of nodes that v can reach in i BFS steps. $R^i_{\text{out}}(v)$ can be obtained from $R^{i-1}_{\text{out}}(v)$ within one pass. We initiate $R^i_{\text{out}}(v)$ to be $R^{i-1}_{\text{out}}(v)$. For each input edge (x, y), if $x \in R^{i-1}_{\text{out}}(v)$, then we add y to $R^i_{\text{out}}(v)$. If $y \in S$, then we also add $R^{i-1}_{\text{out}}(y)$ to $R^i_{\text{out}}(v)$. Before the algorithm starts, we initiate $R^0_{\text{out}}(v) = \{v\}$ for all $v \in S$. We define $R^i_{in}(v)$ similarly as above.

The following lemma shows that, with high probability, after k passes of forward and backward BFS from each sampled node v, the intersection $R_{\text{in}}(v) \cap R_{\text{out}}(v)$ is correctly identified for every sampled node v.

▶ **Lemma 21.** *Let $G = (V, E)$ be an n-node directed graph. Given any $k \in [n]$, let S be a set of $\Theta(n \log n/k)$ nodes sampled uniformly at random from V. Then, with probability $1 - 1/n^{\Omega(1)}$, $R_{out}(v) \cap R_{in}(v) = R^k_{out}(v) \cap R^k_{in}(v)$ for every $v \in S$.*

Proof. Consider a pair of nodes v, u where $v \in S$ and u is a node in the same SCC as v. By definition, there is a path from v to u and a path from u to v within the SCC. Let f be the length of a shortest path P from v to u. If $f \le k$, apparently $u \in R^k_{\text{out}}(v)$. If $f \ge k$, we claim that in each subpath of length $k/2$ in P, there is a node $\in S$ with probaility at least $1 - 1/n^4$. Let $v, p_1, p_2, \cdots, p_w$ be the sampled nodes on the path arranged by their distance from v. By the claim, for all $i \in [w-1]$ the distance between p_i, p_{i+1} is at most k with probability at least $1 - 1/n^3$. Therefore $p_{i+1} \in R^k_{\text{out}}(p_i)$ for all $i \in [w-1]$. Also, by the claim, $u \in R^k_{\text{out}}(p_w)$ and $p_1 \in R^k_{\text{out}}(v)$. Observe that, by our construction, for every pair of $a, b \in S$, if $b \in R^k_{\text{out}}(a)$, then $R^k_{\text{out}}(b) \subseteq R^k_{\text{out}}(a)$. Therefore $u \in R^k_{\text{out}}(v)$. By a similar proof, $u \in R^k_{\text{in}}(v)$. Combining the two, we have $u \in R^k_{\text{out}}(v) \cap R^k_{\text{in}}(v)$ with probability at least $1 - 1/n^3$. Applying the union bound, we complete the proof. It remains to prove the claim. The probability that all $k/2$ nodes in a subpath are not sampled is at most $(1 - c \log n/k)^{k/2} = 1/n^4$ for a sufficiently large constant c, as desired. ◀

The above results lead to the following algorithm for finding all large enough SCCs.

▶ **Lemma 22.** *Let $G = (V, E)$ be an n-node directed graph. Given any $k \in [n]$, there is an $O(k)$ pass $O((n^2 \log n)/k)$ space streaming algorithm that determines all SCCs of G that contain at least k nodes with probability $1 - 1/n^{\Omega(1)}$.*

Proof. Given any k, the algorithm samples $\Theta(n \log n/k)$ nodes uniformly at random, and then performs BFS from each node for k steps in parallel. Lemma 21 shows that the reachability sets for all sampled nodes can be obtained with probability $1 - 1/n^{\Omega(1)}$, and Lemma 20 shows that all SCCs with at least k nodes will be found with probability $1 - 1/n^{\Omega(1)}$. The algorithm uses $O(k)$ passes to simulate the k-step BFS. Performing BFS from each of the $\Theta(n \log n/k)$ sampled nodes requires $O(n)$ space. Therefore, the space usage of the algorithm is $O((n^2 \log n)/k)$. ◀

It is unclear whether the algorithm described in Lemma 22 is optimal. Nevertheless, we present a one-pass lower bound by first proving a lower bound for an easier problem and then extend to it.

▶ **Theorem 23.** *Let $G = (V, E)$ be an n-node directed graph. Deciding whether G is strongly connected in one pass requires $\Omega(n^2)$ bits.*

Proof. A reduction from s, t-reachability to this problem is shown in [13] on a RAM. For completeness, we restate it here. Given an instance of s, t-reachability consisting of a directed graph $G = (V, E)$ and two designated nodes $s, t \in V$, we generate a new graph G' by adding

edges (t, i) for all $i \in V \setminus \{t\}$ and (i, s) for all $i \in V \setminus \{s\}$ to G. G' is strongly connected if and only if G has a path from s to t. Adding the $2n$ edges is straightforward in the streaming model and hence the reduction can be done within $O(n)$ space and without using extra passes. Since s, t-reachability requires $\Omega(n^2)$ bits to solve in one pass [11], the same lower bound extends to this problem. ◀

Theorem 23 immediately implies the following:

▶ **Corollary 24.** *Given any integer $k \in [n]$, identifying all SCCs of G with at least k nodes in one pass requires $\Omega(n^2)$ bits.*

▶ **Corollary 25.** *Identifying all SCCs for an n-node directed graph G in one pass requires $\Omega(n^2)$ bits.*

In the rest of the section, we show that our topological sort algorithm from Section 6 can be modified to produce the SCC decomposition of G when all SCCs in G are small. We start with the following observation.

▶ **Lemma 26.** *Let G be an n-node directed graph, and let H be a source subgraph of G. For any SCC C of G, if C contains at least one node from H, then C is entirely contained within H.*

Proof. Suppose, for the sake of contradiction, that there exists an SCC C such that $C \cap H \neq \emptyset$ but $C \not\subseteq H$. That is, there exist nodes $u \in C$ where $u \notin H$. Take any $v \in C \cap H$, by the definition of an SCC, there must be a path from u to v. This implies the existence of an edge from $G \setminus H$ to H, which contradicts the assumption that H is a source subgraph. ◀

Using Lemma 26, we can design an algorithm for SCC decomposition. The process proceeds as follows. We first identify a source subgraph H of G and compute the SCC decomposition of H. We then remove H from G and repeat this procedure until we obtain the complete SCC decomposition of G.

Here we present an efficient way to find a source subgraph. We can assume that the input graph $G = (V, E)$ is an n-node directed graph with all SCCs having at most k nodes, as larger SCCs can be identified and contracted by the above algorithm. Given any $t \geq k$, we consider a subgraph

$$H_1 = (V, F) \text{ where } F \coloneqq \bigcup_{v \in V} N^-_{2t}(v).$$

In the following lemmas, we will show that by leveraging the structure of H_1 along with the degree information of all nodes in G, we can identify a source subgraph containing at least t nodes.

▶ **Lemma 27.** *Let G be an n-node directed graph. Given $t \geq 1$, let Q be the set of nodes such that for all $v \in Q$, $|N^-(v)| \leq 2t$. Let*

$$H_1 = (V, F) \text{ where } F \coloneqq \bigcup_{v \in V} N^-_{2t}(v).$$

If H_2 is a source subgraph of H_1 and $V(H_2) \subseteq Q$, then H_2 is a source subgraph of G.

Proof. For all nodes $v \in Q$, since $|N^-(v)| \leq 2t$, we have $N^-(v) = N^-_{2t}(v)$. In other words, all nodes in Q have all their incoming edges recorded in H_1. Therefore, if $V(H_2) \subseteq Q$ has no incoming edges in H_1, it also has no incoming edges in G, which completes the proof. ◀

▶ **Lemma 28.** *Let G be an n-node directed graph with all SCCs having at most k nodes. Given any $t \in [k, n]$, let Q be the set of nodes such that for all $v \in Q$, $N^-(v) \leq 2t$. Let*

$$H_1 = (V, F) \text{ where } F \coloneqq \bigcup_{v \in V} N^-_{2t}(v).$$

There exists a subgraph H_2 with at least t nodes s.t. H_2 is a source subgraph of H_1 and $V(H_2) \subseteq Q$.

Proof. Consider any topological ordering of the SCCs $C_1, C_2, \ldots, C_\ell$ of G. Let Δ be the largest integer in $[1, \ell]$ such that the number of nodes in the union of $C_1, C_2, \ldots, C_\Delta$ is at most $2t$. Define H_2 as the subgraph induced by the nodes in the union. We will show that this choice of H_2 satisfies all the required conditions.

Note that the SCCs $C_1, C_2, \ldots, C_\Delta$ have no incoming edges from $C_{\Delta+1}, C_{k+2}, \ldots, C_\ell$. Therefore, H_2 is a source subgraph of G and of any subgraph of G that contains it. Since every node contained in C_i for $i \leq \Delta$ has at most $2t$ incoming edges, each such node belongs to Q and the subgraph induced by the nodes in the union $C_1, C_2, \ldots, C_\Delta$ is contained in H_1. Thus, H_2 is a source subgraph of H_1. In addition, the union of $C_1, C_2, \ldots, C_{\Delta+1}$ must contain more than $2t$ nodes and $C_{\Delta+1}$ has at most k nodes, it follows that the union of $C_1, C_2, \ldots, C_\Delta$ contains at least $2t - k \geq t$ nodes, as required. ◀

By Lemmas 27 and 28, we get:

▶ **Theorem 29.** *Let G be an n-node directed graph such that all SCCs have at most k nodes. For any integer $t \geq k$, there is an $O(n/t)$-pass $O(tn)$-space streaming algorithm that determines the SCC decomposition of G.*

Proof. Combining Lemma 27 and Lemma 28, we can determine the SCC decomposition for at least t nodes by keeping $O(tn)$ edges in memory in one pass. We only need to repeat n/t times to obtain the SCC decomposition of G. The space requirement is $O(tn)$ since we keep at most $O(tn)$ edges in each pass, and maintaining the SCC decomposition uses $O(n)$ space. ◀

Combining Lemma 22 and Theorem 29, we obtain an SCC decomposition algorithm.

▶ **Theorem 30.** *Let G be an n-node directed graph. For any $k \leq \sqrt{n}$, there is an $O(k)$-pass $O((n^2 \log n)/k)$-space algorithm that determines the SCC decomposition of G with probability $1 - 1/n^{\Omega(1)}$ in the insertion-only model.*

Proof. We begin by applying the algorithm from Lemma 22 to G, which identifies all SCCs containing at least k nodes. This step runs in $O(k)$ passes and uses $O((n^2 \log n)/k)$ space, succeeding with probability at least $1 - 1/n^{\Omega(1)}$. After identifying these SCCs, we contract them, resulting in a new graph G', where all SCCs contain at most k nodes.

Next, we apply the algorithm from Theorem 29 to compute the SCC decomposition of G'. We set $t = n/k$, ensuring that $t \geq \sqrt{n} \geq k$, which satisfies the conditions of Theorem 29. Substituting these parameters, the total number of passes used is $O(k + n/t) = O(k)$, and the total space required is $O((n^2 \log n)/k + tn) = O((n^2 \log n)/k)$. ◀

7.3 2-SAT

In this subsection, we focus on the problem of 2-SAT. A 2-SAT instance F is a conjunction of m clauses, $F = \bigwedge_{i \in [m]} c_i$. Each clause $c_i = (\ell_i \vee r_i)$ is a disjunction of two literals. Every ℓ_i, r_i is one of the n Boolean variables $v_1, v_2, \ldots, v_n$ or their negations $\neg v_1, \neg v_2, \ldots, \neg v_n$. A

satisfying assignment of F is an assignment of truth values for each variable v_i that makes F to be true, and we say F is satisfiable if there exists at least one satisfying assignment. The task is to determine if F is satisfiable, and if so, determine a satisfying assignment. Under streaming setting, each input of the input stream is a clause c_i, and the conjunction of all m inputs is the 2-SAT instance F under examination.

Before we show how to apply SCC decomposition to 2-SAT in streaming, we show a one pass lower bound for 2-SAT.

▶ **Theorem 31.** *Deciding if a 2SAT instance is satisfiable in one pass requires $\Omega(n^2)$ bits.*

Proof. We translate the reduction shown in [14] into streaming setting. Suppose to the contrary that we have a 2-SAT solver that could solve 2-SAT in one pass using $o(n^2)$ bits, then we have an algorithm that could solve s,t-reachability in one pass using $o(n^2)$ bits by the following reduction. Given an instance of s,t-reachability consisting of a directed graph $G=(V,E)$ and two designated nodes $s,t \in V$, for each input edge $(u,v) \in E$, generate input $(u \vee \neg v)$ for the 2-SAT solver, and for the two nodes s,t, generate input $(s \vee t), (\neg s \vee \neg t), (\neg s \vee t)$ for the 2-SAT solver. The 2SAT instance generated is unsatisfiable if and only if s can reach t in G. Since testing s,t-reachability in a directed graph in one pass requires $\Omega(n^2)$ bits [11], there cannot exist such a 2-SAT solver. ◀

Theorem 31 shows that, merely deciding if a 2-SAT instance is satisfiable in one pass requires space enough to keep the whole input. Henceforth we focus on multi-pass algorithms.

We follow the approach of [2] which reduces 2-SAT to SCC-decomposition. The reduction constructs an implication graph G from a 2-SAT instance F. The satisfiability and the truth assignment of F can be retrieved from the SCC decomposition of G. We show that, given an input stream of 2-SAT, the construction of a stream of its implication graph only double the input complexity. For each variable v and its negation $\neg v$, construct nodes $v, \neg v \in V$. For each input clause $c_i = (\ell_i \vee r_i)$, construct two edges $(\neg \ell_i, r_i)$, $(\neg r_i, \ell_i)$. The resulting implication graph G is a graph with $2n$ nodes and $2m$ edges. We can apply our SCC decomposition algorithms to G and obtain a solution to the 2-SAT instance.

▶ **Theorem 32.** *Given any $k \leq \sqrt{n}$, 2-SAT can be solved in $O(k)$-pass and $O((n^2 \log n)/k)$-space with probability $1 - 1/n^{\Omega(1)}$ in the insertion-only model.*

Let α_I denote the independence number of the implication graph of the input 2-SAT instance. We can apply our algorithm presented in Corollary 19.

▶ **Theorem 33.** *A 2-SAT instance F can be solved in $O(1/\varepsilon)$-pass $O(\alpha_I n^{1+\varepsilon})$-space in the insertion-only model.*

References

1 Akanksha Agrawal, Arindam Biswas, Édouard Bonnet, Nick Brettell, Radu Curticapean, Dániel Marx, Tillmann Miltzow, Venkatesh Raman, and Saket Saurabh. Parameterized streaming algorithms for Min-Ones d-SAT. In *39th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 150 of *LIPIcs*, pages 8:1–8:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.FSTTCS.2019.8.

2 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.

3 Béla Bollobás and Graham R. Brightwell. The dimension of random graph orders. In *The Mathematics of Paul Erdős II*, pages 47–68. Springer, 2013. doi:10.1007/978-1-4614-7254-4_5.

4 Manuel Cáceres. Minimum chain cover in almost linear time. In *50th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 261 of *LIPIcs*, pages 31:1–31:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.31.

5 Amit Chakrabarti, Prantar Ghosh, Andrew McGregor, and Sofya Vorotnikova. Vertex ordering problems in directed graph streams. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1786–1802. SIAM, 2020. doi:10.1137/1.9781611975994.109.

6 Rajesh Chitnis and Graham Cormode. Towards a theory of parameterized streaming algorithms. In *14th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 148 of *LIPIcs*, pages 7:1–7:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.IPEC.2019.7.

7 Rajesh Hemant Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1234–1251. SIAM, 2015. doi:10.1137/1.9781611973730.82.

8 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

9 R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

10 Stefan Fafianie and Stefan Kratsch. Streaming kernelization. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium (MFCS)*, volume 8635 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2014. doi:10.1007/978-3-662-44465-8_24.

11 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.

12 Alan M. Frieze. On the independence number of random graphs. *Discret. Math.*, 81(2):171–175, 1990. doi:10.1016/0012-365X(90)90149-C.

13 Neil D. Jones. Space-bounded reducibility among combinatorial problems. *J. Comput. Syst. Sci.*, 11(1):68–85, 1975. doi:10.1016/S0022-0000(75)80050-X.

14 Neil D. Jones, Y. Edmund Lien, and William T. Laaser. New problems complete for nondeterministic log space. *Math. Syst. Theory*, 10:1–17, 1976. doi:10.1007/BF01683259.

15 Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for Lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, (PODS)*, pages 49–58. ACM, 2011. doi:10.1145/1989284.1989289.

16 Shimon Kogan and Merav Parter. Faster and unified algorithms for diameter reducing shortcuts and minimum chain covers. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 212–239. SIAM, 2023. doi:10.1137/1.9781611977554.CH9.

17 Daniel Lokshtanov, Pranabendu Misra, Fahad Panolan, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Meta-theorems for parameterized streaming algorithms. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 712–739. SIAM, 2024.

18 Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Rec.*, 43(1):9–20, 2014. doi:10.1145/2627692.2627694.

19 Yaron Minsky, Ari Trachtenberg, and Richard Zippel. Set reconciliation with nearly optimal communication complexity. *IEEE Trans. Inf. Theory*, 49(9):2213–2218, 2003. doi:10.1109/TIT.2003.815784.

20 S. Muthukrishnan. Data streams: Algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2):117–236, August 2005. doi:10.1561/0400000002.

21 Jelle J. Oostveen and Erik Jan van Leeuwen. Parameterized complexity of streaming diameter and connectivity problems. *Algorithmica*, 86(9):2885–2928, 2024. doi:10.1007/S00453-024-01246-Z.

22 Warren Schudy. Finding strongly connected components in parallel using $o(\log^2 n)$ reachability queries. In *Proceedings of the 20th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 146–151. ACM, 2008. doi:10.1145/1378533.1378560.

On the Enumeration of Signatures of XOR-CNF's

Nadia Creignou
Aix-Marseille Université, CNRS, LIS, Marseille, France

Oscar Defrain
Aix-Marseille Université, CNRS, LIS, Marseille, France

Frédéric Olive
Aix-Marseille Université, CNRS, LIS, Marseille, France

Simon Vilmin
Aix-Marseille Université, CNRS, LIS, Marseille, France

Abstract

Given a CNF formula φ with clauses $C_1, \ldots, C_m$ over a set of variables V, a truth assignment $\mathbf{a}\colon V \to \{0,1\}$ generates a binary sequence $\sigma_\varphi(\mathbf{a}) = (C_1(\mathbf{a}), \ldots, C_m(\mathbf{a}))$, called a signature of φ, where $C_i(\mathbf{a}) = 1$ if clause C_i evaluates to 1 under assignment $\mathbf{a}$, and $C_i(\mathbf{a}) = 0$ otherwise. Signatures and their associated generation problems have given rise to new yet promising research questions in algorithmic enumeration. In a recent paper, Bérczi et al. interestingly proved that generating signatures of a CNF is tractable despite the fact that verifying a solution is hard. They also showed the hardness of finding maximal signatures of an arbitrary CNF due to the intractability of satisfiability in general. Their contribution leaves open the problem of efficiently generating maximal signatures for tractable classes of CNFs, i.e., those for which satisfiability can be solved in polynomial time. Stepping into that direction, we completely characterize the complexity of generating all, minimal, and maximal signatures for XOR-CNF's.

2012 ACM Subject Classification Mathematics of computing → Enumeration; Mathematics of computing → Graph enumeration

Keywords and phrases Algorithmic enumeration, XOR-CNF, signatures, maximal bipartite subgraphs enumeration, extension, proximity search

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.19

Related Version *Full Version*: `arXiv:2402.18537` [9]

Funding The authors have been supported by the ANR project PARADUAL (ANR-24-CE48-0610-01).

Acknowledgements The authors would like to thank Yasuaki Kobayashi, Kazuhiro Kurita, Kazuhisa Makino and Kunihiro Wasa for preliminary discussions on the topic of this paper.

1 Introduction

Propositional formulas are ubiquitous in computer science. The complexity of the satisfiability problem has been extensively studied from the point of view of several algorithmic tasks such as decidability, counting, or enumeration to mention but a few. Given a CNF formula φ with clauses $C_1, \ldots, C_m$ over a set of variables $V = \{v_1, \ldots, v_n\}$, a truth assignment $\mathbf{a}\colon V \to \{0,1\}$ leads to a binary sequence $\sigma_\varphi(\mathbf{a})$, called a *signature* of φ, defined by $\sigma_\varphi(\mathbf{a}) = (C_1(\mathbf{a}), \ldots, C_m(\mathbf{a}))$ where $C_i(\mathbf{a}) = 1$ if clause C_i evaluates to 1 under assignment $\mathbf{a}$, and $C_i(\mathbf{a}) = 0$ otherwise. Deciding the satisfiability of φ boils down to deciding whether the all-one sequence is a signature of φ.

In this paper, we investigate the problems of listing all, minimal, and maximal signatures of a CNF, where minimal and maximal are meant bitwise. The task of finding all signatures of a CNF originates from well-designed pattern trees and has first been posed during the

© Nadia Creignou, Oscar Defrain, Frédéric Olive, and Simon Vilmin;
licensed under Creative Commons License CC-BY 4.0

Dagstuhl Seminar on enumeration in data management [5]. However, enumerating the signatures of a given CNF – may it be all, minimal, or maximal ones – will generally produce an exponential number of solutions. Therefore, input sensitive complexity is not a suitable yardstick of efficiency when analyzing enumeration algorithms performance. Instead, *output-sensitive complexity* estimates the running time of an enumeration algorithm using both input and output sizes. In that framework, an algorithm is considered tractable when it runs in *total-polynomial time* (or *output-polynomial* time), that is, if its execution time is bounded by a polynomial in the combined size of the input and the output. Still, further constraining this notion to guarantee some regularity in the enumeration is sometimes desirable. For this reason, *polynomial delay* and *incremental-polynomial time* are customarily regarded as better notions of tractability for enumeration complexity. On the one hand, polynomial delay means that the delay between consecutive outputs is polynomial in the input size. Incremental-polynomial time, on the other hand, means that the time to produce the i^{th} solution is bounded by a polynomial in the input size plus i (see e.g., [17, 22]).

Generating all the signatures of a CNF turns out to be of great fundamental interest, for it is an example of enumeration problem which can be solved in total-polynomial time even though solutions cannot be recognized in polynomial time.[1] In a recent contribution [2], Bérczi et al. investigate this problem. They give an incremental-polynomial time algorithm for listing all the signatures of a k-CNF for fixed k, and show that for tractable CNF's, signatures can be enumerated with polynomial delay. On the other hand, they show that generating maximal signatures is hard, while minimal signatures can be listed with polynomial delay, regardless of the CNF under consideration. Their positive result relies on the equivalence with maximal independent sets enumeration in graphs, known to be tractable [23]. As of their hardness result, it relies on the fact that the existence of at least two solutions would imply the non-satisfiability of the CNF at hand. Thus, the difficulty of the enumeration heavily relies on the intractability of the satisfiability of CNF's in general. This observation naturally leads to the following open question, where by tractable CNF's we mean families of CNF's for which the satisfiability of a formula φ in the family (as well as its sub-formulas) can be decided in polynomial time.

▶ **Question 1.** *Can the maximal signatures of tractable CNF's be enumerated in total-polynomial time?*

Among tractable formulas, Horn and 2-CNF's are two natural candidates for approaching Question 1. These two cases however proved to be very challenging, and the question of their tractability was explicitly stated as an open problem in the 2022's edition of the Workshop on Enumeration Problems and Applications (WEPA'22).

Another natural step toward answering Question 1 is to consider XOR-CNF formulas, being conjunctions of clauses using the "exclusive-or" connective instead of the usual "or" connective. XOR-clauses were revealed in Schaefer's famous complexity classification as a tractable case for deciding the satisfiability of a set of generalized clauses [21]. Interestingly, XOR-clauses are the only ones that lead to a tractable case when it comes to counting the number of satisfying assignments [11]. They subsequently appeared in many Schaefer-like classifications for a variety of algorithmic tasks, including enumeration [10, 14]. The success of XOR-clauses is due to their algebraic properties. Indeed, a set of such clauses can be seen as a system of linear equations over the 2-element field. As a result, the sets

[1] These problems thus lie outside of the class EnumP which is commonly studied in algorithmic enumeration; see e.g. [22].

of satisfying assignments of such subsets are affine subspaces, and they allow for algebraic tools such as Gaussian elimination and matroid algorithms. Also in the special case of 2-XOR clauses, subsets of clauses can be interpreted as graphs with 0/1-labeled edges and benefit from several efficient graph algorithms, e.g., for computing connected components or identifying cycles [14]. This ability to capture XOR clauses using algebraic, matroid, and graph concepts and methods provides efficient tools for analyzing their behavior with respect to our enumeration problem.

Note that these formulas do not fall in the framework of CNF's though. Indeed, rewritting an XOR-CNF as a CNF may result in a blowup on the number of clauses and signatures, as discussed in Section 2. Hence, we remark that the positive results from [2] do not directly apply to XOR-CNF. Yet, we show that signatures enumeration remains tractable using flashlight search as it is done in [2]. Namely, we prove the following.

▶ **Theorem 1.** *The set of signatures of an XOR-CNF formula can be enumerated with polynomial delay and polynomial space.*

Concerning the enumeration of minimal signatures, we show that the problem is in fact equivalent to the case of listing maximal signatures.

▶ **Proposition 2.** *There is polynomial-delay and polynomial space algorithm listing all minimal signatures of an XOR-CNF formula if and only if there is one listing all maximal signatures of an XOR-CNF formula.*

This leaves us with the study of maximal signatures enumeration for XOR-CNF's. Quite surprisingly, we prove this problem to relate well with other problems from graph theory and matroid theory. Specifically, we show the case of 2-XOR-CNF to generalize maximal bipartite subgraphs enumeration, a problem which was recently shown to admit a polynomial-delay algorithm [6]. As of the general case of XOR-CNF's, it can be seen as listing all maximal satisfiable subsystems of a linear system of equations, which is related to the enumeration of circuits passing through a given element in a matroid [4]; see also [18, 19]. Relying on these results, we derive that maximal signatures enumeration for XOR-CNF's is tractable. Namely, we obtain the following.

▶ **Theorem 3.** *There is an incremental-polynomial time algorithm generating the maximal (or minimal) signatures of an XOR-CNF.*

Concerning the case of 2-XOR-CNF, we show it to be even more tractable, that is, we show that it can be solved with polynomial delay using proximity search.

▶ **Theorem 4.** *There is a polynomial-delay algorithm generating the maximal (or minimal) signatures of a 2-XOR-CNF.*

To prove Theorem 4, we represent the input 2-XOR-CNF as an edge-bicolored multigraph where each color codes the parity of the XOR clause. Then, the enumeration amounts to list maximal bipartite subgraphs with additional constraints on colored edges.

A caveat of Theorems 3 and 4 is that a queue of solutions has to be maintained in the generation. More precisely, in [4] the queue is needed to generate a new solution as the algorithm uses the saturation technique where new solutions are derived from obtained ones. In [6], the queue has the other purpose of preventing repetitions. In both cases, getting rid of a potentially exponential space use is a natural enhancement one may seek. It should however be noted that improving the algorithm from [4] to get polynomial space is open for two decades now. As of improving Theorem 4 to use polynomial space, it would imply such

a result for maximal bipartite subgraphs enumeration, another open question [6]. Toward such directions, we prove that the folklore technique of flashlight search used to obtain Theorem 1 may probably not be of great use as deciding whether a subsequence of clauses can be extended into a signature is NP-complete.

Organization of the paper. In the next section we introduce the notions that will be used in this paper and basic properties. However, due to the space constraint, the proofs of the different statements provided in this extended abstract are omitted. The interested reader may still find the arguments in the long version of the manuscript in appendix. The steps to prove the equivalence between maximal and minimal signatures enumeration for XOR-CNF and Theorem 1 are given in Section 3. The paths toward Theorems 3 and 4 are detailed in Section 4. Future directions and the limitations of flashlight search toward an improvement of Theorem 4 are finally discussed in Section 5.

2 Preliminaries

All the objects considered in this paper are finite. If V is a set, $\mathbf{2}^V$ is its powerset. Let $m \in \mathbb{N}$ and let $\sigma \in \{0,1\}^m$ be a binary sequence. We write $\sigma[i]$ to denote the value of the i^{th} element of σ. It will be furthermore convenient to note $\mathbf{1}(\sigma)$ the set of indices $I \subseteq \{1,\dots,m\}$ whose corresponding values in σ is 1; $\mathbf{0}(\sigma)$ is defined analogously. The *complementary* of σ is the binary sequence $\bar{\sigma}$ obtained from σ by flipping all the bits, i.e., $\bar{\sigma}[i] = 1 + \sigma[i] \pmod 2$ for $1 \leq i \leq m$. If $\sigma, \tau \in \{0,1\}^m$, we write $\sigma \leq \tau$ if $\sigma[i] \leq \tau[i]$ for all $1 \leq i \leq m$. In other words, $\leq$ corresponds to the bitwise order, and we write $\sigma < \tau$ if $\sigma \leq \tau$ and $\sigma \neq \tau$.

Enumeration complexity. An enumeration algorithm aims at enumerating a set of solutions of some problem, one after the other, with no repetition. We already have defined relevant time complexities in the introduction. However, space is also an important consideration in the analysis of enumeration algorithms. We assume that the solutions we produce are not stored in memory but rather flashed and discarded. Thus, when measuring space we only consider the space needed by the algorithm in order to conduct the enumeration, in terms of the input size. We refer to [17, 22] for more details on the complexity of enumeration algorithms.

Graphs. Given an undirected graph G, we write $V(G)$ its set of vertices and $E(G)$ its set of edges. For convenience, we will write uv instead of $\{u,v\}$ for edges of G. A graph G is bipartite if there exists a bipartition V_1, V_2 of $V(G)$ such that each edge of $E(G)$ has one endpoint in V_1 and the other in V_2. Equivalently, a graph G is bipartite if and only if it does not contain any odd cycles. In this paper we will consider *multigraphs* as well, meaning graphs with potentially several edges with the same endpoints, i.e., with $E(G) = \{e_1,\dots,e_m\}$ a multiset. In this case and to avoid any ambiguity we will refer to the edges of G by their label rather than their endpoints.

XOR-formulas. We assume basic familiarity with propositional logic. Let $V = \{x_1,\dots,x_n\}$ be a set of Boolean variables. A *literal* is a variable x_i (positive literal) or its negation $\neg x_i$ (negative literal). A *clause* C is a disjunction of literals. The size $|C|$ of C is its number of literals. We say that C is a k-clause if $|C| \leq k$. A formula in conjunctive normal form (or a CNF for short) φ is a conjunction of clauses, i.e., $\varphi = C_1 \wedge \cdots \wedge C_m$ where C_i is a clause for all $1 \leq i \leq m$. It will be convenient to denote by $V(\varphi)$ the variables of φ. The (total) *size* of

φ is defined as $\sum_{1\leq i\leq m} |C_i|$. An assignment of the variables in V is a mapping $\mathbf{a}\colon V \to \{0,1\}$. Given an assignment $\mathbf{a}$, we write $C_i(\mathbf{a}) = 1$ (resp. $C_i(\mathbf{a}) = 0$) if C_i evaluates to 1 (resp. 0) under assignment $\mathbf{a}$. The notations $\varphi(\mathbf{a}) = 1$ and $\varphi(\mathbf{a}) = 0$ are defined analogously. The formula φ is *satisfiable* if there exists an assignment $\mathbf{a}$ of $V(\varphi)$ such that $\varphi(\mathbf{a}) = 1$; it is unsatisfiable otherwise.

An *XOR-clause* is a clause in which the usual connective "or" is replaced by the "exclusive-or" connective. It is well-known that any XOR-clause can be represented as a linear equation $x_1 + \cdots + x_k = \varepsilon$, $\varepsilon \in \{0,1\}$ over the two-elements field $\mathbf{F}_2$. In particular, the 2-XOR-clause $x_1 + x_2 = 0$ (resp. $x_1 + x_2 = 1$) is satisfied if and only if $x_1 = x_2$ (resp. $x_1 \neq x_2$). an XOR-clause is *odd* (resp. *even*) if $\varepsilon = 1$ (resp. 0). Given an XOR-clause $C = (x_1 + \cdots + x_k = \varepsilon)$, with $\varepsilon \in \{0,1\}$, we put $\bar{C} = (x_1 + \cdots + x_k = 1 - \varepsilon)$ and call $\bar{C}$ the *negation* of C. In other words, the bar operator *changes the parity* of C. an XOR-CNF φ is a conjunction of XOR-clauses. Equivalently, φ can be seen as a system of linear equations over $\mathbf{F}_2$. For an XOR-CNF $\varphi = C_1 \wedge \cdots \wedge C_m$, we denote by $\bar{\varphi}$ the XOR-CNF $\bar{C}_1 \wedge \cdots \wedge \bar{C}_m$ and call it the *inverse* of φ. We recall that the satisfiability of an XOR-CNF can be tested in polynomial time [8, Theorem 2.18].

Signatures. The next notations and terminology are borrowed from [2]. Let $\varphi = C_1 \wedge \cdots \wedge C_m$ be a CNF. We refer to the introduction for the definition of a signature, and shall say that an assignment $\mathbf{a}$ of φ *produces* the signature $\sigma_\varphi(\mathbf{a}) = (C_1(\mathbf{a}), \ldots, C_m(\mathbf{a}))$. We will further drop the subscript φ from this notation when it is clear from the context. Note that in particular, φ is satisfiable if and only if $(1, \ldots, 1)$ is a signature of φ. A signature σ of φ is *minimal* if it is minimal with respect to $\leq$ among all signatures of φ, or in other words, if σ is a minimal element of the partial order induced by $\leq$ on the set of all signatures. Similarly, we call *maximal* a signature which is maximal for $\leq$ among signatures of φ.

In this paper we are interested in the problem of enumerating all (resp. minimal, maximal) signatures of a given XOR-CNF. Observe that in these problems, and by the previous discussions, we may indifferently consider φ as a set of XOR clauses or as a set of binary equations over $\mathbf{F}_2$. A subset of simultaneously satisfiable clauses on one side corresponds to a feasible subset of equations on the other side. In the rest of the paper, we will assume that φ is in this latter form. In particular, when φ is seen as a system of linear equations over $\mathbf{F}_2$, there is a one-to-one correspondence between maximal signatures of φ and maximal feasible subsets of equations.

Another remark is that any XOR-CNF may be rewritten as a CNF, i.e., the $\oplus$ operator may be rewritten using $\wedge$ and $\vee$. For example, the XOR-clause $x_1 \neq x_2$ can be rewritten as $(x_1 \vee x_2) \wedge (\bar{x_1} \vee \bar{x_2})$. We however note that we do not have a bijection between the sets of signatures of the two formulas. Typically, generalizing the above example to disjoint clauses will produce an XOR-CNF with one minimal signature while the equivalent CNF has exponentially many such signatures. Though this does not rule out the existence of a reduction between (maximal) signatures of XOR-CNF's to (maximal) signatures of CNF's it should be noted that the status of maximal signatures enumeration is still open for 2-CNF's, a point that is discussed in Section 5. As of the case of all signatures, it admits a simple and direct proof using flashlight search, as explained in Section 3.

We finally argue that duplicate clauses in φ may be ignored as far as the enumeration of signatures is concerned. Indeed, having one of these clauses evaluated to 1 implies that each of its copies is also evaluated to 1. Hence, we will assume without loss of generality that all formulas are without duplicated clauses in the rest of the paper, i.e., that they are defined as pairwise distinct XOR-clauses.

3 Signatures of XOR-CNF's and their properties

We prove that testing whether a binary sequence is a signature (resp. minimal signature, maximal signature) of an XOR-CNF can be done in polynomial time in the size of the CNF at hand, and use it to show that listing all signatures of an XOR-CNF is tractable. For the rest of the section, let us fix some XOR-CNF $\varphi = C_1 \wedge \cdots \wedge C_m$ on variable set $V = \{x_1, \ldots, x_n\}$. A preliminary step is the following.

▶ **Proposition 5.** *A sequence $\sigma \in \{0,1\}^m$ is a signature of φ if and only if $\overline{\sigma}$ is a signature of $\bar{\varphi}$. In particular, σ is a minimal (resp. maximal) signature of φ if and only if $\overline{\sigma}$ is a maximal (resp. minimal) signature of $\bar{\varphi}$.*

From Proposition 5 we already derive Proposition 2.
Given two disjoint subsets $A, B \subseteq \{1, \ldots, m\}$, we define

$$\varphi(A,B) := \left(\bigwedge_{i \in A} C_i\right) \wedge \left(\bigwedge_{j \in B} \bar{C}_j\right)$$

to be the formula obtained from φ by taking the clauses with index in A and the negation of clauses with index in B. We furthermore set this formula to be defined on variable set $V(\varphi)$ even though some variables in $V(\varphi)$ may not appear in any of the clauses of $\varphi(A,B)$ – we want this to ensure that assignments of $\varphi(A,B)$ are well-defined for φ. Remark that the size of $\varphi(A,B)$ is at most that of φ, and that it is defined on precisely m clauses whenever A, B is a bipartition of $\{1, \ldots, m\}$. In the next proposition, we show that testing whether a binary sequence is a signature of an XOR-CNF can be reduced to satisfiability testing.

▶ **Proposition 6.** *A sequence $\sigma \in \{0,1\}^m$ is a signature of φ if and only if the formula $\varphi(\mathbf{1}(\sigma), \mathbf{0}(\sigma))$ is satisfiable.*

Note that signatures are neither closed by subset or superset, i.e., they may exist binary words $\sigma < \tau < \rho$ such that σ, ρ are signatures but not τ. We nevertheless show that testing the minimality (or maximality) of a signature can be done locally in polynomial time.

▶ **Proposition 7.** *Deciding whether σ is a minimal (resp. maximal) signature of φ can be done in polynomial time in the size of φ.*

To conclude this section, we argue that enumerating all the signatures of an XOR-CNF is tractable using the above properties and flashlight search. We recall that the main idea of this technique is to conduct binary search over the space of solutions. In order for flashlight search to run with polynomial delay and polynomial space, it is sufficient to solve the so-called "extension problem" in polynomial time: given two disjoint subsets A, B of the ground set, decide whether there exists a solution including A and avoiding B. We refer to e.g. [3, 20] for a more detailed description of this folklore technique.

In the case of XOR-CNF signatures, A will consist of indices of clauses to be satisfied in the signature – the 1's in the sequence – while B will consist of indices of clauses not to be satisfied – the 0's in the sequence. Then, it follows from the discussion above that φ admits a signature σ satisfying $\sigma[i] = 1$ and $\sigma[j] = 0$ for all $i \in A$ and $j \in B$ if and only if $\varphi(A,B)$ is satisfiable. Indeed, a satisfying assignment $\mathbf{a}$ of $\varphi(A,B)$ satisfies the clauses of φ with index in A, and not those with index in B. Thus the signature of φ produced by $\mathbf{a}$ meets the requirements. As for the converse direction, it follows directly from the definition of a signature. We conclude by Proposition 7 that the extension problem can be solved in polynomial time for XOR-CNF signatures, hence proving Theorem 1 that we recall here.

▶ **Theorem 1.** *The set of signatures of an XOR-CNF formula can be enumerated with polynomial delay and polynomial space.*

We point out that Theorem 1 is not a direct consequence of [2, Theorem 1]. Indeed, in the latter theorem the authors consider families of tractable CNF's which do not contain XOR-CNF's in general. The arguments, however, follow the same line.

4 Maximal signatures of XOR-CNF's

We investigate the complexity of enumerating the maximal signatures of a given XOR-CNF. More precisely, we prove the general case to be solvable in incremental-polynomial time by a result of Boros et al. [4], and the 2-XOR-CNF case to be solvable with polynomial delay using proximity search.

In Section 2 we have mentioned that the clauses of a given XOR-CNF φ may be seen as linear equations $x_1 + \cdots + x_k = \varepsilon$, $\varepsilon \in \{0, 1\}$ over the two-elements field $\mathbf{F}_2$. Hence φ can be seen as a system of linear equations whose maximal feasible subsystems correspond to its maximal signatures. In [4], the authors present an incremental polynomial-time algorithm for enumerating all circuits of a matroid containing an element, and more generally, one for enumerating all maximal subsets not spanning an element. Using Farkas' lemma (more precisely the Fredholm alternative) we can derive an algorithm which, given an infeasible system of linear equations, enumerates all its maximal feasible (or minimal infeasible) subsystems within the same time; see also [18, 19] on the use of this characterization of infeasible systems. This immediately yields Theorem 3.

We now focus on 2-XOR-CNF's. Recall from Section 2 that we can consider these formulas as a set of 2-clauses of the form $x = y$ or $x \neq y$ for positive literals x, y only, together with a set of 1-clauses (z) with z a positive or negative literal. Furthermore we may assume that no clause is repeated in the formula, as far as signatures enumeration is concerned. We shall call *graphic* a 2-XOR-CNF with clauses of the first type only – that is, without 1-clauses – and will assume for now on that our XOR-CNF's are graphic. Later, we will show how to deal with 1-clauses with small modifications on the instance.

We start by defining an underlying multigraph structure of graphic 2-XOR-CNF that will be convenient for applying the framework of proximity search. Given such an XOR-CNF φ we define $G(\varphi)$ (or simply G when it is clear from the context) as the edge-bicolored multigraph defined on vertex set $V(\varphi)$ and where there is a blue edge xy whenever $x \neq y$ is a clause of φ, and a red edge xy whenever $x = y$ is a clause of φ. A same pair of variables x, y can produce both a red and a blue edge: this corresponds to having clauses $x = y$ and $x \neq y$ in φ which typically makes it unsatisfiable. However as no clause is repeated in φ, no two edges of a same color share the same endpoints. In other words, our multigraph has edge multiplicity at most two, and two edges between a same pair of endpoints means that one is blue and the other is red; see Figure 1 for an example of such a multigraph. In the following, we will label edges of G as $e_1, \ldots, e_m$ so that the edge e_j corresponds to the clause C_j in φ. We denote by $R(G)$ the set of red edges of G, and $B(G)$ the set of its blue edges. Observe that by definition, $R(G)$ and $B(G)$ define a bipartition of the edge multiset of G. In other words, it is convenient to see $R(G)$ and $B(G)$ as partitioning the *set* of indices of the edges $e_1, \ldots, e_m$ of G. In the following, when noting $E \subseteq F$ for two edge multisets of G, we mean inclusion of the indices of their edges, that is, by $E \subseteq F$ we mean $I(E) \subseteq I(F)$ for $I(F) \subseteq \{1, \ldots, m\}$ denoting the set of indices of the edges in F.

In the rest of the section, by *subgraph* of an edge-bicolored multigraph G we mean a graph H with $E(H) \subseteq E(G)$ and $V(H) \subseteq V(G)$ where $V(H)$ contains at least the vertices involved in the edges of $E(H)$. For convenience, we may as well refer to a subgraph by its

edge set, in which case we assume its vertex set to be the one induced by the endpoints of its edges. Finally, given two disjoint subsets X, Y of a graph G, by $\delta_G(X, Y)$ we mean the set of edges having an endpoint in X and the other in Y, and may drop G from the subscript if no ambiguity arises.

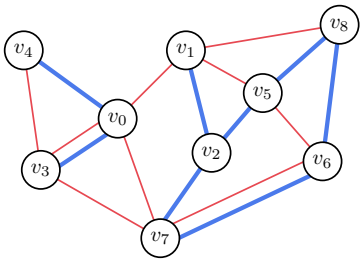


Figure 1 An edge-bicolored multigraph G on nine vertices, with red edges denoted by thin red lines and blue edges denoted by bold blue lines.

Note that an arbitrary (simple, uncolored) graph H is bipartite whenever it admits a bipartition (X, Y) of its vertex set with $E(H) = \delta(X, Y)$. In the next lemma, we relate the satisfiability of a graphic 2-XOR-CNF to a property of its underlying edge-bicolored multigraph which generalizes bipartiteness. Observe that this property is only stated on blue edges, which is still reasonable as blue and red edges partition the edges of the multigraph.

▶ **Lemma 8.** *Let φ be a graphic 2-XOR-CNF. Then φ is satisfiable if and only if there is a bipartition (X, Y) of the vertex set of $G(\varphi)$ such that $B(G) = \delta(X, Y)$. Moreover, every such bipartition corresponds to a model $\mathbf{a}$ of φ with $X = \mathbf{a}^{-1}(1)$ and $Y = \mathbf{a}^{-1}(0)$.*

We derive the following analogue for signatures, where by maximal subgraph we refer to edge inclusion.

▶ **Lemma 9.** *Let φ be a graphic 2-XOR-CNF and $\sigma \in \{0,1\}^m$. Then σ is a maximal signature of φ if and only if the subgraph $H = \{e_j : e_j \in E(G),\ \sigma[j] = 1\}$ of $G(\varphi)$ is a maximal subgraph of G admitting a bipartition (X, Y) such that $\delta_H(X, Y) = B(H)$.*

In the following, given an edge-bicolored multigraph G, we call *red-blue bipartite* any of its subgraph H admitting a bipartition (X, Y) of its vertex set such that $\delta(X, Y) = B(H)$. In Figure 2, we give an example of a maximal red-blue bipartite subgraph H of the multigraph G of Figure 1. Note that red-blue bipartite subgraphs do not contain both a red edge and a blue edge between a same pair of vertices, i.e., they define edge-bicolored graphs.

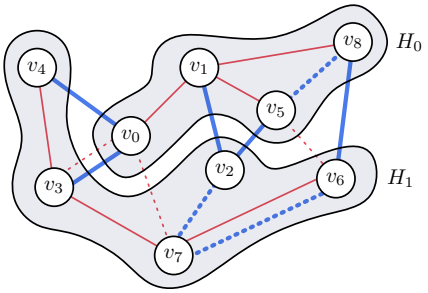


Figure 2 A maximal (connected) red-blue bipartite subgraph H of the multigraph G defined in Figure 1. The bipartition (H_0, H_1) of H is highlighted. The edges of G not in H are dotted.

▶ **Theorem 10.** *There is a polynomial-delay and polynomial-space algorithm enumerating the maximal signatures of a graphic 2-XOR-CNF φ if and only if there is one listing the maximal red-blue bipartite subgraphs of its underlying multigraph $G(\varphi)$.*

A consequence of Theorem 10 is that the enumeration of maximal signatures of a 2-XOR-CNF only consisting of disequalities $x \neq y$ is equivalent to the enumeration of maximal (edge) bipartite subgraphs of a given graph. Indeed in that case, $R(G)$ is empty – hence G is a graph – and we only require $E(G) = \delta(X, Y)$. We derive the following using a recent result by Conte and Uno [6] on maximal bipartite subgraphs enumeration by proximity search.

▶ **Corollary 11.** *There is a polynomial-delay algorithm listing the maximal signatures of a 2-XOR-CNF consisting of disequalities $x \neq y$ only.*

We end the section showing that the result from [6] can be extended into our context using proximity search, and arguing how to capture clauses of size one.

In the following, we say that an edge-bicolored multigraph G is *connected* if its underlying graph (where colors are ignored and multi-edges are considered as a single edge) is connected. Note that we can assume without loss of generality that $G(\varphi)$ is connected, as far as signatures enumeration is concerned. Indeed, if it was not the case we could enumerate the signature of each subformula of φ corresponding to connected components of $G(\varphi)$ and combine the results by doing their Cartesian product. Hence, we may further assume that $G(\varphi)$ is connected, and shall call *connected* a formula such that its underlying multigraph is. We first show that if G is connected then its maximal red-blue bipartite subgraphs are as well, yielding a canonical bipartition.

▶ **Lemma 12.** *Let G be a connected edge-bicolored multigraph. Then any of its maximal red-blue bipartite subgraphs is connected.*

▶ **Lemma 13.** *If G is a connected edge-bicolored red-blue bipartite graph then it admits a unique bipartition (X, Y) of its vertex set with $B(G) = \delta(X, Y)$ and X containing the vertex of smallest label in G.*

Proximity search has been introduced in [7] as a special case of solution graph traversal in which the proof that the solution graph is strongly connected relies on a notion of proximity between solutions that is more involved than their intersection. More precisely, the proximity between two solutions S, S^* is defined as the length of the longest prefix of S^* (according to a carefully chosen ordering of its elements) that is subset of S, and needs not to be symmetric. We refer to [6] for more details on the technique. In [6], the authors extract the necessary conditions that make an enumeration problem "proximity searchable" and thus to be solvable with polynomial delay. We restate these conditions here in the context of XOR-CNF for self-containment and better readability.

In the reformulation below, let G be an edge-bicolored multigraph and $\mathcal{H}(G)$ denote the set of maximal (connected) red-blue bipartite subgraphs of G we shall call *maximal solutions*. We further call *solutions* the connected red-blue bipartite subgraphs of G.

▶ **Theorem 14** (Reformulation of [6, Definition 4.2 and Theorem 4.3])**.** *For each $H \in \mathcal{H}(G)$, let $\mu(H)$ denote an ordering of the edges in H and, for arbitrary $H, H^* \in \mathcal{H}(G)$, let $H \tilde{\cap} H^*$ denote the elements in the longest prefix of $\mu(H^*)$ that is a subset of H. Then, there is a polynomial-delay algorithm enumerating $\mathcal{H}(G)$ whenever:*

1. *For any $H \in \mathcal{H}(G)$, any prefix of $\mu(H)$ is a solution;*
2. *There is a polynomial-time computable function* COMP*, which given a solution H' produces $H = \text{COMP}(H')$ a maximal solution such that $H \supseteq H'$; and*

3. *Given $H \in \mathcal{H}(G)$ and $e \in E(G) \setminus E(H)$ there is a family $\mathcal{K} \subseteq 2^{E(H)}$ of sets called removables such that:*
   - *the family $\mathcal{K}$ can be computed in polynomial time;*
   - *for any $K \in \mathcal{K}$, $H \setminus K \cup \{e\}$ is a solution; and*
   - *for any $H^* \in \mathcal{H}(G)$, if e is the minimal element of $\mu(H^*)$ not in H, then there exists $K \in \mathcal{K}$ with $(H \tilde{\cap} H^*) \cap K = \emptyset$.*

The rest of the section is dedicated to proving that the conditions of Theorem 14 are fulfilled for well-chosen μ and set $\mathcal{K}$. The fact that COMP can be computed in polynomial time follows from the fact that the red-blue bipartiteness is a hereditary property which can be tested in polynomial time, hence that we can greedily obtain a maximal solution.

We start with the definition of μ. Given a maximal red-blue bipartite graph H of G we denote by (H_0, H_1) the unique bipartition given by Lemma 13 where H_0 is the side containing the vertex v_0 of smallest index in G. Consider a BFS ordering of the vertices of G initiated at v_0: we start with v_0 which is unmarked, and at each step, we iteratively consider unmarked vertices ordered so far, and, in order, we mark them and append the unmarked vertices of their neighborhoods in ascending order of their label. Clearly, each prefix of such an ordering defines a connected (vertex) induced subgraph. We define $\mu(H)$ as the increasing ordering of the edges of H with respect to their endpoint occurring later in the BFS ordering launched at v_0, and break the tie by increasing order of their earlier endpoint in the BFS; see Figure 3 for an example of such an edge-ordering. This choice of an edge-ordering of a maximal solution yields the following property which will be crucial in the rest of the proof, and from which, intuitively, we will obtain that pairs of solutions H, H^* must agree on their respective bipartition of the elements in $H \tilde{\cap} H^*$.

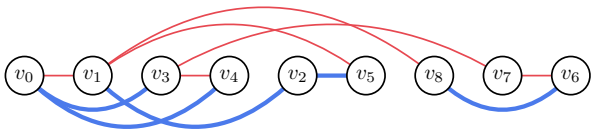


Figure 3 A BFS on the maximal red-blue bipartite subgraph H of G (Figure 2) will give the following order on the vertices: $v_0, v_1, v_3, v_4, v_2, v_5, v_8, v_7, v_6$. The order of the edges of H it produces is $\mu(H) = v_0v_1, v_0v_3, v_0v_4, v_3v_4, v_1v_2, v_1v_5, v_2v_5, v_1v_8, v_3v_7, v_8v_6, v_7v_6$.

▶ **Observation 15.** *If $H \in \mathcal{H}(G)$ then any prefix of $\mu(H)$ is a connected red-blue bipartite subgraph of G and the first edge in this ordering is incident to v_0.*

Let us now consider arbitrary $H, H^* \in \mathcal{H}(G)$ and put $H' := H \tilde{\cap} H^*$ as defined in Theorem 14. Note that if H' is empty then the trivial removable $K = E(G)$ satisfies the above conditions. Hence we may assume for convenience (in the rest of the analysis) that $|H \tilde{\cap} H^*| \neq 0$, while it will appear clear later that this assumption is in fact not needed. By Observation 15 since the first edge in $\mu(H^*)$ belongs to both H', H and H^* we derive that v_0 belongs to all three graphs. Again by Observation 15, H' is a connected red-blue bipartite subgraph of G. Thus by Lemma 13 it admits a unique bipartition (H'_0, H'_1) with blue edges lying between H'_0 and H'_1, and such that $v_0 \in H'_0$. Now since H and H^* contain H' they must agree with that bipartition, meaning that $H'_0 \subseteq H_0 \cap H^*_0$ and $H'_1 \subseteq H_1 \cap H^*_1$, where (H^*_0, H^*_1) denote the unique bipartition of H^* such that $v_0 \in H^*_0$ as provided by Lemma 13. In the following, for each edge $e = ab$ in $E(G) \setminus E(H)$, we use the edges incident to the vertices of e to build the removables. More precisely, we define $K_1 := \{av : av \in E(G)\}$ and $K_2 := \{bv : bv \in E(G)\}$ to be the two removables of (H, e) and argue that $\mathcal{K} := \{K_1, K_2\}$

meets the requirements of the theorem. We stress the fact that in K_1 or K_2 we may find edges lying in both H and H^*, which is not a problem; rather, we will argue that, when choosing e as the minimal element of $\mu(H^*)$ not in H, one of K_1 or K_2 will be disjoint from $H \tilde{\cap} H^*$, hence that $\mathcal{K}$ defines a valid set of removables.

Consider the first edge $e = ab$ in $\mu(H^*)$ that is not an edge of H. We distinguish two symmetric cases depending on whether e is red or blue, and only detail the blue case here. This situation is depicted in Figure 4.

We assume without loss of generality that the endpoint a belongs to H_0^* and that the other endpoint b lies in H_1^*. Recall that $e \notin H$. Since H is maximal, it is connected by Lemma 12, so that a, b lie in $V(H)$. Hence since e is blue, either the two endpoints a, b belong to H_0, or they both belong to H_1. Two symmetric cases arise. Let us assume $\{a, b\} \subseteq H_0$, as depicted in Figure 4. Note that in H^* we have no blue edge between b and H_1', and no red edge between b and H_0'. In H this is the opposite: we have no blue edge between b and H_0', and no red edge between b and H_1'. Now as $E(H') \subseteq E(H^*) \cap E(H)$ we conclude that b is not incident to blue or red edges in H', that is, $E(H') \cap K_2 = \emptyset$. This concludes the case and the other situation of $\{a, b\} \subseteq H_1$ yields the symmetric situation of $E(H') \cap K_1 = \emptyset$ by the same arguments. The other situation of e being red follows the exact same arguments, swapping red for blue. Hence, for one of K_1 or K_2 we have that $(H \tilde{\cap} H^*) \cap K = \emptyset$, as required by the last condition of Theorem 14.

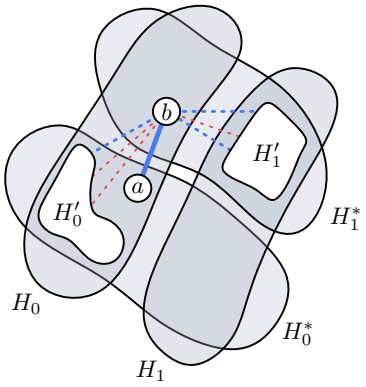


Figure 4 The situation occurring when considering two solutions H, H^* in Theorem 14 with $H' = H \tilde{\cap} H^*$: $H_0' \subseteq H_0 \cap H_0^*$, $H_1' \subseteq H_1 \cap H_1^*$ and the blue edge ab of H^* satisfies $E(H') \cap K_2 = \emptyset$ where $K_2 := \{bv : bv \in E(G)\}$.

We finish the proof observing that the graphic conditions of 2-XOR-CNF's can be relaxed. As a preliminary step, note that 1-clauses containing variables that do not appear in 2-clauses can be removed from the instance as they will be set to true in any maximal signature. Thus we may assume that φ does not contain such isolated 1-clauses. We code every other 1-clauses (x) and $(\overline{y})$ – which can also be seen as clauses $x = 1$ and $y = 0$ – by a blue edge xu and a red edge yu in the edge-bicolored multigraph G we have defined above, for some special additional vertex u that will be connecting all such 1-clauses. Note that G stays connected by the above assumption that φ has no isolated 1-clause. Then we set u to be the vertex of smallest label in G, followed by v_0. Then by the definition of $\mu(H)$ the vertex u will be placed in H_0 which can be seen as forcing the gadget to false, satisfying the clause $y = 0$ if the red edge uy is selected, and satisfying the clause $x = 1$ if the blue edge ux is selected. We conclude with Theorem 4 as a consequence of Theorem 14 and the above discussion.

5 Discussions

In this work we have showed that enumerating all (resp. minimal, maximal) signatures of an XOR-CNF formula is tractable, namely, Theorems 1, 3 and 4. Observe that the algorithm of Theorem 3 runs in incremental-polynomial time while the other two run with polynomial delay. Hence it is natural to ask whether Theorem 3 can be improved to polynomial delay.

▶ **Question 2.** *Can the maximal signatures of an XOR-CNF be listed with polynomial delay?*

The algorithm by Boros et al. [4] underlying Theorem 3 relies on the enumeration of the circuits of a matroid, for which the same question seems open since two decades. Hence, answering Question 2 would either require new techniques, or would need to answer the open question from [4] in the Boolean case. A natural first step is to handle the case of k-XOR-CNF's for fixed values of $k \geq 3$.

Furthermore, except for Theorems 1 and 3 that run with polynomial space, the algorithm of Theorem 4 requires a space that is potentially exponential as solutions must be stored into a queue. A natural question is the following.

▶ **Question 3.** *Can the maximal signatures of a 2-XOR-CNF be generated with polynomial delay and polynomial space?*

We however stress the fact that a positive answer to Question 3 would improve the algorithm by Conte et al. for maximal bipartite subgraphs enumeration for which the questions of achieving polynomial delay and space is open [6].

Toward this direction, we now argue that flashlight search may not be adapted in order to give a positive answer to Question 3, as the extension problem is hard in that case. Moreover, this result even holds when restricted to 2-XOR-CNF's consisting of disequalities, that is, in the context of maximal bipartite subgraph enumeration, which may be of independent interest.

▶ **Theorem 16.** *The problem of deciding, given a graph G and two disjoint edge sets $A, B \subseteq E(G)$, whether there exists a maximal bipartite (edge) subgraph H of G such that $A \subseteq E(H)$ and $B \cap E(H) = \emptyset$ is* NP*-complete.*

Concerning prospective research directions, let us restate here the implicit question from [2] that motivated this work, which additionally was posed at the WEPA'19 open problem session, and that is still open to date. Recall that a family of CNF's is tractable if there exists a polynomial-time algorithm to decide the satisfiability of any formula (and its subformulas) in the family.

▶ **Question 1.** *Can the maximal signatures of tractable CNF's be enumerated in total-polynomial time?*

Natural examples of tractable CNF's include 2-CNF's and Horn CNF's for which, to the best of our knowledge, no progress has been made. Toward this direction, the case of Horn 2-CNF also seems open.

Another question left open by the work of Bérczi et al. deals with the *dimension* of the formula, defined as the maximum size of a clause it contains. Namely, the algorithm of [2, Theorem 4] only performs in incremental-polynomial time for fixed dimension, and it is open whether the same result can be obtained for formulas of arbitrary dimension.

Finally, another promising direction is the parameterized study of signatures enumeration. More particularly, it can be seen that the algorithm of [2, Theorem 4] is XP-incremental parameterized by the dimension k, that is, it generates the i^{th} solution in time $(\|\varphi\| + i)^{f(k)}$

for some computable function f. It would be interesting to know whether this algorithm can be improved to run with FPT-delay, that is, to run with $f(k) \cdot \|\varphi\|^{O(1)}$ delay; see e.g. [1, 12, 13, 15, 16] for more details on the parameterized aspects of enumeration problems. A preliminary step in that direction would be to get such running times for a double parameterization. Among parameters, the maximum occurrence ω of a variable in the formula is natural. This parameter is relevant as in [2, Theorem 3] the authors give a simpler algorithm for signatures enumeration when both k and ω are bounded. Improving this XP-incremental algorithm into one that run with FPT-delay parameterized by k plus ω seems open.

References

1 Valentin Bartier, Oscar Defrain, and Fionn Mc Inerney. Hypergraph dualization with FPT-delay parameterized by the degeneracy and dimension. In *International Workshop on Combinatorial Algorithms*, pages 111–125. Springer, 2024. doi:10.1007/978-3-031-63021-7_9.

2 Kristóf Bérczi, Endre Boros, Ondřej Čepek, Khaled Elbassioni, Petr Kučera, and Kazuhisa Makino. Generating clause sequences of a CNF formula. *Theoretical computer science*, 856:68–74, 2021. doi:10.1016/J.TCS.2020.12.021.

3 Endre Boros, Khaled Elbassioni, and Vladimir Gurvich. Algorithms for generating minimal blockers of perfect matchings in bipartite graphs and related problems. In *European Symposium on Algorithms*, pages 122–133. Springer, 2004. doi:10.1007/978-3-540-30140-0_13.

4 Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Leonid Khachiyan. Algorithms for enumerating circuits in matroids. In *International Symposium on Algorithms and Computation*, pages 485–494. Springer, 2003. doi:10.1007/978-3-540-24587-2_50.

5 Endre Boros, Benny Kimelfeld, Reinhard Pichler, and Nicole Schweikardt. Enumeration in Data Management (Dagstuhl Seminar 19211). *Dagstuhl Reports*, 9(5):89–109, 2019. doi:10.4230/DagRep.9.5.89.

6 Alessio Conte, Roberto Grossi, Andrea Marino, Takeaki Uno, and Luca Versari. Proximity search for maximal subgraph enumeration. *SIAM Journal on Computing*, 51(5):1580–1625, 2022. doi:10.1137/20M1375048.

7 Alessio Conte and Takeaki Uno. New polynomial delay bounds for maximal subgraph enumeration by proximity search. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 1179–1190, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316402.

8 Yves Crama and Peter L. Hammer. *Boolean functions: Theory, algorithms, and applications*. Cambridge University Press, 2011.

9 Nadia Creignou, Oscar Defrain, Frédéric Olive, and Simon Vilmin. On the enumeration of signatures of XOR-CNF's. *CoRR*, abs/2402.18537, 2024. doi:10.48550/arXiv.2402.18537.

10 Nadia Creignou and Jean-Jacques Hébrard. On generating all solutions of generalized satisfiability problems. *RAIRO Theor. Informatics Appl.*, 31(6):499–511, 1997. doi:10.1051/ITA/1997310604991.

11 Nadia Creignou and Miki Hermann. Complexity of generalized satisfiability counting problems. *Inf. Comput.*, 125(1):1–12, 1996. doi:10.1006/INCO.1996.0016.

12 Nadia Creignou, Raïda Ktari, Arne Meier, Julian-Steffen Müller, Frédéric Olive, and Heribert Vollmer. Parameterised enumeration for modification problems. *Algorithms*, 12(9):189, 2019. doi:10.3390/A12090189.

13 Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. Paradigms for parameterized enumeration. *Theory of Computing Systems*, 60:737–758, 2017. doi:10.1007/S00224-016-9702-4.

14 Nadia Creignou, Frédéric Olive, and Johannes Schmidt. Enumerating all solutions of a boolean CSP by non-decreasing weight. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, volume 6695 of *Lecture Notes in Computer Science*, pages 120–133. Springer, 2011. doi:10.1007/978-3-642-21581-0_11.

15 Khaled Elbassioni, Matthias Hagen, and Imran Rauf. Some fixed-parameter tractable classes of hypergraph duality and related problems. In *Parameterized and Exact Computation: Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings 3*, pages 91–102. Springer, 2008. doi:10.1007/978-3-540-79723-4_10.

16 Petr A. Golovach, Christian Komusiewicz, Dieter Kratsch, and Van Bang Le. Refined notions of parameterized enumeration kernels with applications to matching cut enumeration. *Journal of Computer and System Sciences*, 123:76–102, 2022. doi:10.1016/J.JCSS.2021.07.005.

17 David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988. doi:10.1016/0020-0190(88)90065-8.

18 Leonid Khachiyan, Endre Boros, Konrad Borys, Vladimir Gurvich, and Khaled Elbassioni. Generating all vertices of a polyhedron is hard. *Twentieth Anniversary Volume: Discrete & Computational Geometry*, pages 1–17, 2009.

19 Leonid Khachiyan, Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. On the complexity of some enumeration problems for matroids. *SIAM Journal on Discrete Mathematics*, 19(4):966–984, 2005. doi:10.1137/S0895480103428338.

20 Arnaud Mary and Yann Strozecki. Efficient enumeration of solutions produced by closure operations. *Discret. Math. Theor. Comput. Sci.*, 21(3), 2019. doi:10.23638/DMTCS-21-3-22.

21 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proccedings 10th Symposium on Theory of Computing*, pages 216–226. ACM Press, 1978. doi:10.1145/800133.804350.

22 Yann Strozecki. Enumeration complexity. *Bulletin of EATCS*, 1(129), 2019.

23 Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977. doi:10.1137/0206036.

Routing Few Robots in a Crowded Network

Argyrios Deligkas
Department of Computer Science, Royal Holloway, University of London, Egham, UK

Eduard Eiben
Department of Computer Science, Royal Holloway, University of London, Egham, UK

Robert Ganian
Algorithms and Complexity Group, TU Wien, Austria

Iyad Kanj
School of Computing, DePaul University, Chicago, IL, USA

Dominik Leko
Algorithms and Complexity Group, TU Wien, IL, Austria

M. S. Ramanujan
Department of Computer Science, University of Warwick, UK

Abstract

In Graph Coordinated Motion Planning, we are given a graph G some of whose vertices are occupied by robots, and we are asked to route k marked robots to their destinations while avoiding collisions and without exceeding a given budget ℓ on the number of robot moves. We continue the recent investigation of the problem [ICALP 2024], focusing on the parameter k that captures the task of routing a small number of robots in a possibly crowded graph. We prove that the problem is W[1]-hard parameterized by ℓ even for $k = 1$, but fixed-parameter tractable parameterized by k plus the treedepth of G. We complement the latter algorithm with an NP-hardness reduction which shows that both parameters are necessary to achieve tractability.

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

Keywords and phrases graph coordinated motion planning, parameterized complexity, treedepth

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.20

Funding *Argyrios Deligkas*: UKRI EPSRC grant EP/X039862/1.
Robert Ganian: Projects No. 10.55776/Y1329 and 10.55776/COE12 of the Austrian Science Fund (FWF), Project No. ICT22-029 of the Vienna Science Foundation (WWTF).
Iyad Kanj: DePaul URC Grants 606601 and 350130.
M. S. Ramanujan: Engineering and Physical Sciences Research Council (EPSRC) grant EP/V044621/1.

1 Introduction

In many diverse settings, we are faced with the task of efficiently routing a set of marked robots through an environment while avoiding collisions (both between the marked robots themselves and with other robots that may be present). While the meaning of "efficient" is context-dependent, the two most-studied efficiency measures are the *makespan* (optimizing the amount of time) and the *energy* (optimizing the total amount of movement). In this article, we investigate the latter measure and consider the graph-theoretic setting studied in previous works [7, 8, 13, 14, 16, 17, 26]. More specifically, our article employs the problem definition from the recent ICALP paper on the topic [7][1]:

[1] See also the discussion of related work at the end of this section and the formal definitions in Section 2.

© Argyrios Deligkas, Eduard Eiben, Robert Ganian, Iyad Kanj, Dominik Leko, and M. S. Ramanujan; licensed under Creative Commons License CC-BY 4.0

Graph Coordinated Motion Planning (GCMP)

Input: A tuple $(G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), \ell)$, where G is a graph, $\mathcal{R} = \{R_i \mid i \in \mathbb{N}\}$ is a set of robots partitioned into sets $\mathcal{M}$ and $\mathcal{F}$, where each robot in $\mathcal{M}$ is given as a pair of vertices (s_i, t_i) and each robot in $\mathcal{F}$ as a single vertex s_i, and a budget $\ell \in \mathbb{N}$.

Problem: Is there a schedule for $\mathcal{R}$ of total traveled length at most ℓ?

Intuitively, each *marked* robot $R_i \in \mathcal{M}$ is provided with an origin s_i and a destination t_i, while the *free* robots in $\mathcal{F}$ only have origins but no destinations; these typically represent movable obstacles or robots without specified destinations. At each time step, we can move one[2] robot from its current position to a neighboring unoccupied vertex, and a *schedule* is a sequence of moves that delivers all marked robots to their destinations.

While GCMP is in NP [7, 32], it remains NP-hard when restricted to the instances where $|\mathcal{M}| = 1$ (which we call GCMP1) [26], or where $|V(G)| = |\mathcal{R}| + 1$ (which generalizes the $(n^2 - 1)$-puzzle) [9, 27]. The authors of the preceding work [7] investigated the parameterized complexity of GCMP with respect to the two most natural parameterizations of the problem: the total number $|\mathcal{R}|$ of robots and the budget ℓ. In particular, they showed that:

1. GCMP1 is fixed-parameter tractable w.r.t. $|\mathcal{R}|$;
2. GCMP is fixed-parameter tractable w.r.t. $|\mathcal{R}|$ plus the treewidth of G; and
3. GCMP w.r.t. ℓ is W[1]-hard (and also in XP due to a trivial brute-force algorithm), but becomes fixed-parameter tractable when the input graphs have bounded local treewidth.

In this article, we take a refined look at the problem's parameterized complexity by considering the number $k = |\mathcal{M}|$ of marked robots as the parameter. We believe this perspective to be natural, as it better captures the setting where we need to navigate a small number of marked robots through a congested environment that may contain a large number of movable obstacles or other free robots. Given the NP-hardness of GCMP1, we cannot hope for tractability when parameterizing by k alone. Our primary goal is to understand which additional restrictions (more specifically parameterizations) allow us to solve instances of GCMP with possibly many robots, but only few marked robots.

Contributions. A natural first question in this line of enquiry is whether GCMP is fixed-parameter tractable w.r.t. $k + \ell$. We note that the reduction underlying the aforementioned W[1]-hardness result for GCMP [7, Theorem 4] requires a large number of marked robots and completely breaks if we attempt to restrict $|\mathcal{M}|$. Nevertheless, as our first result, we provide a new, multi-layered reduction which strengthens the previous lower bound:

▶ **Theorem 1.** GCMP1 *is* W[1]*-hard when parameterized by* ℓ*.*

Intuitively, Theorem 1 can be seen as ruling out a uniformly polynomial algorithm for routing a single robot even if the budget is bounded by a constant.

The alternative to restricting ℓ is to aim for tractability by combining k with natural graph-structural restrictions; this is the approach which yielded the bulk of the algorithmic contributions in several recent articles on the problem and its variants [7,8,13,14,16,17]. Given the prominence of the graph parameter treewidth, a first question that arises here concerns the complexity of GCMP when parameterized by k plus the treewidth of G. Unfortunately,

[2] When minimizing energy, the considered serial motion model is equivalent to the parallel one without cyclical moves.

progress here seems difficult: the polynomial-time algorithm for GCMP1 on trees is highly non-trivial [26], and whether this result can be lifted to routing two marked robots on trees is a long-standing open question in the field. In other words, even an XP-algorithm for GCMP parameterized by k in the special case of treewidth 1 would be a breakthrough.

Instead, here we propose using a stronger restriction on G –notably, its *treedepth*. Like treewidth, treedepth is a fundamental graph parameter that has close connections to the theory of sparsity [25] and has found applications as a parameter in a number of settings, ranging from space-efficient algorithms [24] through integer programming [20], model checking [19] and graph drawing [1, 3]. As our main algorithmic contribution, we prove:

▶ **Theorem 2.** GCMP *is fixed-parameter tractable when parameterized by the number* $k = |\mathcal{M}|$ *of marked robots plus the treedepth of the input graph.*

The proof of Theorem 2 is non-trivial and does not directly follow from any of the previously developed techniques for treedepth-based algorithms on their own. For the proof, we partition instances of interest into three groups and provide different fixed-parameter algorithms for each of the three groups:

1. **Instances where the number $|V(G)| - |\mathcal{R}|$ of free vertices is small.** Here, our main contribution is a structural result showing that every YES-instance admits a schedule whose length is bounded by a function of the combination of k, the treedepth and the number of free vertices. Towards this, we show that such an instance has an optimal schedule in which at least one robot never moves from its initial position and moreover, such an "irrelevant" robot can be computed in FPT-time. We then show how this fact can be used to design a reduction rule that ultimately produces a graph of bounded size without affecting the solution size.
2. **Instances where ℓ is small.**[3] This is by far the simplest case, as it follows by directly adapting the previous algorithm for GCMP parameterized by ℓ on graphs of bounded local treewidth [7, Theorem 5].
3. **Instances where both ℓ and the number $|V(G)| - |\mathcal{R}|$ of free vertices are large.** This case is handled by providing a constructive procedure which correctly solves every instance of GCMP. The challenge here lies in the fact that if we do not outright reject the instance, then our procedure must terminate within at most ℓ steps on all graphs of bounded treedepth, including those where the walks used by the marked robots may overlap in complicated ways.

At this point, it is natural to ask whether Theorem 2 is tight in the sense of requiring both treedepth *and* k to achieve tractability. On one hand, one can exclude fixed-parameter algorithms, as well as XP-algorithms, for GCMP w.r.t. k alone due to the known NP-hardness of GCMP1. However, none of the existing lower bounds rule out such algorithms when parameterized by treedepth alone. As our final result, we close this gap by establishing:

▶ **Theorem 3.** GCMP *is* NP*-hard even when restricted to planar graphs with treedepth at most nine.*

Theorem 3 implies that neither of the two parameters in our main algorithmic result can be dropped, and is obtained via a reduction from 3D-Matching that is entirely different from the one in Theorem 1 and complements other known lower bounds for coordinated motion planning on treelike graphs [16, 17].

[3] To provide intuition, we use "small" and "large" to indicate whether or not a value is upper-bounded by a specific function of the parameter.

Further Related Work. Coordinated motion planning problems – sometimes also called multirobot pathfinding or robot routing problems – have been extensively studied by several research communities, including computational geometry [2,21,31] and artificial intelligence [4, 22, 29, 30]. The energy-minimization variant of the problem (i.e., the one considered in this article) also featured in the Third Computational Geometry Challenge at SoCG 2021 [15].

While the classical complexity of several variants of coordinated motion planning was settled already in the '90s [26], there has been a recent concentrated effort to obtain a deeper understanding of these problems via the lens of parameterized complexity. Apart from the previously mentioned work on GCMP [7], recent advances include a parameterized study of the setting with fast robots [14], fixed-parameter algorithms on solid grids [13], parameterized lower bounds for tree-like and other well-structured graphs [16, 17] and fixed-parameter approximation algorithms on trees [8]. We remark that the latter three articles primarily consider the task of makespan minimization in the parallel motion setting, which exhibits different complexity-theoretic behavior than the energy minimization setting targeted in our work. For instance, determining the existence of a schedule of constant makespan is NP-hard even when restricted to solid grid graphs [13], but schedules involving constant energy can be computed in polynomial time (and this can even be lifted to fixed-parameter tractability on graph classes of bounded local treewidth [7, Theorem 5]).

2 Preliminaries

All graphs considered in this paper are undirected and simple. We assume familiarity with standard graph-theoretic concepts and terminology [10] as well as with basic notions in parameterized complexity, including *fixed-parameter tractability* and W[1]*-hardness* [6, 11, 18]. For a subgraph H of a graph G and two vertices $u, v \in V(H)$, we denote by $\mathrm{dist}_H(u, v)$ the length of a shortest path in H between u and v. For $n \in \mathbb{N}$, we let $[n]$ and $[n]_0$ denote the sets $\{1, \ldots, n\}$ and $\{0, \ldots, n\}$, respectively.

Treewidth and Treedepth. *Treewidth* is a fundamental graph parameter which can be seen as a measure of how similar a graph is to a tree; trees have treewidth 1, while the complete n-vertex graph has treewidth $n - 1$. A formal definition of treewidth will not be necessary to obtain our results; however, we will make use of Courcelle's Theorem [5], which essentially says that problems expressible in a certain fragment of logic can be solved efficiently on graphs of bounded treewidth. Hence, we proceed by defining our other parameter of interest.

▶ **Definition 4** (Forest embedding and treedepth)**.** A *forest embedding* of a graph G is a pair (F, f), where F is a rooted forest and $f : V(G) \to V(F)$ is a bijective function, such that for each $\{u, v\} \in E(G)$, either $f(u)$ is a descendant of $f(v)$, or $f(v)$ is a descendant of $f(u)$. The *depth* of the forest embedding is the number of vertices in the longest root-to-leaf path in F. The *treedepth* of a graph G, denoted by $\mathsf{td}(G)$, is the minimum over the depths of all possible forest embeddings of G. When G is connected, F is a tree and we call it a *tree embedding*.

Below, we state three facts about treedepth which will be useful for our considerations.

▶ **Proposition 5** ([25])**.** *The treewidth of a graph G is at most its treedepth.*

▶ **Proposition 6** ([25])**.** *For a graph G and any vertex $v \in V(G)$, it holds that $\mathsf{td}(G) \leq 1 + \max_{i \in [p]} \mathsf{td}(G_i)$, where $G_1, \ldots, G_p$ are the connected components of $G - v$.*

▶ **Proposition 7** ([25])**.** *For a graph G, the maximum distance between any two vertices in G is at most $2^{\mathsf{td}(G)}$.*

Problem Definition. In our problems of interest, we are given an undirected graph G and a set $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ of k robots where $\mathcal{R}$ is partitioned into two sets $\mathcal{M}$ and $\mathcal{F}$. Each $R_i \in \mathcal{M}$, has a starting vertex s_i and a destination vertex t_i in $V(G)$ and each $R_i \in \mathcal{F}$ is associated only with a starting vertex $s_i \in V(G)$. We refer to the elements in the set $\{s_i \mid i \in [k]\} \cup \{t_i \mid R_i \in \mathcal{M}\}$ as *terminals*. The set $\mathcal{M}$ contains robots that have specific destinations they must reach, whereas $\mathcal{F}$ is the set of remaining "free" robots. We assume that all the s_i's are pairwise distinct and that all the t_i's are pairwise distinct. A vertex $v \in V(G)$ is *free* at time step $x \in [0, t]$ if no robot is located at v at time step x; otherwise, v is *occupied*. We use a discrete time frame $[0, t]$, $t \in \mathbb{N}$, to reference the sequence of moves of the robots and in each time step $x \in [0, t]$, exactly one robot moves from its current vertex to an adjacent free vertex.

A *route* for robot R_i is a tuple $W_i = (u_0, \ldots, u_t)$ of vertices in G such that (i) $u_0 = s_i$ and $u_t = t_i$ if $R_i \in \mathcal{M}$ and (ii) $\forall j \in [t]$, either $u_{j-1} = u_j$ or $u_{j-1}u_j \in E(G)$. Put simply, W_i corresponds to a "walk" in G, with the exception that consecutive vertices in W_i may be identical (representing waiting time steps), in which R_i begins at its starting vertex at time step 0, and if $R_i \in \mathcal{M}$ then R_i reaches its destination vertex at time step t. Two routes $W_i = (u_0, \ldots, u_t)$ and $W_j = (v_0, \ldots, v_t)$, where $i \neq j \in [k]$, are *non-conflicting* if $\forall r \in \{0, \ldots, t\}$, $u_r \neq v_r$. Otherwise, we say that W_i and W_j *conflict*. Intuitively, two routes conflict if the corresponding robots are at the same vertex at the end of a time step.

A *schedule* S for $\mathcal{R}$ is a set of pairwise non-conflicting routes $W_i, i \in [k]$, during a time interval $[0, t]$ such that at every time step $x \in [0, t-1]$, exactly one robot moves; i.e., there is $i \in [k]$ with route $W_i = (u_0, \ldots, u_t)$ such that $u_{x+1} \neq u_x$ and for every other robot $j \in [k] \setminus \{i\}$ with route $W_j = (v_0, \ldots, v_t)$, it holds $v_{x+1} = v_x$. The (*traveled*) *length* of a route (or its associated robot) within S is the number of time steps j such that $u_j \neq u_{j+1}$. The *total traveled length* of a schedule is the sum of the lengths of its routes; this value is often called the *energy* in the literature (e.g., see [15]).

We denote by GCMP1 the restriction of GCMP to instances where $|\mathcal{M}| = 1$. We remark that even though GCMP is stated as as a decision problem, all the algorithms provided in this paper are constructive and can output a corresponding schedule (when it exists).

3 The Hardness Results

In this section, we establish our lower bounds. We start by excluding algorithms for GCMP parameterized by treedepth alone:

▶ **Theorem 3.** GCMP *is* NP*-hard even when restricted to planar graphs with treedepth at most nine.*

Next, we will prove that GCMP is W[1]-hard when parameterized by the energy ℓ even when we have only one marked robot, i.e., GCMP1. Hence, our results show that in order to derive any positive results it is necessary to combine the structural parameters of the graph and the number of marked robots.

To prove the W[1]-hardness for GCMP1, we provide a parameterized reduction from the W[1]-complete problem Multicolored Clique (MCC) parameterized by the number k of colors. We describe the reduction here.

▶ **Theorem 1.** GCMP1 *is* W[1]*-hard when parameterized by* ℓ*.*

Towards a proof for Theorem 1, we construct an instance $\mathcal{I}$ of GCMP1 given an instance $\mathcal{I}'$ of MCC. Let $\mathcal{I}'$ consist of $G' = (V', E')$ and a partitioning $(W_1 = \{w_{1,1}, \ldots, w_{1,|W_1|}\}, \ldots, W_k = \{w_{k,1}, \ldots, w_{k,|W_k|}\})$ of V' into k disjoint subsets. From this we construct an instance $\mathcal{I} = (G, \mathcal{R} = (\{(s_1, t_1)\}, \mathcal{F}), \ell)$ of GCMP1 that is a YES-instance if and only if there is a clique of size k in G'. Furthermore, ℓ is bounded by a function of k.

The main idea of the construction is to force R_1, the only robot with a destination, to gradually choose k "lanes" to traverse on his path from s_1 to t_1. Every lane corresponds to exactly one vertex in V'. The construction and the precise selection of the budget force (i) the robot to choose exactly one lane (and thus vertex) of each color and (ii) the existence of an edge between any two vertices corresponding to chosen lanes. The combination of (i) and (ii) implies a k-clique in G'. For an intuitive understanding of the reduction, see Figures 1a and 2 which illustrate the full construction through a simple example.

For the remainder of this section, we call a path *empty* (resp. *full*) if it consists solely of free (resp. *occupied*) vertices. We say we *connect* two disjoint sets of vertices V_1 and V_2 with a full (or empty) path of length n, if we create a new path $P = (p_1, \ldots, p_n)$ and connect (with an edge) p_1 to the vertices in V_1 and connect p_n to the vertices in V_2. The sets V_1 and V_2 are the *endpoints* of P. If an endpoint V_i is a singleton $\{v_i^*\}$, we may directly refer to v_i^* as the *endpoint* instead. We proceed with a formalization of the gadgets used in the reduction.

Decision Component. Intuitively, the *decision component* G_D is the part of the construction where R_1 will choose the aforementioned lanes, and it is depicted in reference to Figure 1b. Let a *lane* be a full path of length $k-1$. For each color W_m, the component G_D contains an induced subgraph called a *layer* consisting of $|W_m|$ many vertex-disjoint lanes – one for each vertex in W_m.

The reason each lane has length precisely $k-1$ is that each vertex on it will be used to check adjacency between the vertex it represents in G' and a chosen vertex in one of the remaining $k-1$ colors. To facilitate this intuition, we use the following indexing for the individual vertices over the layers and lanes. Refer to Figure 1b for visual aid. For $m, n \in [k]$ with $m \neq n$ and $j \in [|W_m|]$, we define $v_{m,j,n}$ as follows:

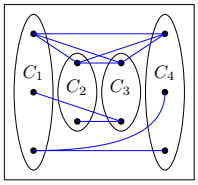
- if $m < n$, then $v_{m,j,n}$ is the $(n-1)$-st vertex in the j-th lane of the m-th layer, and
- if $m > n$, then $v_{m,j,n}$ is the n-th vertex in the j-th lane of the m-th layer.

Connect s_1 to each $\{v_{1,j,2} \mid j \in [|W_1|]\}$ with an edge. For $m \in [k-1]$, connect $\{v_{m,j,k} \mid j \in [|W_m|]\}$ and $\{v_{m+1,j,1} \mid j \in [|W_{m+1}|]\}$ with an empty path $Q_m = q_{m,1}, \ldots, q_{m,k^6}$ of length k^6. $Q = G[\{Q_m \mid m \in [k-1]\}]$ are the *tiny separators*. Let $L_m = G[\{v_{m,j,n} \mid n \in [k] \setminus \{m\}, j \in |W_m|\}]$ be the m-th *layer* and let $L = G[L_m \mid m \in [k]]$ be the graph induced on all lanes of all layers. We add two more sets of paths, the *clear paths* and the *return paths*.

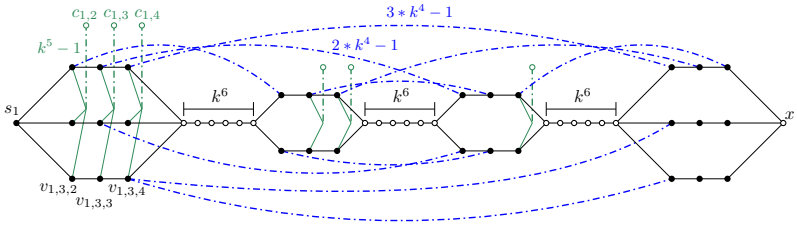
Create for every $1 \leq m < n \leq k$ a free vertex $c_{m,n}$. Then connect $\{c_{m,n}\}$ to $\{v_{m,j,n} \mid j \in |W_m|\}$ with a full path $C_{m,n}$ of length $k^5 - 1$. The set of all $C_{m,n}$ constitutes the clear paths. We define $C = \{\{c_{m,n}\} \cup C_{m,n} | 1 \leq m < n \leq k\}$.

For every edge $(w'_{m,j_1}, w'_{n,j_2}) \in E'$ with $m < n$, connect $\{v_{m,j_1,n}\}$ and $\{v_{n,j_2,m}\}$ with a full path T_{m,j_1,n,j_2} of length $k^4(n-m) - 1$. The set $T = G[\{T_{m,j_1,n,j_2} \mid (w'_{m,j_1}, w'_{n,j_2}) \in E', m < n\}]$ constitutes the return paths. Let the decision component be $G_D = G[L \cup Q \cup C \cup T]$.

Separator Component. The separator component G_S is an empty path $(x, b_2, \ldots, b_{k^{20}-1}, y)$ of length k^{20}. The first vertex x is connected to all vertices in $\{v_{k,j,k-1} \mid j \in [|W_k|]\}$. We further partition G_S into $G_{S,1} = G[\{x, b_2, \ldots, b_{k^8}\}]$ and $G_{S,2} = G[\{b_{k^8+1}, \ldots, b_{k^{20}-1}, y\}]$.



(a) The MCC instance from which to reduce.



(b) The decision component G_D plus s_1 plus the first vertex of the separator component x. The clear paths are green and the return paths are blue. Filled (resp. unfilled) vertices are occupied (resp. free) and dot dashed lines signify full paths.

Figure 1 An instance of MCC and the decision component for that instance.

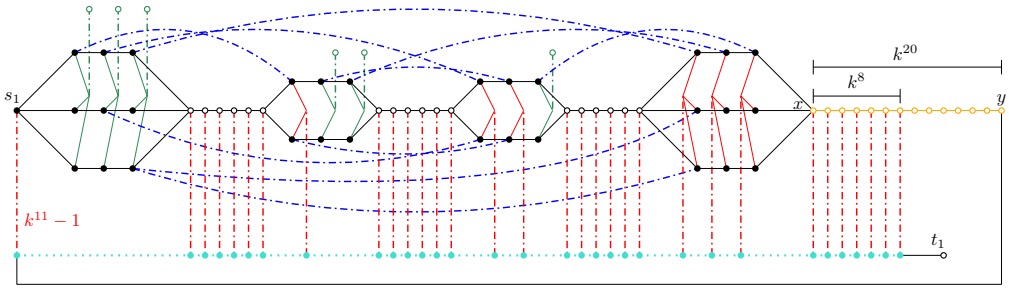


Figure 2 The full construction for the previous MCC-instance. L and Q are black, C is green, T is blue, G_S is orange and G_T is turquoise. All other edges are black. Filled (resp. unfilled) vertices are occupied (resp. free), dot dashed lines signify full paths and the dotted edges in G_T signify that those edges are each subdivided k^8 times.

Test Component. Connect $\{y\}$ and $\{t_1\}$ with a full path P of length $k^6(k-1)+\binom{k}{2}+k^8+1$. Connect vertices in P with vertices in G_D and $G_{S,1}$ with full paths consisting of $k^{11}-1$ vertices, the so-called *test paths*. To define how the test paths are connected to the remainder of the graph, partition P into the following sequence of consecutive subpaths: P_0, P_1, P'_1, P_2, P'_2, …, P'_{k-1}, P_k, P^*_k.

1. P_0 consists of a single vertex and is connected to $\{s_1\}$ with a test path.
2. For each $m \in [k]$, we set $|P_m| = m-1$ (hence P_1 is empty and only listed for uniformity) and let $P_m = p_1, \ldots, p_{m-1}$. For $n \in [m-1]$, connect $\{v_{m,j,n} \mid j \in [|W_m|]\}$ and $\{p_n\}$ with a test path.
3. For each $m \in [k-1]$, we set $|P'_m| = k^6$ and let $P'_m = p_{m,1}, \ldots, p_{m,k^6}$. For each $i \in [k^6]$, we connect $\{p_{m_i}\}$ and $\{q_{m,i}\}$ with a test path.
4. We set $|P^*_k| = k^8$ and let $P^*_k = p_1, \ldots, p_{k^8}$. Connect $\{p_1\}$ and $\{x\}$ with a test path. Then for $i \in \{2, \ldots, k^8\}$, connect $\{p_i\}$ and $\{b_i\}$ with a test path.

Finally, subdivide every edge in P, k^8 times. The vertices created in the subdivisions should not hold robots. The resulting path is the test component G_T. Note that for $m \in [k], n \in [k] \setminus \{m\}, j \in [|W_m|]$, the vertex $v_{m,j,n} \in L$ is incident to a clear path if $m < n$. Otherwise it is incident to a test path. We set $\ell = \ell_{1,1} + \ell_{1,2} + \ell_2 + \ell_3 + \ell_4$, where the terms have the values shown in Table 1. This completes the description of the reduction.

Table 1 The respective parts of the budget used in the proof of Theorem 1.

Budget	Value	Use
$\ell_{1,1}$	$k(k-1)+k^6(k-1)+1$	Move R_1 from s_1 onto x.
$\ell_{1,2}$	$k^{20}+(k^8+1)(k^6(k-1)+\binom{k}{2}+k^8)+2$	Move R_1 from x onto t_1.
ℓ_2	$k^5\binom{k}{2}$	Shift over clear paths.
ℓ_3	$\frac{1}{6}k^5(k^2-1)$	Shift over return paths.
ℓ_4	$k^{11}(k^6(k-1)+\binom{k}{2}+k^8+1)$	Shift over test paths.

4 Fixed-Parameter Tractability Parameterized by Treedepth + $|\mathcal{M}|$

In this section, we establish the following theorem:

▶ **Theorem 2.** GCMP *is fixed-parameter tractable when parameterized by the number* $k = |\mathcal{M}|$ *of marked robots plus the treedepth of the input graph.*

The proof of Theorem 2 is based on a win-win argument that distinguishes whether or not the number of free vertices in the underlying graph is upper-bounded by a function of the treedepth k. We will assume henceforth that the underlying graph in the problem instance is connected; otherwise, the problem can be solved on each connected component separately.

4.1 The Case of Few Free Vertices

The main goal of this subsection is to prove the following lemma. Recall that k denotes the number of marked robots, i.e., those in $\mathcal{M}$.

▶ **Lemma 8.** *Let* $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), \ell)$ *be an instance of* GCMP *where* G *has treedepth at most* td *and there are at most* n_f *free vertices. If there is a schedule for* $\mathcal{I}$*, then there is an optimal schedule that uses energy at most* $\gamma(n_f, k, \mathsf{td})$ *for some computable function* γ*. Moreover, an optimal schedule (if it exists) can be computed in time* $\gamma(n_f, k, \mathsf{td}) \cdot n^{\mathcal{O}(1)}$*.*

In what follows, let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), \ell)$ be an instance of GCMP. We assume that robots $R_1, \ldots, R_k$ are the marked robots. Let S be a schedule for $\mathcal{I}$. A vertex set X is said to be *fully blocked* in S at time step t if at the end of this time step, every vertex in X is occupied by a robot from $\mathcal{F}$.

▶ **Definition 9** (Configurations). *A* pseudo-configuration *for* $\mathcal{I}$ *is a pair* (τ, Q) *where* $\tau = (\tau[1], \ldots, \tau[k])$ *is a tuple of vertices and* $Q \subseteq V(G)$ *is a vertex set. Let* $V(\tau) = \cup_{i \in k}\{\tau[i]\}$*. A* configuration *for* $\mathcal{I}$ *is a pseudo-configuration* (τ, Q) *where* (i) $|V(\tau)| = k$*,* (ii) $|Q| = |\mathcal{F}|$ *and* (iii) $V(\tau) \cap Q = \emptyset$*. We say that a configuration* (τ, Q) *is* the starting configuration *if for each* $i \in [k]$*,* $\tau[i]$ *is the starting vertex of* R_i *and* Q *is the set of starting vertices of the robots in* $\mathcal{F}$*. We say that a configuration* (τ, Q) *is* a destination configuration *if for each* $i \in [k]$*,* $\tau[i]$ *is the destination vertex of the robot* R_i*.*

The intuitive meaning of a configuration is that at any time step, a configuration precisely describes the positions of the robots in $\mathcal{M}$ using the tuple τ and the positions of the robots in $\mathcal{F}$ (which are essentially indistinguishable from each other) are given by the set Q. Naturally, we do not want two robots to occupy the same vertex of G, hence we require that $V(\tau) \cap Q = \emptyset$. Moreover, note that since we do not care about the destinations of the robots in $\mathcal{F}$, there could be multiple destination configurations.

▶ **Definition 10** (Moves between configurations). *We say that there is a* move *from a configuration* (τ_1, Q_1) *to a configuration* (τ_2, Q_2) *if:*

- *either* $Q_1 = Q_2$ *and* τ_1 *and* τ_2 *differ at exactly one index* $i \in [k]$ *and* $\tau_1[i]\tau_2[i] \in E(G)$*; or*
- $\tau_1 = \tau_2$ *and* $Q_1 \Delta Q_2$[4] *has exactly two vertices* u, v *and* $uv \in E(G)$*.*

In the above definition, the first condition corresponds to moving exactly one of the robots in $\mathcal{M}$ from the vertex $\tau_1[i]$ to the vertex $\tau_2[i]$ along an edge, and the second condition corresponds to moving one of the robots in $\mathcal{F}$ between the vertices u and v while the rest remain stationary.

▶ **Definition 11** (Induced configurations). *Given a schedule* S *for the instance* $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), \ell)$*, we define the configuration* induced *by* S *at each time step* s *in the natural way, that is, it is the tuple* (τ_s, Q_s) *where* $\tau_s[i]$ *is the vertex occupied by robot* $R_i \in \mathcal{M}$ *at time step* s *and the robots in* $\mathcal{F}$ *occupy exactly the vertices in* Q_s *at the same time step.*

▶ **Definition 12** (Legal sequences). *A sequence* $(\tau_0, Q_0), \ldots, (\tau_t, Q_t)$ *of configurations is called* legal *if:* (i) (τ_0, Q_0) *is the starting configuration; and* (ii) (τ_t, Q_t) *is a destination configuration; and* (iii) *for every* $i \in [t-1]_0$*, there is a move from* (τ_i, Q_i) *to* (τ_{i+1}, Q_{i+1})*.*

▶ **Observation 13.** *Suppose that a schedule* S *for the instance* $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), \ell)$ *takes* t *time steps. For each* $s \in [t]_0$*, let* (τ_s, Q_s) *denote the configuration induced by* S *at time step* s*. Then,* $(\tau_0, Q_0), \ldots, (\tau_t, Q_t)$ *is a legal sequence of configurations. Conversely, from every legal sequence of configurations* $(\tau_0, Q_0), \ldots, (\tau_t, Q_t)$*, one can obtain a schedule* S *for* $\mathcal{I}$ *where the configuration induced by* S *at time step* s *is precisely* (τ_s, Q_s)*.*

We say that a legal sequence of configurations is *optimal* if its length is one plus the number of time steps in an optimal schedule.

▶ **Definition 14.** *Consider two disjoint vertex sets* Z_1 *and* Z_2 *and a bijection* $\phi : Z_1 \to Z_2$*. The bijection* $\phi^\star : V(G) \to V(G)$ *is defined as follows. For every* $v \in V(G)$*,* (i) *if* $v \in Z_1$*, then* $\phi^\star(v) = \phi(v)$*,* (ii) *if* $v \in Z_2$*, then* $\phi^\star(v) = \phi^{-1}(v)$*, and* (iii) $\phi^\star(v) = v$*, otherwise.*

The function $\phi^\star$ above simply swaps the vertices of Z_1 with the vertices of Z_2 while keeping the remaining vertices the same. In what follows (Definition 15 to Lemma 17), fix a legal sequence of configurations $\Gamma = (\tau_0, Q_0), \ldots, (\tau_t, Q_t)$ for the instance $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), \ell)$.

▶ **Definition 15.** *Consider two disjoint vertex sets* Z_1 *and* Z_2 *and a bijection* $\phi : Z_1 \to Z_2$*. The operation of* applying the bijection ϕ to Γ from time step s onwards *involves defining a new sequence* $\hat{\Gamma} = (\hat{\tau}_0, \hat{Q}_0), \ldots, (\hat{\tau}_t, \hat{Q}_t)$ *of pseudo-configurations as follows. For every* $i < s$*, set* $(\hat{\tau}_i, \hat{Q}_i) := (\tau_i, Q_i)$*. For every* $i \geq s$*,* (i) *set* $\hat{Q}_i := \cup_{v \in Q_i} \phi^\star(v)$*; and* (ii) *for every* $j \in [k]$*, set* $\hat{\tau}_i[j] := \phi^\star(\tau_i[j])$*.*

In words, $\hat{\Gamma}$ mimics Γ exactly, except that from time step s onward, any vertex in Z_1 (respectively, Z_2) is swapped with its image (pre-image) under ϕ.

Applying the bijection ϕ to Γ does not necessarily yield a sequence of configurations in general. However, we will demonstrate that when ϕ and the sets Z_1, Z_2 are chosen carefully, the pseudo-configurations in $\hat{\Gamma}$ are in fact, configurations – this will be useful for our analysis.

[4] The symmetric difference Δ is defined as: $Q_1 \Delta Q_2 = (Q_1 \setminus Q_2) \cup (Q_2 \setminus Q_1)$.

▶ **Definition 16.** *Let $Z \subseteq V(G)$. We say that a pair of connected components C_1 and C_2 of $G - Z$ are* strongly isomorphic *with respect to Z if there is a bijection $\phi : V(C_1) \to V(C_2)$ such that ϕ is an isomorphism from C_1 to C_2, and moreover, for every $u \in V(C_1)$ and $v \in Z$, $uv \in E(G)$ if and only if $\phi(u)v \in E(G)$. We drop the explicit reference to Z if it is clear from the context. We also say that ϕ is a* witness *for C_1 and C_2 being strongly isomorphic.*

▶ **Lemma 17.** *Let $Z \subseteq V(G)$ and consider a pair of connected components C_1 and C_2 of $G - Z$ that are disjoint from the set of terminals of $\mathcal{M}$. Suppose that C_1 and C_2 are strongly isomorphic with respect to Z, witnessed by ϕ. Suppose also that at time step c, $V(C_1)$ and $V(C_2)$ are fully blocked in S and both components are disjoint from the terminals of $\mathcal{M}$. Let $\hat{\Gamma} = (\hat{\tau}_0, \hat{Q}_0), \ldots, (\hat{\tau}_t, \hat{Q}_t)$ denote the sequence of pseudo-configurations obtained by applying the bijection ϕ to Γ from time step $c > 0$ onward. Then, the following hold:*

1. *The length of $\hat{\Gamma}$ is the same as that of Γ.*
2. *The sequence $\hat{\Gamma}$ is a legal sequence of configurations.*

▶ **Lemma 18.** *Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), \ell)$ be an instance of* GCMP *with at most n_f free vertices. Consider a set Z in G and a set $\mathcal{C} = \{C_1, \ldots, C_r\}$ of connected components of $G - Z$ such that:*

- *they are pairwise strongly isomorphic with respect to Z;*
- *$r > n_f + 3k + 1$; and*
- *C_r is fully blocked at time step 0 and disjoint from the terminals of $\mathcal{M}$.*

If there is a schedule for $\mathcal{I}$, then there is an optimal schedule in which $V(C_r)$ is fully blocked at every time step.

The most important consequence of Lemma 18 can be easily summarized in the following, which enables us to remove an "irrelevant" vertex without affecting an optimal schedule.

▶ **Corollary 19.** *Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), \ell)$ be an instance of* GCMP*. Suppose there is a vertex v that remains occupied at every time step of an optimal schedule by a robot R. Then the energy used by an optimal schedule for $\mathcal{I}' = (G - v, \mathcal{R} = (\mathcal{M} \setminus \{R\}, \mathcal{F} \setminus \{R\}), \ell)$ is the same as the energy used by an optimal schedule for $\mathcal{I}$. Moreover, given an optimal schedule for $\mathcal{I}'$, one for $\mathcal{I}$ can be produced in polynomial time.*

We next argue that if the graph is sufficiently large, then an irrelevant vertex can be found efficiently.

▶ **Lemma 20.** *There are computable functions $\mu_1, \mu_2 : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and an algorithm that, given a connected graph G, a tree embedding (F, f) of G of depth at most d and a number $\eta \in \mathbb{N}$, runs in time $\mu_1(\eta, d) \cdot n^{\mathcal{O}(1)}$ and if G has more than $\mu_2(\eta, d)$ vertices, then it produces a set $Z \subseteq V(G)$ and a set $\mathcal{C} = \{C_1, \ldots, C_\eta\}$ of components of $G - Z$ that are pairwise strongly isomorphic with respect to Z.*

We are now ready to prove the main result of this subsection.

Proof of Lemma 8. We may assume that G is connected. Otherwise, at most k of the connected components of G can contain a robot from $\mathcal{M}$ and we simply multiply the bound obtained for a connected graph by a factor of k. Consider a tree embedding (F, f) of G of depth td. This can be computed in time $2^{\mathcal{O}(\mathsf{td}^2)} \cdot n^{\mathcal{O}(1)}$ [23, 28].

Let $r = n_f + 3k + 2$. If G has more than $\mu_2(r, \mathsf{td})$ vertices, then by Lemma 20, in time $\mu_1(r, \mathsf{td}) \cdot n^{\mathcal{O}(1)}$, we can compute a set $Z \subseteq V(G)$ and a set $\mathcal{C} = \{C_1, \ldots, C_r\}$ of components of $G - Z$ that are pairwise strongly isomorphic with respect to Z. Since r is chosen to be large enough compared to n_f and k, we may assume without loss of generality that C_r is fully blocked at time step 0 and is also disjoint from the terminals of $\mathcal{M}$.

By Lemma 18 and Corollary 19, deleting the vertices in C_r (and the robots occupying them) leads to a strictly smaller, equivalent instance. We repeat this exhaustively until we obtain an instance $\mathcal{I}' = (G', \mathcal{R}' = (\mathcal{M}', \mathcal{F}'), \ell)$ where the graph G' has at most $\mu_2(n_f + 3k + 1, \mathsf{td})$ vertices. Moreover, obtaining an optimal schedule (if one exists) for $\mathcal{I}'$ can be done by a brute-force computation on G'. Since we may assume that configurations do not repeat in an optimal schedule, the number of possible legal sequences is bounded by $\nu!$, where ν is the number of possible configurations, which is clearly bounded by a function of $|V(G')|$. ◀

4.2 The Case of Many Free Vertices

We begin by noting that if $\ell \leq g(k, \mathsf{td})$, where $g(k, \mathsf{td})$ is any computable function of k and td, then the problem is FPT. This follows by an adaptation of the proof of [7, Theorem 5]:

▶ **Lemma 21.** GCMP *is* FPT *parameterized by* $|\mathcal{M}| + \mathsf{tw}(G) + \ell$.

Since, it is well know that $\mathsf{td}(G) \leq tw(G)$, we get the following corollary.

▶ **Corollary 22.** GCMP *is* FPT *parameterized by* $|\mathcal{M}| + \mathsf{td}(G) + \ell$.

By Corollary 22 and Lemma 8, we may assume in what follows that both the budget ℓ and the number n_f of free vertices are "large". We start by establishing the following structural "subgraph-freeing" lemma:

▶ **Lemma 23.** *Let H be a connected subgraph in a connected graph G, where G has treedepth $\mathsf{td}(G)$ and contains at least $|V(H)|$ many free vertices. There exists a polynomial-time computable schedule of length at most $2^{\mathsf{td}(G)} \cdot |V(H)|$ that frees up all the vertices in H.*

We use the above lemma to resolve the special case of GCMP where $|\mathcal{M}| = 1$, that is, GCMP1.

▶ **Lemma 24.** *Given an instance $\mathcal{I}$ of* GCMP1 *with $n_f \geq 2^{\mathsf{td}(G)}$, in polynomial time we can compute a schedule for $\mathcal{I}$ with a total travel length of at most $2^{2\mathsf{td}(G)+2}$.*

Proof. Let $\mathcal{I}$ be an instance of GCMP1 whose underlying graph is G. Let R be the only robot in $\mathcal{M}$, and let s and t be its starting and destination vertices, respectively. Let P be a shortest path from s to t in G. Let C_P be the connected component of $G - s$ containing $V(P) - \{s\}$. If the number of free vertices in C_P is at least $|V(P)| - 1$, then we can apply Lemma 23 to C_P to free up all the occupied vertices in $V(P) - \{s\}$. (Note that $|V(P)| \leq 2^{\mathsf{td}(G)}$ by Proposition 7.) Afterwards, we route R from s to t. Suppose now that the number of free vertices in C_P is smaller than $|V(P)| - 1 < 2^{\mathsf{td}(G)}$. Since $n_f \geq 2^{\mathsf{td}(G)}$, there is a connected component in $G - s$ other than C_P that contains a free vertex; let v be a free vertex in $G - C_P$, and let Q be a shortest path from s to v in $G - C_P$. Note again that $|V(Q)| \leq 2^{\mathsf{td}(G)}$ by Proposition 7. Moreover, s cuts $V(Q) \setminus \{s\}$ from t. We shift all robots, including R, on $V(Q)$ by one vertex towards v. We obtain a new instance $\mathcal{I}'$ with the same underlying graph G, where the starting vertex of robot R is a neighbor s' of s. Moreover, s is a cut-vertex between s' and t, and hence this operation increases the distance between the starting vertex and the destination vertex of R. We repeat the above argument on $\mathcal{I}'$. Note that every time we are unable to free the chosen path for R, the distance between its source and destination in the current instance increases by one from the previous iteration. Since the shortest path between any two vertices in G has length at most $\min(2^{\mathsf{td}(G)}, |V(G)|)$, we repeat this operation polynomially-many times, afterwards, we can free an s-t path, and the total length traveled is at most $2^{2\mathsf{td}(G)}$. ◀

We now show how the above lemma can be employed for routing all robots with destinations, in the case where both the budget and the number of free vertices are large. Before doing that in Lemma 26, we first establish the following auxiliary result:

▶ **Lemma 25.** *Let G be a connected graph and $p \in \mathbb{N}$ such that $|V(G)| \geq 2^{\mathsf{td}^2(G)} \cdot p^{\mathsf{td}(G)}$. Then there exists a separator S of size at most $\mathsf{td}(G)$ and a set $\mathcal{C} = \{C_1, \ldots, C_p\}$ of p many connected components of $G - S$ such that, for all $i \in [p]$, it holds that $N(C_i) = S$ and $|N(C_i)| \leq 2^{\mathsf{td}^2(G)} \cdot p^{\mathsf{td}(G)-1}$. Moreover, we can compute S and $\mathcal{C}$ in* FPT *time parameterized by $p + \mathsf{td}(G)$.*

▶ **Lemma 26.** *Given an instance $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ of* GCMP *with $n_f \geq 2^{\mathsf{td}^2(G)} \cdot (3k)^{\mathsf{td}(G)}$, in* FPT*-time parameterized by $\mathsf{td} + k$ we can compute a schedule for $\mathcal{I}$ with a total travel length of at most $2^{2\mathsf{td}(G)+2} \cdot (3k + \mathsf{td}(G))$.*

Proof. Note that $|V(G)| \geq n_f \geq 2^{\mathsf{td}^2(G)} \cdot (3k)^{\mathsf{td}(G)}$, and hence by Lemma 25, we can find a set S of vertices of size at most $\mathsf{td}(G)$ and a family $\mathcal{C} = \{C_1, \ldots, C_{3k}\}$ of connected components in $G - S$ such that, for all $i \in [3k]$, it holds that $N(C_i) = S$ and $|C_i| \leq 2^{\mathsf{td}^2(G)} \cdot (3k)^{\mathsf{td}(G)-1}$. Without loss of generality, we can assume that, for all $i \in [k]$, C_i does not contain a starting vertex or a destination vertex of any robot in $\mathcal{M}$.

Our goal is to navigate, one-by-one, the robots in $\mathcal{M}$ to $N(S) \cap (C_1 \cup C_2 \cup \cdots \cup C_k)$ such that, when we navigate R_i to C_i, the robots $R_1, R_2, \ldots, R_{i-1}$ are already in components $C_1, C_2, \ldots, C_{i-1}$ and we perform the navigation in $G - (C_1 \cup C_2 \cup \cdots \cup C_{i-1})$; so none of the robots $R_1, \ldots, R_{i-1}$ move when we navigate robot R_i. Notice that the graph $G - (C_1 \cup C_2 \cup \cdots \cup C_{i-1})$ is connected, as each of the components $C_i, \ldots, C_{3k}$ provides the same connectivity as $(C_1 \cup C_2 \cup \cdots \cup C_{i-1})$ outside of $\mathcal{C}$, and that G is connected to start with. Hence, $G - (C_1 \cup C_2 \cup \cdots \cup C_{i-1})$ has at least $2^{\mathsf{td}^2(G)} \cdot (3k)^{\mathsf{td}(G)-1} \cdot (3k - i + 1) \geq 2^{\mathsf{td}(G)}$ many free vertices, and we can navigate R_i to C_i using Lemma 24 with total travel length at most $2^{2\mathsf{td}(G)+2}$ during this computation. However, there is one caveat. If during the execution of the schedule computed by Lemma 24, a robot R_j, for $j \in \{i+1, i+2, \ldots, k\}$, enters C_i, we stop the execution of the schedule at that point and swap the names of robots R_i and R_j. That is, robot R_j will be in C_i, and hence we manged to navigate a robot (that has not been navigated before) to C_i, and robot R_i will be navigated at a later point.

When all robots in $\mathcal{M}$ are in $N(S) \cap (C_1 \cup C_2 \cup \cdots \cup C_k)$, we compute a minimum Steiner tree T of $S \cup \{t_i \mid (s_i, t_i) \in \mathcal{M}\}$ in $G - (C_1 \cup C_2 \cup \cdots \cup C_k)$. We can compute a minimum Steiner tree in a graph in FPT-time parameterized by the number of terminals [12]. Moreover, since a shortest path between any two vertices in a graph of treedepth td has length at most 2^{td} and we are connecting $k + \mathsf{td}(G)$ many terminals, it is easy to see that $|V(T)| \leq (k + \mathsf{td}(G) - 1) \cdot 2^{\mathsf{td}(G)}$ (as we can connect one terminal to the remaining ones by shortest paths), which is less than $n_f - \sum_{i \in [k]} |C_i|$. We apply Lemma 23 to T in $G - (C_1 \cup C_2 \cup \cdots \cup C_k)$. This frees all vertices of T with a schedule of total travel length at most $2^{\mathsf{td}(G)} \cdot |V(T)| \leq 2^{2\mathsf{td}(G)} \cdot (k + \mathsf{td}(G) - 1)$. Finally, we navigate the robots in $\mathcal{M}$ inside T, one by one, to their destinations in the right order. This can done by choosing the robot whose destination in T is the farthest from S, and navigating it to its destination in T. Note that the treedepth is closed under taking induced subgraphs and $G[V(T)]$ has treedepth at most $\mathsf{td}(G)$ as well. Hence, during this final navigation each robot in $\mathcal{M}$ does at most $2^{\mathsf{td}(G)}$ many moves. Hence, the total length of the schedule is at most $2^{2\mathsf{td}(G)+2} \cdot k + 2^{2\mathsf{td}(G)} \cdot (k + \mathsf{td}(G) - 1) + k \cdot 2^{\mathsf{td}(G)} \leq 2^{2\mathsf{td}(G)+2} \cdot (3k + \mathsf{td}(G))$. ◀

Proof of Theorem 2. Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), \ell)$ be an instance of GCMP. If $n_f \leq 2^{\mathsf{td}^2(G)} \cdot (3k)^{\mathsf{td}(G)}$ then $\mathcal{I}$ can be solved in FPT-time by Lemma 8. Otherwise, we have $n_f > 2^{\mathsf{td}^2(G)} \cdot (3k)^{\mathsf{td}(G)}$. If $\ell < 2^{2\mathsf{td}(G)+2} \cdot (3k + \mathsf{td}(G))$ then $\mathcal{I}$ can be solved in FPT-time by Corollary 22. Finally, if both n_f and ℓ are at least $2^{\mathsf{td}^2(G)} \cdot (3k)^{\mathsf{td}(G)}$, then $\mathcal{I}$ can be solved in FPT-time by Lemma 26. It follows that $\mathcal{I}$ can be solved in FPT-time, and GCMP is FPT. ◀

5 Concluding Remarks

The algorithms and lower bounds presented in this paper provide novel insights into the complexity of GCMP, specifically targeting the natural case where we need to route a small number of robots through a complicated environment. Nevertheless, we believe it is important to conclude by highlighting the prominent gaps in our understanding of the problem's complexity which remain unresolved. Even in the special case where $|\mathcal{R}| = |\mathcal{M}|$, the fixed-parameter tractability of planar GCMP when parameterized by the number $|\mathcal{R}|$ of robots remains open; here, the planar case is interesting not only because of the many usage scenarios where planar environments occur, but also because it would represent a natural generalization of the fixed-parameter tractability of the problem on solid grids [13]. In a similar vein, one might wonder whether GCMP parameterized by $|\mathcal{M}|$ is fixed-parameter or at least XP-tractable on trees – while highly non-trivial, a natural starting point in this direction is to target a polynomial-time algorithm solving the case with two marked robots on trees, which would generalize the classical algorithm for routing a single marked robot [26].

References

1 Michael J. Bannister, Sergio Cabello, and David Eppstein. Parameterized complexity of 1-planarity. *J. Graph Algorithms Appl.*, 22(1):23–49, 2018. doi:10.7155/JGAA.00457.

2 Bahareh Banyassady, Mark de Berg, Karl Bringmann, Kevin Buchin, Henning Fernau, Dan Halperin, Irina Kostitsyna, Yoshio Okamoto, and Stijn Slot. Unlabeled multi-robot motion planning with tighter separation bounds. In *SoCG*, volume 224, pages 12:1–12:16, 2022. doi:10.4230/LIPICS.SOCG.2022.12.

3 Sujoy Bhore, Robert Ganian, Fabrizio Montecchiani, and Martin Nöllenburg. Parameterized algorithms for queue layouts. *J. Graph Algorithms Appl.*, 26(3):335–352, 2022. doi:10.7155/JGAA.00597.

4 Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Solomon Eyal Shimony. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *IJCAI*, pages 740–746, 2015. URL: http://ijcai.org/Abstract/15/110.

5 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.

6 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.

7 Argyrios Deligkas, Eduard Eiben, Robert Ganian, Iyad Kanj, and M. S. Ramanujan. Parameterized algorithms for coordinated motion planning: Minimizing energy. 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 53:1–53:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.53.

8 Argyrios Deligkas, Eduard Eiben, Robert Ganian, Iyad Kanj, and Ramanujan Sridharan. Parameterized algorithms for multiagent pathfinding on trees. In *Proceedings of the 2025 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2025*, 2025. to appear.

9 Erik D. Demaine and Mikhail Rudoy. A simple proof that the $(n^2 - 1)$-puzzle is hard. *Theoretical Computer Science*, 732:80–84, 2018.

10 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

11 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

12 Stuart E. Dreyfus and Robert A. Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971. doi:10.1002/NET.3230010302.

13 Eduard Eiben, Robert Ganian, and Iyad Kanj. The parameterized complexity of coordinated motion planning. In *SoCG*, volume 258, pages 28:1–28:16, 2023. doi:10.4230/LIPICS.SOCG.2023.28.

14 Eduard Eiben, Robert Ganian, Iyad Kanj, and Ramanujan Sridharan. A minor-testing approach for coordinated motion planning with sliding robots. In *SoCG*, 2025. doi:10.4230/LIPIcs.SoCG.2025.44.

15 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing coordinated motion plans for robot swarms: The CG: SHOP challenge 2021. *ACM Journal on Experimental Algorithmics*, 27:3.1:1–3.1:12, 2022. doi:10.1145/3532773.

16 Foivos Fioravantes, Dusan Knop, Jan Matyás Kristan, Nikolaos Melissinos, and Michal Opler. Exact algorithms and lowerbounds for multiagent path finding: Power of treelike topology. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence*, pages 17380–17388. AAAI Press, 2024. doi:10.1609/aaai.v38i16.29686.

17 Foivos Fioravantes, Dusan Knop, Jan Matyás Kristan, Nikolaos Melissinos, and Michal Opler. Exact algorithms for multiagent path finding with communication constraints on tree-like structures. In *AAAI*, 2025. to appear. doi:10.48550/arXiv.2412.08556.

18 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer, Berlin, 2006. doi:10.1007/3-540-29953-X.

19 Fedor V. Fomin, Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, and Ioan Todinca. Distributed model checking on graphs of bounded treedepth. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing, DISC 2024, October 28 to November 1, 2024, Madrid, Spain*, volume 319 of *LIPIcs*, pages 25:1–25:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.DISC.2024.25.

20 Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. *Artif. Intell.*, 257:61–71, 2018. doi:10.1016/J.ARTINT.2017.12.006.

21 Paul Liu, Jack Spalding-Jamieson, Brandon Zhang, and Da Wei Zheng. Coordinated motion planning through randomized k-Opt (CG challenge). In *SoCG*, volume 189, pages 64:1–64:8, 2021. doi:10.4230/LIPICS.SOCG.2021.64.

22 Hang Ma, Craig Tovey, Guni Sharon, TK Kumar, and Sven Koenig. Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. In *AAAI*, volume 30(1), 2016.

23 Wojciech Nadara, Michal Pilipczuk, and Marcin Smulewicz. Computing treedepth in polynomial space and linear FPT time. 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 79:1–79:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ESA.2022.79.

24 Jesper Nederlof, Michal Pilipczuk, Céline M. F. Swennenhuis, and Karol Wegrzycki. Hamiltonian cycle parameterized by treedepth in single exponential time and polynomial space. *SIAM J. Discret. Math.*, 37(3):1566–1586, 2023. doi:10.1137/22M1518943.

25 Jaroslav Nesetril 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.

26 Christos H. Papadimitriou, Prabhakar Raghavan, Madhu Sudan, and Hisao Tamaki. Motion planning on a graph (extended abstract). In *STOC*, pages 511–520, 1994. doi:10.1109/SFCS.1994.365740.

27 Daniel Ratner and Manfred Warmuth. The $(n^2 - 1)$-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, 1990.

28 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014. doi:10.1007/978-3-662-43948-7_77.

29 Pavel Surynek. An optimization variant of multi-robot path planning is intractable. In *Proceedings of the AAAI conference on artificial intelligence*, volume 24(1), pages 1261–1263, 2010. doi:10.1609/AAAI.V24I1.7767.

30 Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015. doi:10.1016/J.ARTINT.2014.11.001.

31 Hyeyun Yang and Antoine Vigneron. A simulated annealing approach to coordinated motion planning (CG challenge). In Kevin Buchin and Éric Colin de Verdière, editors, *SoCG*, volume 189, pages 65:1–65:9, 2021. doi:10.4230/LIPICS.SOCG.2021.65.

32 Jingjin Yu and Daniela Rus. Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms. In *WAFR*, volume 107 of *Springer Tracts in Advanced Robotics*, pages 729–746, 2014. doi:10.1007/978-3-319-16595-0_42.

A WSPD, Separator and Small Tree Cover for c-Packed Graphs

Lindsey Deryckere
The University of Sydney, Australia

Joachim Gudmundsson
The University of Sydney, Australia

André van Renssen
The University of Sydney, Australia

Yuan Sha
The University of Sydney, Australia

Sampson Wong
The University of Copenhagen, Denmark

Abstract

The c-packedness property, proposed in 2010, is a geometric property that captures the spatial distribution of a set of edges. Despite the recent interest in c-packedness, its utility has so far been limited to Fréchet distance problems. An open problem is whether a wider variety of algorithmic and data structure problems can be solved efficiently under the c-packedness assumption, and more specifically, on c-packed graphs.

In this paper, we prove two fundamental properties of c-packed graphs: that there exists a linear-size well-separated pair decomposition under the graph metric, and there exists a constant size balanced separator. We then apply these fundamental properties to obtain a small tree cover for the metric space and distance oracles under the shortest path metric. In particular, we obtain a tree cover of constant size, an exact distance oracle of near-linear size and an approximate distance oracle of linear size.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Well-separated pair decomposition, separator, tree cover, distance oracles, realistic graphs

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.21

Related Version *Full Version*: https://arxiv.org/abs/2505.06884

Funding This research was partially funded by the Australian Government through the Australian Research Council (project number DP240101353).
Joachim Gudmundsson: This research was partially funded by the Australian Government through the Australian Research Council (project number DP240101353).
André van Renssen: This research was partially funded by the Australian Government through the Australian Research Council (project number DP240101353).
Sampson Wong: This research was partially funded by the European Union through the Marie Skłodowska-Curie Actions Postdoctoral Fellowship (project number 101146276).

1 Introduction

The study of graphs and their properties is a cornerstone of theoretical computer science. A wide variety of graph properties have been proposed in the literature, such as planarity [34], treewidth [4] and doubling dimension [23]. By assuming these graph properties, one can often obtain better algorithmic or data structure solutions to graph theoretic problems.

© Lindsey Deryckere, Joachim Gudmundsson, André van Renssen, Yuan Sha, and Sampson Wong; licensed under Creative Commons License CC-BY 4.0

The c-packedness property [14], proposed in 2010, is a geometric property that captures the spatial distribution of the edges in a graph. A graph is c-packed if, for any positive real r and any ball of radius r, the length of the edges contained in the ball is at most $c \cdot r$. Driemel, Har-Peled and Wenk [14] introduced the c-packedness property for polygonal curves, and showed that one can compute the Fréchet distance between a pair of c-packed curves in near-linear time. In 2013, Gudmundsson and Smid [21] adapted the c-packedness definition to graphs, and proposed a Fréchet distance data structure on c-packed trees with long edges that decides if there is a path in the tree with small Fréchet distance to a query curve. In 2023, Gudmundsson, Seybold and Wong [20] generalised the result of [21] by proposing a Fréchet distance data structure for all c-packed graphs.

Despite the recent interest in c-packedness, its utility so far has been limited to Fréchet distance problems. Moreover, the fundamental properties of c-packed graphs are not well studied, thereby limiting the number of problems that can be solved efficiently on c-packed graphs. An open problem is whether c-packed graphs have applications beyond Fréchet distance problems.

1.1 Our Results

We show that c-packed graphs lie in the intersection of two important graph classes, that is, doubling metrics and bounded treewidth graphs. Using the properties of doubling metrics and bounded treewidth graphs, one can obtain, for c-packed graphs, a linear-size well-separated pair decomposition, a linear-size exact distance oracle, a linear-size approximate distance oracle, and a constant-size tree cover. However, these constructions have processing times that are either (i) randomised, (ii) depend on the spread of the c-packed metric, or (iii) are exponential in the treewidth. See Table 1, Column 3[1].

We provide the first deterministic constructions that are independent of the spread of the c-packed metric, for the aforementioned structures. Moreover, our preprocessing times and data structure sizes are polynomial in both c and ε, whereas previous constructions are not.

We summarise the main results of our paper. First, we show that any c-packed graph in $\mathbb{R}^d$ admits a well-separated pair decomposition (WSPD) of size $O((c^3/\varepsilon) \cdot n)$, where $1/\varepsilon$ is the separation constant of the WSPD. Note that we avoid the $(1/\varepsilon)^d$ factor that appears in the sizes of many other WSPD constructions [7, 25]. Then, we show that any c-packed graph in $\mathbb{R}^d$ admits an $O(c)$-size separator. We use this separator to show that c-packed graphs have $O(c)$ treewidth, and admits an exact distance oracle (EDO) of size $O(cn \log n)$. Finally, we combine our WSPD and the EDO to construct a $(1+\varepsilon)$ distortion tree cover with $O(c^{2d+2}/\varepsilon^d)$ trees. Our tree cover implies an approximate distance oracle (ADO) of size $O((c^{2d+2}/\varepsilon^d) \cdot n)$ and $O(c^{2d+2}/\varepsilon^d)$ query time. We summarise our results in Table 1.

1.2 Related work

The c-packedness property is a popular model for realistic curves. A wide range of Fréchet distance problems have been studied on c-packed curves, including map matching [11], the mean curve [27], the shortcut Fréchet distance [13], subtrajectory clustering [5, 18] and the approximate nearest neighbour data structure [12]. However, c-packed graphs are less well understood [20, 21], despite often being used as a vital stepping stone towards a related property called λ-low density, which was also introduced by Driemel and Har-Peled [13].

[1] A reviewer pointed us to the tree-like properties of the graph and the possibility of adapting the work by Chazelle [10]. However, such adaptations would be likely to incur an exponential dependency on c.

Table 1 In the table, c = the c-packedness value, ε = either an ε^{-1} separation constant or a $(1+\varepsilon)$ approximation ratio, Rand. = randomised algorithm, Δ = spread of the doubling metric, d = dimension of the Euclidean space, dd = previous results using doubling dimension, tw = previous results using treewidth.

	Previous				New		
		Preprocessing	Size	Source	Preprocessing	Size	Source
WSPD	dd	$(1/\varepsilon)^{O(\log c)} \cdot n \log(\Delta n)$	$(1/\varepsilon)^{O(\log c)} \cdot n$	[38]	$(c^3/\varepsilon) \cdot n \log n$	$(c^3/\varepsilon) \cdot n$	Thm 3
		Rand. $(1/\varepsilon)^{O(\log c)} \cdot n \log n$		[25]			
EDO	tw	$2^{O(c^3)} n$	$c^2 n$	[9]+ [15]	$c^2 n \log^2 n$	$cn \log n$	Thm 18
Tree cover	dd	$(1/\varepsilon)^{O(\log c)} \cdot n \log^2(\Delta n)$	$(1/\varepsilon)^{O(\log c)}$	[3] + [38]	$(c^{2d+6}/\varepsilon^{d+1}) \cdot n \log n$	c^{2d+2}/ε^d	Thm 5
ADO	dd	$(1/\varepsilon)^{O(\log c)} \cdot n \log^2(\Delta n)$	$(1/\varepsilon)^{O(\log c)} \cdot n$	[3] + [38]	$(c^{2d+6}/\varepsilon^{d+1}) \cdot n \log n$	$(c^{2d+2}/\varepsilon^d) \cdot n$	Thm 6

Low density graphs have been studied in map matching [6, 11]. The well-separated pair decomposition of [22] has size polynomial in λ and ε, but its disadvantage over those stated in Table 1 is that it has size $O(n \log n)$.

Well-Separated Pair Decompositions (WSPD) are used for compact representation of the quadratic distances between pairs of points in the metric. For metrics that allow for a subquadtratic size WSPD, they have therefore been used as fundamental tools to approximate solutions to a range of proximity problems that require looking at the distances between all pairs of points, such as nearest neighbour, diameter, stretch and minimum spanning tree. Not all metrics allow for a WSPD of subquadratic size, an example of which is the metric induced by a star tree with unit weight on all edges. For this metric, any WSPD requires a quadratic number of pairs. For a point set in $\mathbb{R}^d$, where d is considered a constant, Callahan and Kosaraju [7] showed that there exists a WSPD with separation factor σ of size $O(\sigma^d n)$ that can be computed in $O(n \log n + \sigma^d n)$ time. In contrast to this, we show that for c-packed graphs, the size of the WSPD is not exponential in d, while maintaining that the size is linear. For graphs with bounded doubling dimension (dim), Har-Peled and Mendel [25] designed a $O(2^{O(dim)} n \log n + n\varepsilon^{-O(dim)})$ expected time randomised algorithm to construct a WSPD of linear size with logarithmic query time. They also designed a deterministic construction which incurs a logarithmic dependency on the aspect ratio of the metric space. In the full version of this paper we show that a c-packed graph has doubling dimension $O(\log c)$. However, in contrast to previous results, our deterministic construction of the WSPD is, to the best of our knowledge, the first that does not depend on the aspect ratio of the metric space and does not incur any factors exponential in the dimension, assuming constant dimension.

Distance oracles are shortest path data structures for graphs. For planar graphs, Lipton and Tarjan [34, 35] proved the planar separator theorem, which states that any planar graph with n vertices has a balanced separator of size $\sqrt{n}$. Using the separator to build a separator hierarchy, they constructed an exact distance oracle of size $O(n\sqrt{n})$. Thorup [39] and Klein [30] independently presented $(1+\varepsilon)$-approximate distance oracles for planar graphs of size $O(n \log n)$, for any constant $\varepsilon > 0$. Subsequent works [28, 29, 33, 41] improved the preprocessing time, space, and query time. Dvořák and Norin [15] showed that if a graph admits a small size balanced separator, it also has small treewidth. Combined with our separator results this implies that c-packed graphs have treewidth $O(c)$. Chaudhuri and Zaroliagis [9] designed an exact distance oracle whose preprocessing time is single exponential in the treewidth of the graph. In contrast to these results, our algorithms do not incur any terms exponential in c.

Approximate distance oracles have been studied on non-planar graphs. Thorup and Zwick [40] constructed a $(2k-1)$-approximate distance oracle of $O(kn^{1+1/k})$ size on any general graph, and showed that the size bound is optimal under the Erdös girth conjecture. Gudmundsson, Levcopoulos, Narasimhan and Smid [19] provided a $(1+\varepsilon)$-approximate distance oracle of size $O(n \log n)$ on any t-spanner, assuming $\varepsilon, t > 0$ are constants. Gao and Zhang [16] constructed a $(1+\varepsilon)$-approximate distance oracle of size $O(n \log n)$ for any unit-disk graph, assuming $\varepsilon > 0$ is a constant.

Balanced separators of sublinear size have been found for planar graphs [34], graphs of bounded genus [17], graphs that exclude a fixed minor [1], and graphs that are the 1-skeletons of simplicial complexes in 3-dimensions [36]. They have been used as a fundamental tool in devising efficient algorithms for graphs [1, 17, 35] and in numerical analysis [32, 36]. Randomized balanced separators can be found in expected linear time for c-packed graphs [26].

Metric embeddings approximate harder metric spaces by simpler, well-structured metric spaces, such as tree metrics. A tree cover for a finite metric space is a small number of trees such that the distance between any two points in the metric is preserved with low distortion in at least one of the trees. Tree covers with $(1+\epsilon)$-distortion and a constant number of trees have been found for Euclidean metrics [2], doubling metrics [3] and planar metrics [8].

2 Preliminaries

Let $G(V, E)$ be a geometric graph in $\mathbb{R}^d$ consisting of the point set V and edge set E. In this paper, we consider graphs in a fixed d-dimensional space that satisfy c-packedness. We denote the graph distance between two nodes $u, v \in V$ as $dist_G(u, v)$, and their Euclidean distance as $dist(u, v)$. We first define a Well-Separated Pair Decomposition [7] for Euclidean point sets, followed by its counterpart for geometric graphs.

▶ **Definition 1** (Geometric Well-Separated Pair). *Let σ be a real positive number, and let A and B be two finite sets of points in $\mathbb{R}^d$. We say that A and B are* well-separated with respect to σ *if the distance between the bounding boxes C_A and C_B of A and B, respectively, is at least $\sigma \cdot \max(rad(C_A), rad(C_B))$.*

Here $rad(C_A)$ refers to the radius of C_A, which is half its diameter.

▶ **Definition 2** (Geometric Well-Separated Pair Decomposition (WSPD)). *Let S be a set of n points in $\mathbb{R}^d$, and let σ be a real positive number. A* well-separated pair decomposition *(WSPD) for S, with respect to σ, is a sequence $\{A_1, B_1\}, \ldots \{A_m, B_m\}$ of pairs of nonempty subsets of S, for some integer m, such that (i) for each i with $1 \leq i \leq m$, A_i and B_i are well separated w.r.t. σ, and (ii) for any two distinct points p and q of S, there is exactly one index i with $1 \leq i \leq m$, such that $p \in A_i$ and $q \in B_i$, or vice versa.*

The notion of well-separated pairs and WSPD can easily be extended to graphs. For the graph version we will simply replace all the geometric distances with the distances in the graph. We will refer to a WSPD in a graph G as WSPD_G.

3 Technical Overview

In Section 3.1, we describe our deterministic WSPD_G construction. In Section 3.2, we discuss our separator theorem. Finally, in Section 3.3 we show how to use the WSPD_G and the exact distance oracle to construct a $(1+\varepsilon)$-distortion tree cover of size $O(c^{2d+2}(\frac{1}{\varepsilon})^d)$ for c-packed metrics. Due to space limitation, all omitted proofs are available in the full version of the paper on ArXiv.

3.1 A Well-Separated Pair Decomposition for c-packed Graphs

Split trees [7] or quadtrees [37] are commonly used in the construction of a geometric WSPD of a point set. An essential property of these is that the maximum Euclidean distance between two points contained in the same cell is always bounded by a function of the diameter of the cell. We construct a tree that fulfills a similar purpose to the trees above but for graph distances between points. We call this new type of tree a *δ-connected tree* (δ-CT). Each cell, corresponding to a cube s, of the δ-connected tree is a δ-connected set, meaning that points contained in the cell are within a graph distance of at most $\delta \cdot diam(s)$ from one another. To construct the δ-CT, we use a bottom up approach. The leaves of the compressed quadtree are already δ-connected sets. At higher levels of the compressed quadtree, we consider the δ-connected sets of its children, and merge together pairs of previously δ-connected sets that are also a δ-connected set in the higher level. To obtain an efficient running time, we make two observations. First, when computing the δ-connected sets for a higher level, it suffices to maintain a vertex representative for each δ-connected set of the lower level. Second, to check if a pair of sets are δ-connected, it suffices to check whether their representatives are path-connected in the cube centred at the cell but with double its radius.

To upper bound the graph diameter of the δ-connected set in each cell of the δ-CT we compute the length of intersection of edges with the cell and the 3^d surrounding cells in a canonical grid. To do this efficiently, we note that not all edges intersecting with a cell can contribute to a δ-connected component inside the cell. We therefore construct a data structure that can be queried for the total length of all edges that can contribute to a δ-connected component contained in a cell.

Combining these ideas obtains the following theorem. For details refer to Section 4.2.

▶ **Theorem 3.** *Given a c-packed graph G in $\mathbb{R}^d$, for fixed d, one can construct a $WSPD_G$ with separation factor σ of size $O(c^3\sigma n)$ in $O(cn \log n + c^3\sigma n)$ time, using $O(cn)$ space.*

3.2 A Separator Theorem for c-packed Graphs

We prove that every c-packed graph admits a balanced vertex separator of size $O(c)$. We start with the ring separator of Har-Peled and Mendel [25], which states that for a point set in $\mathbb{R}^d$, one can efficiently compute a pair of balls so that $n/2\lambda^3$ of the points are inside the inner ball, and $n/2\lambda^3$ of the points are outside the outer ball, where λ is the doubling constant of $\mathbb{R}^d$. Using the ring separator, we construct a max-flow instance in a similar fashion to Gudmundsson et al. [20] to locate a cut of size $O(c)$. This cut $(1 - 1/2\lambda^3)$-separates the graph, in that it separates the graph into two components each with at most $n \cdot (1 - 1/2\lambda^3)$ points. We obtain the following theorem, for details refer to Section 5.

▶ **Theorem 4.** *Given a c-packed graph G in $\mathbb{R}^d$, where d is fixed, with n vertices, one can find a separator of size $O(c)$ that $(1 - \frac{1}{2\lambda^3})$-separates G, in $O(c^2 n)$ time.*

3.3 A Small Tree Cover for c-packed Graphs

Our approach follows that of the celebrated "Dumbbell Theorem" [2] for Euclidean metrics. For c-packed graphs, we construct a linear number of dumbbells from the WSPD_G, which is constructed using the graph distances. The dumbbells are partitioned into groups, each of which satisfies the *length-group property* and the *empty-region property* with respect to the graph distance. The c-packedness property enables us to partition the dumbbells into a small number (depending only on the packedness value c and the separation ratio σ of WSPD_G) of groups, each of which satisfies the empty-region property. The main difficulty is proving the packing lemmas required for establishing the empty-region property. A dumbbell tree, which

connects the dumbbells in a group hierarchically, is built for each group of dumbbells. The c-packedness property and the c-CT also enable us to do range searching and efficiently build the dumbbell trees. The dumbbell trees together constitute a tree cover for the c-packed metric. We obtain the following theorem. For details refer to the full version of the paper.

▶ **Theorem 5.** *Given a c-packed graph G in $\mathbb{R}^d$ of fixed d and any $0 < \varepsilon < 1$. In $O(n \log n(c^{2d+6}(\frac{1}{\varepsilon})^{d+1} + c \log n))$ time, one can construct $O(c^{2d+2}(\frac{1}{\varepsilon})^d)$ dumbbell trees, each of size $O(n)$, such that the dumbbell trees constitute a $(1+\varepsilon)$-distortion tree cover for the graph metric induced by G.*

The tree cover of Theorem 5 immediately implies a $(1+\varepsilon)$-approximate distance oracle for the c-packed metric.

▶ **Corollary 6.** *Given a c-packed graph G in $\mathbb{R}^d$ of fixed d, and any $0 < \varepsilon < 1$, one can preprocess it in $O(n \log n(c^{2d+6}(\frac{1}{\varepsilon})^{d+1} + c \log n))$ time, using $O(c^{2d+2}(\frac{1}{\varepsilon})^d n)$ space, to answer a $(1+\varepsilon)$-approximate distance query between any two vertices in G in $O(c^{2d+2}(\frac{1}{\varepsilon})^d)$ time.*

4 A Well-Separated Pair Decomposition for c-packed Graphs

Given a c-packed graph $G(V, E)$ in d dimensions, where d is fixed, and a positive constant σ, we show how to deterministically construct a linear-size WSPD_G with separation factor σ (Section 4.2). Section 4.1 introduces some important terminology. In Section 4.2 we show that one can construct such a well-separated pair decomposition in $O(n \log n + c^3 \sigma n)$ time.

4.1 Notation and Preliminaries

Given a geometric graph $G(V, E)$ and cube s in $\mathbb{R}^d$, we define the diameter of s to be the length of a diagonal and denote it as $diam(s)$. We define the radius of s to be half the diameter and denote it as $rad(s)$. We define $V(s)$ to be the subset of vertices of V within s. We will need the following definitions.

▶ **Definition 7** (δ-Connected Set)**.** *Given a geometric graph $G(V, E)$ and a cube s in $\mathbb{R}^d$. Let s^+ be the concentric cube with twice the diameter. Two vertices $u, v \in V(s)$ are* δ-connected *if there is a path between u and v that lies within s^+ and has length at most $\delta \cdot diam(s)$. We say that a set of vertices $C \subseteq V(s)$ is a δ-connected set with respect to s if all pairs of vertices in C are δ-connected, and no vertex in $V(s) \setminus C$ is δ-connected to a vertex in C.*

▶ **Definition 8** (Partition into δ-Connected Sets)**.** *Let s be a cube in $\mathbb{R}^d$. A partition $\Psi_\delta(V(s)) = \{C_1, ...C_k\}$ of $V(s)$ into δ-connected sets is a set of k disjoint subsets of $V(s)$ that satisfies the following properties:*

1. $\bigcup_{1 \le i \le k} C_i = V(s)$.
2. *For all i s.t. $1 \le i \le k$, C_i is a δ-connected set with respect to s.*

4.2 Constructing a WSPD for c-packed graphs

Split trees [7] or quadtrees [37] are commonly used in the construction of a geometric WSPD of a point set. An essential property of these is that the Euclidean distance between two points contained in the same cell is always bounded by the diameter of the cell. In order to construct a WSPD_G we construct a new type of tree, which we will refer to as a δ-connected tree (δ-CT). This tree will satisfy the similar property, but relative to *graph distances* between the points in the graph, rather than their Euclidean distances. The main difference is that cells in a δ-connected tree may overlap. We formally define this tree below.

▶ **Definition 9** (δ-Connected Tree (δ-CT)). *Given a connected geometric graph $G(V, E)$ in $\mathbb{R}^d$ and a quadtree Q of V, a δ-connected tree T of G is a rooted tree, where every node u stores a cube s, corresponding to a cell of Q, and a representative point of a δ-connected set with respect to s. The root of T stores the cube s corresponding to the cell stored in the root of Q, and every leaf contains a single point in V. In particular, the leaves contained in the subtree rooted at u contain the points in V that are represented by the point stored at u.*

Once we have a δ-CT, it remains to upper bound the graph diameter of the δ-connected set represented in each cell. With this upper bound, one can then follow a standard approach to compute a WSPD_G with separation factor σ from the δ-CT.

In Subsection 4.2.1 we show how to construct a c-CT, T, of a c-packed graph G. In Subsection 4.2.2 we then show how to upper bound the diameter of the c-connected set represented in each cell of T. In Subsection 4.2.4 we analyze the complexity of the construction.

4.2.1 Constructing a c-Connected Tree

The algorithm to compute a c-CT takes as input a c-packed graph G, as well as its corresponding compressed quadtree Q. Without loss of generality we assume that the cell representing the root of Q is a unit cube. The *level value* of a node u in Q is said to be i if the cell/cube $s(u)$ stored at u has side length 2^{-i}. The root of Q has level value 0.

We are now ready to construct a c-CT, T. Initially, every leaf in Q becomes a leaf in T, and are then deleted from Q. Assume w.l.o.g. that the number of distinct level values in the remaining compressed quadtree Q' is ℓ and let $I = \langle i_1, i_2, \ldots, i_\ell \rangle$ be the level values sorted in decreasing order. Let U_{i_j} be the set of nodes in Q' having level value i_j, $1 \leq j \leq \ell$.

Next, we iteratively process the values in I in decreasing order. For each value i_j in I we will build a graph H_{i_j} from $H_{i_{j-1}}$. Initially we set $H_{i_0}(V_{i_0}, E_{i_0}) = G$.

Set $H_{i_1} = H_{i_0}$. For each node $u \in U_{i_1}$, let $s^+(u)$ be a concentric cube of twice the diameter of $s(u)$. The algorithm picks an arbitrary point p in $V_{i_1}(s(u))$ and merges all nodes in $V_{i_1}(s(u))$ that are path-connected to p in H_{i_0} within the cube $s^+(u)$. For all edges (u, v), if both u and v have been merged, we remove the edge. If only one endpoint has been merged, we move this endpoint to p and redefine the length of the edge to be the Euclidean distance between this point and p. Finally, we remove any parallel edges. Note that all changes are made to H_{i_1}, while H_{i_0} stays unaffected. The surviving node p is the representative point in H_{i_1} of the merged set of nodes in H_{i_0}. A node is added to T storing p and its children are the nodes in T storing the points that were merged into p. The algorithm repeats this process until all the points in $V_{i_1}(s(u))$ have been merged into representative points and the process has been complete for each $u \in U_{i_1}$.

In the next iteration, we let $H_{i_2} = H_{i_1}$. The nodes in U_{i_2} are now processed using H_{i_1} as input to merge the nodes in H_{i_2}. This continues until all the level values in I have been processed. The process for iteration j of the algorithm is visualised in Figure 1.

The proof of Lemma 10 can be found in the full version of the paper.

▶ **Lemma 10.** *Given a c-packed graph $G(V, E)$ the above algorithm produces a valid c-connected tree T of $G(V, E)$.*

4.2.2 Bounding the Graph Diameter of a Cell in the c-CT

In order to construct the well-separated pairs, it is necessary to upper bound the graph diameter of the c-connected set represented in each cell of the c-CT, T, constructed above. Let u be a node in Q. Denote the corresponding canonical cube as $s(u)$, and a concentric cube

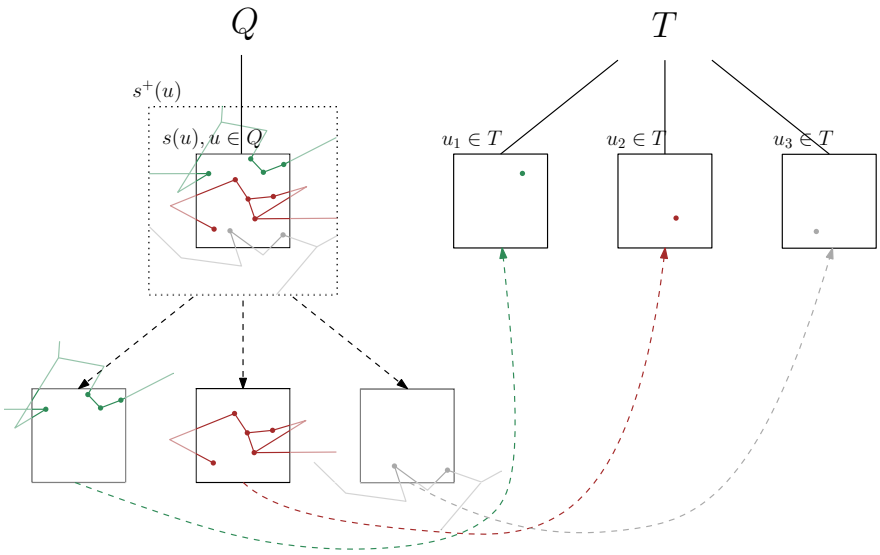


Figure 1 An illustration of iteration j of the algorithm constructing the c-connected tree.

of twice its diameter as $s^+(u)$. We call a connected subgraph of G a connected component of a cell u if it is fully contained within $s^+(u)$, with at least one vertex inside $s(u)$. To compute an upper bound on the graph diameter of the c-connected set we instead give an upper bound on *any* connected component of $u \in Q$. This upper bound can then be copied across during the construction of T. We show how to compute this upper bound below.

Observe that in order for an edge to contribute to a connected component of u, its length can be at most twice the diameter of $s(u)$. Let us call the set of edges of length at most twice the diameter, which overlap with $s(u)$, *relevant* edges w.r.t. $s(u)$ (see Figure 2).

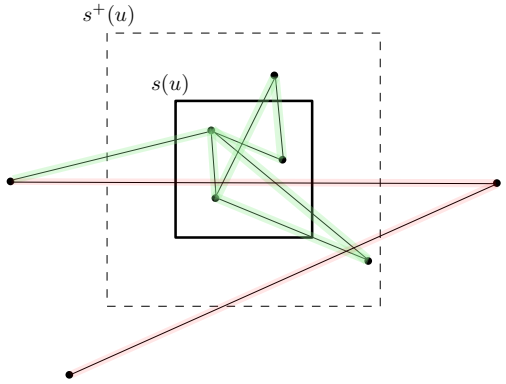


Figure 2 Set of *relevant* edges w.r.t. $s(u)$ (green edges) and *irrelevant* edges (red edges).

Let $N(s(u))$ be the set of $3^d - 1$ neighboring canonical cubes on a grid (note that not all members of $N(s(u))$ may correspond to cells in Q). Observe that all edges in any connected component associated with u must be a subset of the union of the relevant edges w.r.t. $s(u)$, and all the relevant edges w.r.t. cubes in $N(s(u))$. We refer to the total overlapping length of $s(u)$ with relevant edges w.r.t. $s(u)$ as $\mathrm{REL}(s(u))$. The above observation allows us to upper bound the graph diameter of any connected component associated with $u \in Q$ by $\mathrm{REL}(s(u)) + \sum_{s_n \in N(s(u))} \mathrm{REL}(s_n)$.

It thus remains to compute, for all $u \in Q$, $\text{REL}(s(u))$ and $\text{REL}(s_n)$ for all $s_n \in N(s(u))$. Let $\ell(e)$ denote the length of an edge $e \in E$. For all $e \in E$ we compute the canonical cubes of size 2^{-i} such that $2^{-i} = \lceil \ell(e)/2 \rceil$. Denote this set of canonical cubes as S. For each cube $s \in S$, store the length of the overlap of the edge with the cube. Note that this correctly gives us the value $\text{REL}(s)$ for all $s \in S$. One can now use Lemma 2.11 from [24] to compute a compressed quadtree Q_S from S. For any cell v in Q_S whose corresponding cube is not in S, initialize its value $\text{REL}(v) = 0$. Next, one can use a bottom-up approach to compute, for every cell in Q_S its relative edge length. For each parent, simply add the relative edge lengths stored in the children to its current value. In order to perform efficient searching on Q_S we preprocess it into a finger tree, T_f, using similar techniques to Theorem 2.14 in [24]. We are now ready to compute, for each cell $u \in Q$, an upper bound on the graph diameter of any connected component. For each $u \in Q$, search for the largest cell in T_f whose corresponding cube is fully contained within $s(u)$, and return the relative edge length stored at the cell. If such a cell does not exist, return zero. Repeat the search for each $s \in N(s(u))$ and add up the results. The resulting value is our upper bound on the graph diameter of any connected component of the cell u. We call this value $\text{dub}_G(u)$.

When constructing the c-CT from Q using the algorithm in Section 4.2.1, we upper bound the graph diameter of any connected component of $u \in Q$ by $\text{dub}_G(u)$.

The proof of Lemma 11 can be found in the full version of the paper.

▶ **Lemma 11.** *The value $dub_G(v)$ for any cell in $v \in T$ upper bounds the graph diameter of the connected subgraph represented in the cell v.*

We are now ready to describe the construction of a WSPD_G for c-packed graphs.

4.2.3 Constructing the WSPD_G

Construct a compressed quadtree Q and compute, for each cell $u \in Q$, $\text{dub}_G(u)$ using the techniques described in Section 4.2.2. Next, construct a c-CT, T, from Q using the algorithm described in Section 4.2.1 and store, for each cell $v \in T$, created from $u \in Q$, $\text{dub}_G(u)$ as an upper bound on the graph diameter of the connected component represented in v. Apply the algorithm of [24] (Section 3.1.1) to compute a WSPD_G from T. To determine whether a pair of nodes is well-separated we use the upper bound on the graph diameter and the Euclidean distance between their representatives.

To bound the size of the resulting WSPD_G we first bound the size of T by observing that each cell $u \in Q$ contains at most $O(c)$ c-connected sets w.r.t. u.

▶ **Lemma 12.** *For all $u \in U_{i_j}$, $1 \leq j \leq \ell$, each vertex in H_{i_j} within $s(u)$ represents a c-connected component in $H_{i_{j-1}}$ with respect to $s(u)$.*

Proof. We argue the algorithm maintains this property by induction on the number of iterations. The algorithm initially uses as input the c-packed graph $H_{i_1} = H_{i_0} = G$. After the first iteration of the algorithm, for each node $u \in U_{i_1}$, each surviving point x in V_{i_1} represents a set of points in V_{i_0} that were connected to x within $s^+(u)$. Since $H_{i_0} = G$ is c-packed, we can conclude that the distance between x and any point merged into x is at most c times the radius of $s^+(u)$, and therefore c times the diameter of $s(u)$.

Assume that the algorithm maintains the invariant for iterations 1 to $k < \ell$. Note that the graph H_{i_k}, used as input for iteration $k+1$, is no longer guaranteed to satisfy c-packedness. However, observe that after each merge step, the representative nodes in $V_{i_{k+1}}$ always correspond to original nodes present in G. Since the process of merging each connected set into a single node can only reduce the length of the paths between the

remaining representatives, the graph distances between the remaining nodes in H_{i_k} can only have decreased in comparison to their original graph distance in G. We therefore maintain the property that, for every node $u \in U_{i_{k+1}}$, for each representative point in $V_{i_{k+1}}$ within u, the points in V_{i_k} that it represents can be at most c times the diameter of u apart in H_{i_k}. ◀

With the help of Lemma 12 one can bound the size of T.

▶ **Lemma 13.** *For every node $u \in Q$, the number of nodes added to T is at most $O(c)$.*

Proof. From Lemma 12 we know that each node in $V_{i_l}(s(u))$ must represent a valid c-connected set in $H_{i_{l-1}}(s(u))$. We now observe that one can upper bound the number of c-connected sets in $H_{i_{l-1}}(s(u))$ by the number of c-connected sets in $G(s(u))$. This observation follows from the fact that for any $1 \leq j \leq \ell$, H_{i_j} was constructed by merging c-connected sets w.r.t. $s(u)$ in $H_{i_{j-1}}$, for every $u \in U_{i_j}$. Note that the argument used in the proof of Lemma 12 shows that this process, for every j, can only cause the length of the paths between surviving nodes to become smaller. Since the position of surviving nodes does not change from their original position in G, it follows that the number of c-connected components in H_{i_j} within $s(u)$ can only be smaller than that in G within $s(u)$. It thus remains to argue that the number of c-connected components within each $s(u)$, for all $u \in V$ is at most $O(c)$. To this end, note that in order for any pair of points in $V(s)$ to be disconnected within s^+, every path between them must contain a subpath which intersects the boundaries of s and s^+ and therefore has length at least $\frac{r}{\sqrt{d}}$, as shown in Figure 3. From the c-packedness of G we conclude there cannot be more than $\frac{2cr\sqrt{d}}{r} = 2c\sqrt{d}$ such paths in s^+. We conclude there can be at most $O(c)$ c-connected sets found by the algorithm. ◀

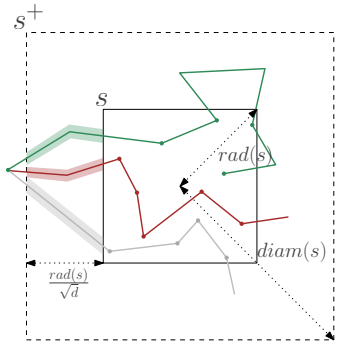


Figure 3 Two c-connected components in s must be separated by a path of length at least $\frac{rad(s)}{\sqrt{d}}$.

With the help of the above lemma we can analyse the size of the resulting WSPD_G using a simple charging argument inspired by the one used in [24].

▶ **Lemma 14.** *The resulting $WSPD_G$, W, is σ-well separated and is of size $O(c^3\sigma n)$.*

Proof. The fact that W is σ-well separated follows from our use of the Euclidean distance as a lower bound on the distance between two representatives, and our use of dub_G (Section 4.2.2) as upper bound on the graph diameter, to determine whether a pair is σ-well separated.

To prove the size of W we use a simple charging argument. For a pair $(u, v) \in W$, let $\pi(v)$ denote the parent of v in T. Assume the last call to the algorithm in [24] (Section 3.1.1) was via $(\pi(v), u)$. We charge the pair to u and argue any node can be charged at most $O(c^2\sigma)$ times. Since the pair $(\pi(v), u)$ was considered not well-separated by the algorithm, it

follows that $d(\pi(v), u) \leq d_G(\pi(v), u) < \text{dub}_G(\pi(v))\sigma$. Consider a geometric ball $B(u, r)$ of radius $r = (1 + \sigma)\text{dub}_G(\pi(v))$ centered at u. Note that the connected set in G represented by $\pi(v)$ must be fully contained inside B. By c-packedness, there can be at most $c\sigma + 1$ candidates for $\pi(v)$. Since $\pi(v)$ can have at most $2^d \cdot O(c)$ children, it follows that there are $O(c^2\sigma)$ candidates for v. We conclude a node u can be charged at most $O(c^2\sigma)$ in this way. By Lemma 13 there are $O(cn)$ nodes in T. This concludes the lemma. ◀

We now argue the runtime and space complexity of our construction.

4.2.4 Complexity analysis

In Lemma 15 the runtime of the construction of the c-CT is analysed, as well as the size of the resulting tree. We then analyse the runtime and space complexity for computing the upper bound on the graph diameter of the subgraph represented in each cell of the c-CT in Lemma 16. Finally, with the help of these lemmas we analyse the construction time and space complexity of our WSPD_G in Theorem 3. The proofs of Lemma 15 and Lemma 16 can be found in the full version of the paper.

▶ **Lemma 15.** *Given a c-packed graph G, one can construct a c-CT of size $O(cn)$ in $O(n \log n + c^2 n)$ time.*

▶ **Lemma 16.** *Let G be a c-packed graph and Q its corresponding compressed quadtree. One can compute an upper bound on the graph diameter of any c-connected component contained in u, for all $u \in Q$, in $O(cn \log n)$ time, using at most $O(cn)$ space.*

With the help of the above lemmas we obtain the following theorem.

▶ **Theorem 3.** *Given a c-packed graph G in $\mathbb{R}^d$, for fixed d, one can construct a WSPD_G with separation factor σ of size $O(c^3\sigma n)$ in $O(cn \log n + c^3\sigma n)$ time, using $O(cn)$ space.*

Proof. Constructing the compressed quadtree Q requires $O(n \log n)$ time and $O(n)$ space. Upper bounding the diameter using the techniques introduced in Section 4.2.2 can be done in $O(cn \log n)$ time, using $O(cn)$ space (Lemma 16). We conclude from Lemma 15 that we can construct a c-CT, T, from Q in $O(c^2 n)$ time, using $O(cn)$ space. Finally, using the construction from [24] we can compute the WSPD_G from T in $O(c^3\sigma n)$ time using $O(cn)$ space. This concludes the proof. ◀

5 A Separator Theorem for c-packed Graphs

A subset of vertices C of a graph with n vertices is called a (balanced) separator if the remaining vertices can be partitioned into two sets A and B such that there are no edges between A and B, with $|A| < \alpha \cdot n$ and $|B| < \alpha \cdot n$ for some constant $1/2 \leq \alpha < 1$. The subset C is said to *α-separate* the graph.

In this section we show how to compute a separator of size $O(c)$ for any c-packed graph that $(1 - \frac{1}{2\lambda^3})$-separates the graph. The constant λ is the doubling constant of $\mathbb{R}^d$.

We have the following separating lemma.

▶ **Lemma 17.** *Given a set P of n points in $\mathbb{R}^d$, where d is fixed, one can compute a ball $b(p, r)$, which is centered at p and has radius r, such that $\mathbf{b}(p, r)$ contains at least $n/(2\lambda^3)$ points of P, where λ is the doubling constant of $\mathbb{R}^d$, and $\mathbf{b}(p, 2r)$ of twice the radius contains at most $n/2$ points of P. The running time of the algorithm is $O(\lambda^{3d} n)$.*

Proof. Let $r_{opt}(P,k)$ be the radius of the smallest ball enclosing k points of P. Har-Peled and Mazumdar [24] gave an algorithm for computing a k-enclosing ball with radius at most $2r_{opt}(P,k)$ in $O(n(n/k)^d)$ time. As a result, one can compute a $(n/(2\lambda^3))$-enclosing ball $\mathbf{b}(p,r)$ in $O(\lambda^{3d}\cdot n)$ time.

Now one can prove that $\mathbf{b}(p,2r)$ contains at most $n/2$ points. Since $r \leq 2r_{opt}(P,n/(2\lambda^3))$, by the doubling property, $\mathbf{b}(p,2r)$ can be covered by at most λ^3 balls of radius $r_{opt}(P,n/(2\lambda^3))/2$. Note that any ball of radius $r_{opt}(P,n/(2\lambda^3))/2$ contains strictly less than $n/(2\lambda^3)$ points. Thus $\mathbf{b}(p,2r)$ contains at most $n/2$ points. This concludes the proof of the lemma. ◀

Based on Lemma 17, we can show that one can construct separators of size $O(c)$ that $(1-\frac{1}{2\lambda^3})$-separate c-packed graphs, as shown in Theorem 4.

▶ **Theorem 4.** *Given a c-packed graph G in $\mathbb{R}^d$, where d is fixed, with n vertices, one can find a separator of size $O(c)$ that $(1-\frac{1}{2\lambda^3})$-separates G, in $O(c^2n)$ time.*

Proof. First we construct a separator C. Then we prove that C satisfies the required properties. Finally we analyze the running time of the algorithm.

Let P be the vertices of c-packed graph G and run the algorithm described in Lemma 17. Let $\mathbf{b}(p,r)$ be the computed ball that contains at least $n/(2\lambda^3)$ vertices of G. Let $\hat{A}$ be the vertices of G contained in $\mathbf{b}(p,r)$ and let $\hat{B}$ be the vertices of G lying outside $\mathbf{b}(p,2r)$. See Figure 4 for an illustration.

Now set up a max-flow instance. A similar construction was used in Lemma 11 of [20]. Set the capacity of each edge of G to 1. Let the vertices in $\hat{A}$ be the sources, and let the vertices in $\hat{B}$ be the sinks. Run the Ford-Fulkerson algorithm ([31], Chapter 7) on the instance. According to the max-flow min-cut theorem ([31], Chapter 7), the max-flow of the instance is equal to its min-cut. Let (S,T) be a min-cut and let $\{e_1,\ldots,e_l\}$ be the set of edges from S to T, where $\hat{A}\subseteq S$ and let $\hat{B}\subseteq T$. Choose one endvertex for each edge in $\{e_1,\ldots,e_l\}$ and add it to C (C is initially empty). Let $A=S\setminus C$ and let $B=T\setminus C$. This finishes the construction of C.

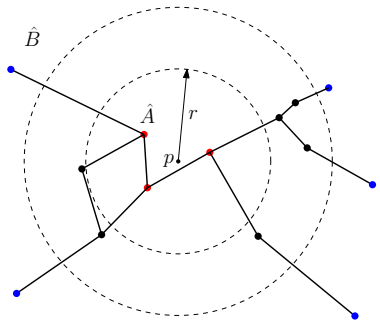


Figure 4 Illustration of the proof of Theorem 4 where p is the center of ball $\mathbf{b}(p,r)$. Vertices in $\hat{A}$ are drawn in red. Vertices in $\hat{B}$ are drawn in blue. The value of the min-cut in the figure is 4.

Next we show that C (i) has size $O(c)$, and (ii) $(1-\frac{1}{2\lambda^3})$-separates G. Property (ii) follows from (S,T) being a min-cut. Since $\hat{A}$ contains at least $n/(2\lambda^3)$ vertices of G and $\hat{A}\subset S$, A contains at least $n/(2\lambda^3)$ vertices of G. Since $\hat{B}$ contains at least $n/2$ vertices of G and $\hat{B}\subset T$, B contains at least $n/2$ vertices of G. Property (i) follows from c-packedness. In the max-flow instance, the value of the max-flow is l. Since all the edges have capacity 1, there are l edge-disjoint paths from $\hat{A}$ to $\hat{B}$. Each such path pierces both the inner and the

outer boundaries of the d-dimensional spherical shell $\mathbf{b}(p, 2r) \setminus \mathbf{b}(p, r)$. Due to c-packedness, the length of edges inside the spherical shell is at most $2cr$. Since the width of the spherical shell is r, there are at most $2c$ edge-disjoint paths from $\hat{A}$ to $\hat{B}$. Therefore $|C| = l \leq 2c$.

Finally we analyze the running time of the algorithm. Running the algorithm in Lemma 17 on the vertices of G takes $O(\lambda^{3d}n)$ time. We use the Ford-Fulkerson algorithm to solve the max-flow instance. The running time of the algorithm is proportional to the number of edges in G times the value of the max-flow. Since $l \leq 2c$ and there are $O(cn)$ edges in G, solving the max-flow instance takes $O(c^2n)$ time. Since $\lambda = O(d)$, the overall running time of the algorithm is $O(c^2n)$. ◀

Putting the above results together gives us an exact distance oracle. We refer the reader to the full version of the paper for further details.

▶ **Theorem 18.** *Given any c-packed graph G with n vertices, using $O(c^2 n \log n + cn \log^2 n)$ preprocessing time and $O(cn \log n)$ space, a distance query between any two vertices in G can be answered in $O(c \log n)$ time.*

This enables us to obtain Theorem 5 and Corollary 6 (see the full version of the paper for details).

References

1 N. Alon, P. D. Seymour, and R. Thomas. A separator theorem for graphs with an excluded minor and its applications. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 293–299, 1990.

2 S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. H. M. Smid. Euclidean spanners: short, thin, and lanky. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC)*, pages 489–498, 1995.

3 Y. Bartal, O. N. Fandina, and O. Neiman. Covering metric spaces by few trees. *J. Comput. Syst. Sci.*, 130:26–42, 2022. doi:10.1016/J.JCSS.2022.06.001.

4 U. Bertele and F. Brioschi. *Nonserial dynamic programming*. Academic Press, Inc., 1972.

5 F. Brüning, J. Conradi, and A. Driemel. Faster approximate covering of subcurves under the Fréchet distance. In *Proceedings of the 30th Annual European Symposium on Algorithms, (ESA)*, volume 244 of *LIPIcs*, pages 28:1–28:16, 2022. doi:10.4230/LIPICS.ESA.2022.28.

6 K. Buchin, M. Buchin, J. Gudmundsson, A. Popov, and S. Wong. Map matching queries under Fréchet distance on low-density spanners. In *Proceedings of the 40th Annual Symposium on Computational Geometry (SoCG)*, 2024.

7 P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42(1):67–90, 1995. doi:10.1145/200836.200853.

8 H. Chang, J. Conroy, H. Le, L. Milenkovic, and C. Than. Covering planar metrics (and beyond): O(1) trees suffice. In *Proceedings of the 64th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2231–2261, 2023.

9 S. Chaudhuri and C. D. Zaroliagis. Shortest paths in digraphs of small treewidth. part I: sequential algorithms. *Algorithmica*, 27(3):212–226, 2000. doi:10.1007/S004530010016.

10 B. Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2:337–361, 1987. doi:10.1007/BF01840366.

11 D. Chen, A. Driemel, L. J. Guibas, A. Nguyen, and C. Wenk. Approximate map matching with respect to the Fréchet distance. In *Proceedings of the 13th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 75–83, 2011.

12 J. Conradi, A. Driemel, and B. Kolbe. $(1+\epsilon)$-ANN data structure for curves via subspaces of bounded doubling dimension. *Comput. Geom. Topol.*, 3(2):6:1–6:22, 2024. URL: https://www.cgt-journal.org/index.php/cgt/article/view/45.

13 A. Driemel and S. Har-Peled. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013. doi:10.1137/120865112.

14 A. Driemel, S. Har-Peled, and C. Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discret. Comput. Geom.*, 48(1):94–127, 2012. doi:10.1007/S00454-012-9402-Z.

15 Z. Dvorák and S. Norin. Treewidth of graphs with balanced separations. *J. Comb. Theory B*, 137:137–144, 2019. doi:10.1016/J.JCTB.2018.12.007.

16 J. Gao and L. Zhang. Well-separated pair decomposition for the unit-disk graph metric and its applications. *SIAM J. Comput.*, 35(1):151–169, 2005. doi:10.1137/S0097539703436357.

17 J. R. Gilbert, J. P. Hutchinson, and R. E. Tarjan. A separator theorem for graphs of bounded genus. *J. Algorithms*, 5(3):391–407, 1984. doi:10.1016/0196-6774(84)90019-1.

18 J. Gudmundsson, Z. Huang, A. van Renssen, and S. Wong. Computing a subtrajectory cluster from c-packed trajectories. In *Proceedings of the 34th International Symposium on Algorithms and Computation (ISAAC)*, volume 283 of *LIPIcs*, pages 34:1–34:15, 2023. doi:10.4230/LIPICS.ISAAC.2023.34.

19 J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. H. M. Smid. Approximate distance oracles for geometric spanners. *ACM Trans. Algorithms*, 4(1):10:1–10:34, 2008. doi:10.1145/1328911.1328921.

20 J. Gudmundsson, M. P. Seybold, and S. Wong. Map matching queries on realistic input graphs under the Fréchet distance. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1464–1492, 2023.

21 J. Gudmundsson and M. H. M. Smid. Fast algorithms for approximate Fréchet matching queries in geometric trees. *Comput. Geom.*, 48(6):479–494, 2015. doi:10.1016/J.COMGEO.2015.02.003.

22 J. Gudmundsson and S. Wong. A well-separated pair decomposition for low density graphs. *CoRR*, abs/2411.08204, 2024. doi:10.48550/arXiv.2411.08204.

23 A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Proceedings of the 44th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 534–543, 2003.

24 S. Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, 2011.

25 S. Har-Peled and M. Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006. doi:10.1137/S0097539704446281.

26 S. Har-Peled and K. Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. *SIAM J. Comput.*, 46(6):1712–1744, 2017. doi:10.1137/16M1079336.

27 S. Har-Peled and Benjamin Raichel. The Fréchet distance revisited and extended. *ACM Trans. Algorithms*, 10(1):3:1–3:22, 2014. doi:10.1145/2532646.

28 K. Kawarabayashi, P. N. Klein, and C. Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 6755 of *Lecture Notes in Computer Science*, pages 135–146, 2011. doi:10.1007/978-3-642-22006-7_12.

29 K. Kawarabayashi, C. Sommer, and M. Thorup. More compact oracles for approximate distances in undirected planar graphs. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 550–563, 2013.

30 P. N. Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 820–827, 2002.

31 Jon M. Kleinberg and Éva Tardos. *Algorithm design*. Addison-Wesley, 2006.

32 R. Kyng, R. Peng, R. Schwieterman, and P. Zhang. Incomplete nested dissection. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 404–417, 2018.

33 H. Le and C. Wulff-Nilsen. Optimal approximate distance oracle for planar graphs. In *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 363–374, 2021.

34 R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math*, 36(2):177–189, 1979.

35 R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980. doi:10.1137/0209046.

36 G. L. Miller and W. P. Thurston. Separators in two and three dimensions. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 300–309, 1990.

37 G. Narasimhan and M. H. M. Smid. *Geometric spanner networks*. Cambridge University Press, 2007.

38 K. Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 281–290, 2004.

39 M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, 2004. doi:10.1145/1039488.1039493.

40 M. Thorup and U. Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

41 C. Wulff-Nilsen. Approximate distance oracles for planar graphs with improved query time-space tradeoff. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 351–362, 2016.

On Minimizing Wiggle in Stacked Area Charts

Alexander Dobler
TU Wien, Austria

Martin Nöllenburg
TU Wien, Austria

Abstract

Stacked area charts are a widely used visualization technique for numerical time series. The x-axis represents time, and the time series are displayed as horizontal, variable-height layers stacked on top of each other. The height of each layer corresponds to the time series values at each time point. The main aesthetic criterion for optimizing the readability of stacked area charts is the amount of vertical change of the borders between the time series in the visualization, called *wiggle*. While many heuristic algorithms have been developed to minimize wiggle, the computational complexity of minimizing wiggle has not been formally analyzed. In this paper, we show that different variants of wiggle minimization are NP-hard and even hard to approximate. We also present an exact mixed-integer linear programming formulation and compare its performance with a state-of-the-art heuristic in an experimental evaluation. Lastly, we consider a special case of wiggle minimization that corresponds to the fundamentally interesting and natural problem of ordering a set of numbers as to minimize their sum of absolute prefix sums. We show several complexity results for this problem that imply some of the mentioned hardness results for wiggle minimization.

2012 ACM Subject Classification Mathematics of computing → Permutations and combinations; Theory of computation → Theory and algorithms for application domains; Theory of computation → Problems, reductions and completeness

Keywords and phrases Stacked area charts, NP-hardness, Mixed-integer linear programming

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.22

Related Version *Full Version*: `https://arxiv.org/abs/2506.21175` [5]

Supplementary Material *Software (Source code and experimental evaluation)*: `https://doi.org/10.5281/zenodo.15745936`

Funding *Alexander Dobler*: Supported by the Vienna Science and Technology Fund (WWTF) under grant [10.47379/ICT19035].
Martin Nöllenburg: Supported by the Vienna Science and Technology Fund (WWTF) under grant [10.47379/ICT19035].

1 Introduction

Stacked area charts are a widely used method for visualizing numerical time series across discrete time points, such as population data of countries over time or movie revenues over multiple weeks [4, 9]. In stacked area charts, the x-dimension depicts time, and the time series are stacked as variable-height strips on top of each other without gaps, such that their heights depict their values (see Figure 1a). The lowest border of the stacked time series is a straight horizontal line.

A primary aesthetic criterion for stacked area charts is *wiggle* – the aggregated amount of vertical change of the borders between the time series in the visualization (Figure 1b). Further, the wiggle is usually weighted by the time series values, resulting in *weighted wiggle* (for a formal definition refer to Section 2). For a stacked area chart of a specific dataset, the amount of wiggle solely depends on the vertical ordering of the time series. The task of minimizing wiggle has already been tackled with heuristic methods [4, 13]. But despite the

© Alexander Dobler and Martin Nöllenburg;
licensed under Creative Commons License CC-BY 4.0

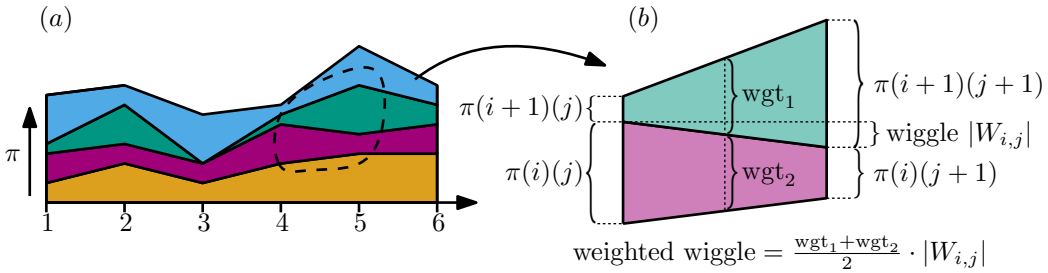


Figure 1 (a) A stacked area chart of four time series on six time points. The time series are ordered according to π from bottom to top. (b) Illustration of the wiggle between time points 4 and 5 on the border between the green and violet time series. For the weighted wiggle, $|W_{i,j}|$ is multiplied by the average of $\pi(i)(j)$, $\pi(i)(j+1)$, $\pi(i+1)(j)$, and $\pi(i+1)(j+1)$. The total (weighted) wiggle is the sum of all (weighted) wiggle values.

popularity of stacked area charts in mainstream visualizations, no work was done with regard to the computational complexity of minimizing wiggle. This paper fills the gap and presents several complexity lower bounds for minimizing wiggle and weighted wiggle. We also compare a state-of-the-art heuristic for minimizing weighted wiggle with a new mixed-integer linear programming formulation on a set of real-world data; the evaluation shows that the heuristic performs well with regard to wiggle minimization and that the mixed-integer program can solve small to medium-size instances exactly. Further, a special case of minimizing wiggle is discussed that results in a fundamentally interesting problem – ordering a set of numbers such that the accumulation of all absolute prefix sums is minimized. Despite its natural problem definition, the problem has not been studied yet. Its relevance comes from its relation to several ordering problems that minimize accumulated cost, such as scheduling problems. Results for this problem are also used to show some hardness results for minimizing wiggles.

Related work. The origin of stacked area charts is unclear, as they are a very natural and frequently used visualization for a commonly occurring type of data. The most notable work for popularizing research on stacked area charts is from 2008 by Byron and Wattenberg [4], since receiving more than 600 citations. Given the extensive literature on stacked area charts, we focus on studies specifically addressing wiggle minimization and similar aesthetic criteria in related visualizations. Byron and Wattenberg [4] only deal with data that represent movie revenues. Such data have very specific properties – a movie starts airing, its revenues grow quickly, and then the revenues slowly decrease. Thus, they present a very specific ordering approach that orders the movies by their first screening. They also consider a similar type of visualization, called *streamgraphs*. The only difference between streamgraphs and stacked area charts is that the lower border of a streamgraph does not have to be a horizontal line, introducing another degree of freedom. Still, wiggle minimization is defined equivalently for streamgraphs. Byron and Wattenberg present a similar ordering procedure for streamgraphs. Greffard and Kuntz [7] present an approach for weighted squared wiggle minimization in streamgraphs that first defines pairwise dissimilarity values between time series and then applies traveling salesperson approaches. Di Bartolomeo and Hu [2] present a heuristic and local search algorithm for weighted wiggle minimization in streamgraphs, which also work for stacked area charts. Bu et al. [3] present an ordering algorithm for streamgraphs that is based on clustering. They also optimize for minimizing what they call *sine illusion effect* – time series borders mimicking a sine wave. He and Li [8] present an ordering approach for stacked area charts based on a traveling salesperson formulation. Their approach aims to minimize the covariance between adjacent time series in the stacked area chart. Lastly, Mathiesen

and Schulz [13] present a heuristic algorithm that is able to minimize a weighted sum of quality criteria for stacked area charts, including wiggle. Their approach is an improvement over an algorithm by Di Bartolomeo and Hu [2], and they demonstrate its effectiveness in an experimental evaluation.

The problem of ordering a set of numbers to minimize the accumulation of all absolute prefix sums is related to some classic problems from scheduling. Tsai [12] studies the problem where instead of minimizing the accumulation, one wants to minimize the maximum absolute prefix sum. They claim strong NP-hardness (a formal proof is absent). Kellerer et al. [10] consider the same problem as Tsai with the additional constraint that each prefix sum must be positive, and show constant-factor approximation algorithms. Further, they explicitly pose the open problem of minimizing the sum of all prefix sums. In their definition, though, the ordering has to satisfy that each prefix sum is positive, which is not the case for us.

Our contribution. We consider wiggle minimization and weighted wiggle minimization in stacked area charts from a computational complexity perspective. For this, we define the two computational problems p-WiggleMin and Weighted-p-WiggleMin in Section 2. The value p in both problem definitions corresponds to the exponent of each wiggle value in the objective function, in turn capturing variants of wiggle minimization such as the weighted squared wiggle minimization from Byron and Wattenberg [4]. Further, we introduce and investigate a special case of instances leading to the problem Min-$\sum$|Prefixsum| – ordering a set of numbers to minimize the sum of absolute prefix sums.

In Section 3, we show that Min-$\sum$|Prefixsum| is strongly NP-hard and that both wiggle problems with $p = 1$ are strongly NP-hard, even if the number of time points is constant. Next, we also consider special cases of Min-$\sum$|Prefixsum|, where there is only one positive or one negative element in the input. Section 4 shows that both wiggle problems for arbitrary p are strongly NP-hard, and hard to approximate under specific complexity assumptions. Further, a lower bound on the approximation ratio of a known greedy heuristic used in [2] and [13] is shown. Lastly, Section 5 presents a mixed-integer linear program for Weighted-1-WiggleMin and compares it with the state-of-the-art heuristic of Mathiesen and Schulz [13] in an experimental evaluation on real-world data.

Due to space constraints, statements marked with ★ are proved in the appendix.

2 Preliminaries and Problem Definitions

We define $[n] = \{1, \dots, n\}$. A permutation π of a multiset $S = \{s_1, \dots, s_n\}$ is a bijection $\pi : [n] \mapsto S$. We sometimes treat π as the list $(\pi(1), \pi(2), \dots, \pi(n))$. Further, we define $\mathrm{pos}_\pi(s) = i$ if and only $\pi^{-1}(s) = i$.

An *ℓ-time series* f is a function $f : [\ell] \to \mathbb{R}_0^+$. We refer to the elements of its image as *data points*. A set F of ℓ-time series, is *balanced* if $\sum_{f \in F} f(i) = \sum_{f \in F} f(j)$ for all $i, j \in [\ell]$.

A computational problem is *strongly* NP*-hard* if it remains NP-hard even if its numerical parameters are integers that are polynomial in the input size.

Problem Definitions

In most problems discussed in this paper, we are given a set $F = \{f_1, \dots, f_n\}$ of ℓ-time series. Given a permutation π of F, $i \in \{0\} \cup [n]$, and $j \in [\ell - 1]$, we define the *wiggle* value $W^\pi_{i,j} = \sum_{k=1}^{i} (\pi(k)(j+1) - \pi(k)(j))$. This results in the following two problem variants of *wiggle minimization for stacked area charts*, the first unweighted, and the second weighted by the time series data points - as is more common in the stacked area charts literature [2, 7, 13] (see also Figure 1b). For both problems, p is a positive integer corresponding to the exponent of the wiggle values. The second problem is equivalent to the minimization of *flatness* in [13].

▶ **Problem 1** (p-WiggleMin)**.** Given a set $F = \{f_1, \ldots, f_n\}$ of ℓ-time series, find a permutation π of F such that $\sum_{i=1}^{n} \sum_{j=1}^{\ell-1} |W_{i,j}^{\pi}|^p$ is minimized.

▶ **Problem 2** (Weighted-p-WiggleMin)**.** Given a set $F = \{f_1, \ldots, f_n\}$ of ℓ-time series, find a permutation π of F such that

$$\sum_{i=1}^{n} \sum_{j=1}^{\ell-1} \frac{\pi(i)(j) + \pi(i)(j+1) + \pi(i+1)(j) + \pi(i+1)(j+1))}{4} |W_{i,j}^{\pi}|^p$$

is minimized. Above, $\pi(n+1)$ is the time series containing only zeroes.

If $\ell = 2$, then 1-WiggleMin is equivalent (see Lemma 4) to the following fundamentally interesting problem, which we also study in this paper.

▶ **Problem 3** (Min-$\sum$|Prefixsum|)**.** Given a multiset $S = \{s_1, \ldots, s_n\}$ of real numbers, find a permutation π of S such that $\sum_{i=1}^{n} \left| \sum_{j=1}^{i} \pi(j) \right|$ is minimized.

Given a solution π of Min-$\sum$|Prefixsum|, we define for $i = 0, 1, \ldots, n$, $P_i^{\pi} = \sum_{j=1}^{i} \pi(j)$.

Next, we want to show relationships between the previously defined problems.

▶ **Lemma 4** (★)**.** *Given an instance $S = \{s_1, \ldots, s_n\}$ of* Min-$\sum$|Prefixsum|*, there exists an instance $F = \{f_1, \ldots, f_n\}$ of* 1-WiggleMin *on two time points such that S has a solution with value x if and only if F has a solution with value x. If $\sum_{s \in S} s = 0$, then F is balanced. Further, data point values in F are bounded by a polynomial of the values in S.*

Conversely, for an instance F of 1-WiggleMin *on two time points, there exists an instance S of* Min-$\sum$|Prefixsum| *such that S has a solution with value x if and only if F has a solution with value x.*

▶ **Lemma 5** (★)**.** *Let $p \geq 1$ be an integer. Given a balanced instance $F = \{f_1, \ldots, f_n\}$ of* p-WiggleMin *on ℓ time points, there is an instance $F' = \{f'_1, \ldots, f'_n\}$ of* Weighted-p-WiggleMin *on $4\ell + 1$ time points and a constant C such that F has a solution with value x if and only if F' has a solution with value Cx. Further, the values of the data points in F' are polynomially bounded w.r.t. the values of the data points in F.*

A trivial algorithm for the mentioned problems would need to go over all permutations, and thus need $\mathcal{O}(n!|I|)$ time when $|I|$ is the instance size. A simple improvement is to use dynamic programming by computing the optimal solution for each subset of time series (or elements in S), leading to an algorithm needing time $\mathcal{O}(2^n|I|^c)$ for some constant c. In the following sections, we want to find out whether polynomial time exact, or approximation algorithms for the problems are likely.

3 Complexity of Min-$\sum$|Prefixsum| and (Weighted)-1-WiggleMin

In this section, we consider the problem Min-$\sum$|Prefixsum|. We will show several complexity results, some of which will imply results for 1-WiggleMin and Weighted-1-WiggleMin.

3.1 Strong NP-hardness

Most of this section is devoted to showing strong NP-hardness of Min-$\sum$|Prefixsum|. The proof idea is to reduce instances of a known NP-hard problem to instances S of Min-$\sum$|Prefixsum|, such that S has a solution value which is equal to a lower bound if and only if the instance which we reduced from is a yes instance. This lower bound is stated next.

▶ **Lemma 6.** *A lower bound for the objective function of* Min-$\sum$|Prefixsum| *is* $\sum_{s\in S}|s|/2$.

Proof. Let π be some solution of Min-$\sum$|Prefixsum|. It is easy to see that for $i = 1, \ldots, n$,

$$|P_i^\pi| + |P_{i-1}^\pi| \geq |\pi(i)|.$$

Thus,

$$2\sum_{i=0}^{n}|P_i^\pi| \geq |P_0^\pi| + |P_n^\pi| + 2\sum_{i=1}^{n-1}|P_i^\pi| \geq \sum_{i=1}^{n}|s_i| \qquad \blacktriangleleft$$

We call a solution π achieving this bound *bound-achieving*. The next lemma states a sufficient condition that implies that the solution value exceeds the previously stated lower bound.

▶ **Lemma 7.** *Let π be a solution of* Min-$\sum$|Prefixsum| *and $\sum_{i=1}^{n} s_i = 0$. There exists $j \in [n]$ such that $|P_j^\pi| + |P_{j-1}^\pi| > |\pi(j)|$ if and only if $\sum_{i=1}^{n}|P_i^\pi| > \sum_{i=0}^{n}|s_i|/2$.*

Proof. The backwards direction is clear. For the forward direction, we have

$$2\sum_{i=0}^{n}|P_i^\pi| = 2\sum_{i=1}^{n-1}|P_i^\pi| > \sum_{i=1}^{n}|s_i|. \qquad \blacktriangleleft$$

Additionally, in our reduction, we will consider instances S of Min-$\sum$|Prefixsum| that have specific properties. These properties are described below.

(IP1) $\sum_{i=1}^{n} s_i = 0$.
(IP2) $n \equiv 0 \pmod 3$, i.e., n is a multiple of 3.
(IP3) S has exactly $\frac{2n}{3}$ positive elements and $\frac{n}{3}$ negative elements.

Hence, S also has no elements equal to zero. We call an instance that satisfies **(IP1)-(IP3)** *nice*, and observe the following properties that will be useful for the reduction.

▶ **Lemma 8.** *Let π be a bound-achieving solution of a nice instance S. At least one of $\pi(1), \pi(2)$ is negative and at least one of $\pi(n-1), \pi(n)$ is negative.*

Proof. If both $\pi(1), \pi(2)$ are positive, then $\pi(2) < |P_1^\pi| + |P_2^\pi|$. Lemma 7 gives us a contradiction, as we assumed that π is bound-achieving.

If both $\pi(n-1), \pi(n)$ are positive, then we have, as $P_n^\pi = 0$, that $P_{n-1}^\pi = -\pi(n)$ and $P_{n-2}^\pi = -(\pi(n) + \pi(n-1))$. It follows that $|P_{n-2}^\pi| + |P_{n-1}^\pi| > |\pi(n-1)|$, a contradiction. ◀

▶ **Lemma 9.** *Let π be a bound-achieving solution of a nice instance S. If for $i \in [n-1]$, $\pi(i)$ and $\pi(i+1)$ are both positive or both negative, then $P_i^\pi = 0$.*

Proof. If $\pi(i)$ and $\pi(i+1)$ are both positive, and $P_i^\pi \neq 0$ then we have two cases.

- If $P_i^\pi > 0$ then $|P_i^\pi| + |P_{i+1}^\pi| > \pi(i+1)$, a contradiction.
- If $P_i^\pi < 0$ then $|P_{i-1}^\pi| + |P_i^\pi| > \pi(i)$, a contraction.

When both $\pi(i)$ and $\pi(i+1)$, are negative, the proof works the same. ◀

The following is a direct consequence of the last lemma.

▶ **Corollary 10.** *Let π be a bound-achieving solution of a nice instance S. There exists no $i \in [n-2]$ such that $\pi(i), \pi(i+1), \pi(i+2)$ are all negative or all positive.*

With these lemmas out of the way, we can show that π has a unique structure with regard to the signs of its elements.

▶ **Lemma 11.** *Let π be a bound-achieving solution of a nice instance S. Then, for $i \in [n]$,*

- *$\pi(i)$ is positive if and only if $i \equiv 0 \pmod 3$ or $i \equiv 1 \pmod 3$,*
- *$\pi(i)$ is negative if and only if $i \equiv 2 \pmod 3$, and*
- *if $i \equiv 0 \pmod 3$, then $P_i^\pi = 0$.*

Proof. We show the claim by induction on i. For the base case, $i = 1$, assume for a contradiction that $\pi(1)$ is negative. Together with Lemma 8 and the pigeonhole principle, it follows that there must be three consecutive positive elements, a contradiction to Corollary 10.

For the induction step $i > 1$, we have the following cases.

- If $i \equiv 0 \pmod 3$ then $\pi(i)$ must be positive; otherwise, for the remaining $n - i$ following elements, we have $(n-i)/3 - 1$ negative elements and $2(n-i)/3 + 1$ positive elements. By the pigeonhole principle, there are three consecutive positive elements, a contradiction. Further $P_i^\pi = 0$ must hold: For $i = n$ it holds trivially. Otherwise, if P_i^π was not zero, then $\pi(i+1)$ must be negative by Lemma 9. For there to be no three consecutive positive elements in the remaining array, $\pi(n-1)$ and $\pi(n)$ must be positive, a contradiction.
- If $i \equiv 1 \pmod 3$ then $\pi(i)$ is positive by the same argumentation as in the base case.
- If $i \equiv 2 \pmod 3$ then $\pi(i)$ is clearly negative; otherwise, as P_{i-1}^π is positive, $|P_i^\pi| + |P_{i+1}^\pi| > |\pi(i)|$ would hold; a contradiction (see Lemma 7). ◀

With this groundwork, we are able to show the following result.

▶ **Theorem 12.** *The decision variant of* Min-$\sum$|Prefixsum| *is strongly* NP*-complete.*

Proof. NP-membership is obvious, thus we continue with NP-hardness. We reduce from the strongly NP-hard problem Numerical 3-Dimensional Matching [17]. The problem input consists of three multisets X, Y, and Z, each containing k integers. Let $b = (\sum_{e \in X \cup Y \cup Z} e)/k$. The problem is to decide whether there exists a subset M of $X \times Y \times Z$ such that

1. each element from X, Y, and Z appears in exactly one triple, and
2. $x + y + z = b$ holds for each triple $(x, y, z) \in M$.

We can assume that each element from $X \cup Y \cup Z$ is positive (otherwise add to all elements $-\alpha + 1$, where $\alpha = \min X \cup Y \cup Z$ and obtain an equivalent instance with regard to the solution). We also assume that each element from X, Y, and Z is smaller than b. Now define the multisets $X' = \{x \mid x \in X\}$, $Y' = \{y + 2b \mid y \in Y\}$, and $Z' = \{z - 3b \mid z \in Z\}$. Let $n = |X' \cup Y' \cup Z'|$. Notice that $\sum_{e \in X' \cup Y' \cup Z'} e = 0$, and X', Y' and Z' are pairwise disjoint. We now define the instance S of Min-$\sum$|Prefixsum| as $S = X' \cup Y' \cup Z'$. Notice that S is a nice instance satisfying **(IP1)-(IP3)**.

We claim that (X, Y, Z) is a yes-instance of Numerical 3-Dimensional Matching if and only if there exists a solution π of S that is bound-achieving. We prove both directions.

"⇒". Let M be a subset of $X \times Y \times Z$ constituting a solution to the instance of Numerical 3-Dimensional Matching. We constructively define π by going through the triples (x, y, z) in M in any order. Let (x, y, z) be the ith triple. We define $\pi(3i-2) = x, \pi(3i-1) = z - 3b$, and $\pi(3i) = y + 2b$. By the definition of X', Y', and Z' such a permutation always exists. Now notice for $j = 0, 1, \ldots, n$ that P_j^π is

- 0, if and only if $j \equiv 0 \pmod 3$,
- positive if and only if $j \equiv 1 \pmod 3$, and
- negative if and only if $j \equiv 2 \pmod 3$.

Thus, we have $|P_j^\pi| + |P_{j-1}^\pi| = |\pi(j)|$ for all $j \in [n]$. It follows that $\sum_{j=1}^{n} |P_j^\pi| = \sum_{s \in S} |s|/2$.

"⇐". Assume that π is bound-achieving. We construct M, starting with $M = \emptyset$. Because S is nice, we have that Lemma 11 holds for π. For each $i \in [k]$ we have the following. Let $u := \pi(i-2), v := \pi(i-1)$, and $w := \pi(i)$. Because of Lemma 11 we have that $u+v+w = 0$, v is negative, and u and w are positive. Hence, $v \in Z'$ holds. Now we claim that exactly one of u and w is from X' and exactly one is from Y'. Indeed, if both u and w were from X', then $u + w < 2b < |v|$ and $u + v + w = 0$ is impossible. Similarly, if both u and w were from Y', then $u + w > 4b > |v|$ and $u + v + w = 0$ is impossible. Thus, w.l.o.g. assume that $u \in X'$ and $w \in Y'$. Add to M the triple $(u, w - 2b, v + 3b)$. It is easy to verify that after adding this triple for each $i \in [k]$, M verifies that (X, Y, Z) is a yes-instance of Numerical 3-Dimensional Matching. ◀

Lemma 4 and Lemma 5 imply the following corollary.

▶ **Corollary 13.** *The decision variants of* 1-WiggleMin *and* Weighted-1-WiggleMin *are strongly* NP*-complete, even for a constant number of time points.*

3.2 Restricted instances of Min-$\sum$|Prefixsum|

We further ask for which types of instances Min-$\sum$|Prefixsum| becomes tractable. In particular, we want to know whether restricting the number of negative (or positive) elements of S has an effect on the computational complexity of Min-$\sum$|Prefixsum|. Indeed, if S only contains positive or only negative elements, then an optimal ordering π will simply order the elements of S by their absolute value. But what if S contains exactly one positive or exactly one negative element? We obtain the following peculiar results, which state that the sum of S has an effect on the complexity of Min-$\sum$|Prefixsum| in this case.

▶ **Theorem 14 (★).** Min-$\sum$|Prefixsum| *can be solved in time* $\mathcal{O}(n \log n)$ *if* S *contains exactly one negative (positive) element, and* $\sum_{i=1}^{n} s_i = 0$.

▶ **Theorem 15 (★).** *The decision variant of* Min-$\sum$|Prefixsum| *is* NP*-complete, even if* S *contains exactly one negative (positive) element.*

We dedicate the rest of the section to show both results, starting with Theorem 14.

Proof sketch of Theorem 14. We show the result for S containing a single negative element x, the result for a single positive element is shown equivalently. Now consider some permutation π of S, and let $k = \mathrm{pos}_\pi(x)$. We observe that the objective value can be split into absolute prefix sums before and after position k as follows.

$$\begin{aligned}\sum_{i=1}^{n} |\sum_{j=1}^{i} \pi(j)| &= \sum_{i=1}^{k-1}\sum_{j=1}^{i} \pi(j) + \sum_{i=1}^{n-k}\sum_{j=1}^{i} \pi(n-j+1) \\ &= \sum_{i=1}^{k-1} \pi(i)\cdot(k-i) + \sum_{i=1}^{n-k} \pi(n-i+1)\cdot(n-k-i+1).\end{aligned}$$

Thus, we obtain a weighted sum of the positive elements. The coefficient of an element depends only on its position in relation to the position of the negative element. This leads to a simple algorithm, which is also given as pseudocode in the long version of this paper: The permutation is chosen such that the negative element is in the middle. The smallest positive elements are placed at positions 1 and n, the next smallest at positions 2 and $n-1$, etc. ◀

To show Theorem 15, we reduce from the following problem.

▶ **Problem 16** (OneInTwoPartition)**.** Given is a set X containing m pairs of positive integers. No integer appears twice in the union of all the integers. The question is: Are there two sets X_1, X_2 of size m such that both sets contain exactly one element from each pair in X, no element is contained in both sets, and $\sum_{x\in X_1} x = \sum_{x\in X_2} x$?

Clearly, OneInTwoPartition is NP-hard by a straightforward reduction from Partition [6]: Let $Y = \{y_1, \ldots, y_n\}$ be an instance of Partition, asking for a subset $Y' \subseteq Y$ with $\sum_{y\in Y'} y' = \frac{1}{2}\sum_{y\in Y} y$. We can assume all integers in Y to be positive. Let $b = \sum_{y\in Y} y$. The reduced instance consists of the pairs $(i \cdot b + y_i, i \cdot b)$ for $i = 1, \ldots, n$. The correctness of this reduction is immediate. We are ready to prove Theorem 15.

Proof sketch of Theorem 15. Note that the proof is rather technical; to understand it in full detail, the interested reader should refer to the full description in full version of the paper. We reduce from OneInTwoPartition. Consider an instance $X = \{p_1, p_2, \ldots, p_m\}$ of OneInTwoPartition, where $m > 1$. Define $M = \sum_{(x,y)\in X} \frac{x+y}{2}$. We define the following three sets.

$$S_1 = \{m(x + iM) \mid i \in [m], p_i = (x, y)\}$$
$$S_2 = \{m(y + iM) \mid i \in [m], p_i = (x, y)\}$$
$$S_3 = \{imM + 1 \mid i \in [m]\}$$

We continue with further definitions.

$$M_1 = \frac{\sum_{s\in S_1} s + \sum_{s\in S_2} s}{2} = mM + mM\sum_{i=1}^{m} i$$
$$M_2 = \sum_{s\in S_3} s = m + mM\sum_{i=1}^{m} i$$
$$S = S_1 \cup S_2 \cup S_3 \cup \{-(M_1 + M_2)\}$$

For each $(x, y) \in X$ there are unique values $x', y' \in S$ such that for some i, $x' = m(x + iM)$ and $y' = m(y + iM)$. We say that x' is the *correspondent* of x and y' is the *correspondent* of y, and vice versa. We observe that $S \setminus \{-(M_1 + M_2)\}$ can be partitioned uniquely into sets $t_1, t_2, \ldots, t_m$ such that for each t_i we have that

- t_i contains exactly three elements,
- t_i contains one element from S_1, one from S_2, and one from S_3,
- the correspondent of $\{x\} = S_1 \cap t_i$ and the correspondent of $\{y\} = S_2 \cap t_i$ form a pair in X, and
- each integer from t_i is in the interval $[imM, (i + 1)mM)$.

We set S as the instance of Min-$\sum$|Prefixsum| and claim that X is a yes-instance of OneInTwoPartition if and only if there exists a permutation π of S such that

$$\sum_{i=1}^{n} |\sum_{j=1}^{i} \pi(j)| \leq \sum_{i=1}^{m} (m - i + 1) \sum_{x\in t_i} x.$$

We sketch the more interesting direction of the proof, assuming that π is a solution achieving this objective, showing how to construct X_1 and X_2. The idea now is that π will order the elements as shown in Figure 2. Namely, the prefix sum P_{2m+1} will be zero, and the last m elements will be ordered increasingly. Further, $\pi(2m + 1 + i)$ will be from set t_i for each $i \in [n]$. This further implies that the last m elements sum to M_1, which is a

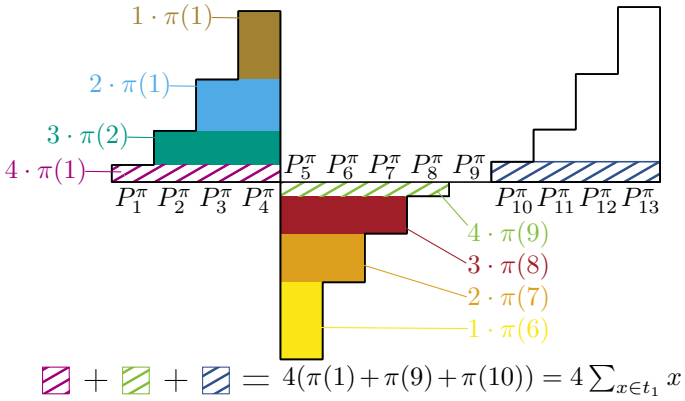


Figure 2 Schematization of the distribution of prefix sums for an optimal solution π of S in the proof of Theorem 15, where $m = 4$. The height corresponds to the prefix sum value, while values below the horizontal middle line are negative. The figure illustrates how the objective value (the area of the union of all enclosed regions) can be decomposed into a weighted sum of the positive elements. Further, the three smallest elements at positions $1, 9$, and 10 contribute to the sum with coefficient 4, resulting in the hatched regions. In an optimal solution, these elements correspond to the three smallest values, constituting t_1.

multiple of m. Together, this means that the last m elements cannot contain any elements from S_3. Now let X_1 be the set of correspondents of the last m elements in π, and let X_2 be the remaining elements from X. The above properties imply that $\sum_{x\in X_1} x = \sum_{x\in X_2} x$. ◀

Note that a pseudo-polynomial algorithm for Min-$\sum$|Prefixsum| cannot be ruled out for the case of one negative (positive) element, as we did not show strong NP-hardness.

4 p-WiggleMin and Weighted-p-WiggleMin

In this section, we show hardness result for p-WiggleMin and Weighted-p-WiggleMin for arbitrary p. We present a reduction which has several complexity implications in Section 4.1. Section 4.2 contains results with regard to approximation lower bounds.

4.1 A Reduction Implying NP-hardness

The reduction is from the well-known problem Minimum Linear Arrangement [6] to p-WiggleMin. The problem is defined as follows.

▶ **Problem 17** (Minimum Linear Arrangement (MLA)). Given an undirected graph $G = (V, E)$ and an integer k, does there exist a permutation π of V such that

$$\sum_{\{u,v\}\in E} |\mathrm{pos}_\pi(u) - \mathrm{pos}_\pi(v)| \leq k?$$

Consider a constant positive integer p. The reduction will work for any such p, and goes as follows. Let (G, k) be an instance of MLA with $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. We construct a set $F = \{f_1, \ldots, f_n\}$ of $(m+1)$-time series. We define the time series inductively based on the time point, starting with the first time point: let $f_i(1) = m$ for all $i = 1, \ldots n$. For time point $j > 1$ consider the edge $e_{j-1} = \{v_a, v_b\}$ with $a < b$. We define $f_i(j)$ as

$$f_i(j) = \begin{cases} f_i(j-1)+1 & \text{if } i = a, \\ f_i(j-1)-1 & \text{if } i = b, \\ f_i(j-1) & \text{otherwise.} \end{cases} \tag{1}$$

Now consider an arbitrary permutation π^V of V. Let π^F be a permutation of F that is defined such that $\text{pos}_{\pi^V}(v_i) = \text{pos}_{\pi^F}(f_i)$ for all $i \in [n]$. We have the following.

▶ **Lemma 18 (★).** *It holds that*

$$\sum_{i=1}^{n}\sum_{j=1}^{m} |W_{i,j}^{\pi^F}|^p = \sum_{\{u,v\}\in E} |pos_\pi(u) - pos_\pi(v)|. \tag{2}$$

Thus, a solution π^V with solution value k w.r.t. MLA corresponds to a solution π^F with solution value k w.r.t. p-WiggleMin. Hence, hardness and inapproximability results of MLA directly carry over to p-WiggleMin. Further, the instance F is balanced, so results also carry over to Weighted-p-WiggleMin by Lemma 5. We obtain the following by the NP-hardness of MLA [6].

▶ **Theorem 19.** *Let $p \geq 1$ be an arbitrary integer. The decision variants of p-WiggleMin and Weighted-p-WiggleMin are strongly NP-complete.*

4.2 Approximation Lower Bounds

In this section, we consider the complexity of approximating p-WiggleMin and Weighted-p-WiggleMin. Firstly, the reduction from the previous section implies two hardness results due to hardness results for MLA shown Ambühl et al. [1] and Raghavendra et al. [15].

▶ **Theorem 20.** *Let $p \geq 1$ be an arbitrary integer. Let $\epsilon > 0$ be an arbitrarily small constant. If there is a PTAS for p-WiggleMin or for Weighted-p-WiggleMin, then there is a (probabilistic) algorithm that decides whether a given SAT instance of size n is satisfiable in time 2^{n^ϵ}.*

Note that it is widely believed that such an algorithm for SAT is unlikely, thus it is unlikely that there exists a PTAS for both problems.

▶ **Theorem 21.** *Let $p \geq 1$ be an arbitrary integer. Under the Small-Set Expansion Hypothesis [14], there is no constant-factor approximation for p-WiggleMin and Weighted-p-WiggleMin.*

The Small-Set Expansion Hypothesis is a hypothesis that implies the Unique-Games Conjecture [11, 14]. It was introduced by Raghavendra and Steurer in 2010 [14] and has since received much attention, as this conjecture would imply some hardness and inapproximability results, including the non-existence of a constant-factor approximation for treewidth [16].

A known greedy heuristic. Next, we want to look at a known greedy heuristic called **BestFirst** for Weighted-1-WiggleMin that has been applied in works on stacked area charts [2,13]. Given an instance $F = \{f_1, \ldots, f_n\}$ of Weighted-1-WiggleMin, the heuristic iteratively builds the order π, starting from the empty order. At step $i = 1, \ldots, n$ of the heuristic, the partial ordering of length $i-1$ is extended by appending a not yet chosen time series to the end that has the smallest increase in the objective value. Such heuristics equivalently exist for 1-WiggleMin and Min-$\sum$|Prefixsum|, we also call them **BestFirst**.

▶ **Theorem 22 (★).** *The **BestFirst** heuristic has approximation factor at least $\Omega(\sqrt[3]{n})$ for Weighted-1-WiggleMin, 1-WiggleMin, and Min-$\sum$|Prefixsum|.*

5 MILP and Experiments for Weighted-1-WiggleMin

Here, we present a mixed-integer linear program (MILP) for Weighted-1-WiggleMin and compare it to the state-of-the art heuristic from Mathiesen and Schulz [13]. This heuristic is different from the **BestFirst** heuristic from the last section and will be explained in more detail later. The aim of this comparison is to determine how an exact approach for Weighted-p-WiggleMin scales with respect to input size and to estimate how the heuristic of [13] compares with an exact algorithm with respect to solution quality. Note that the heuristic we compare with is strictly better than **BestFirst** as was demonstrated in an experimental evaluation by Mathiesen and Schulz [13]. Due to space constraints, the description of the MILP can be found in the full version.

5.1 Experimental Setup

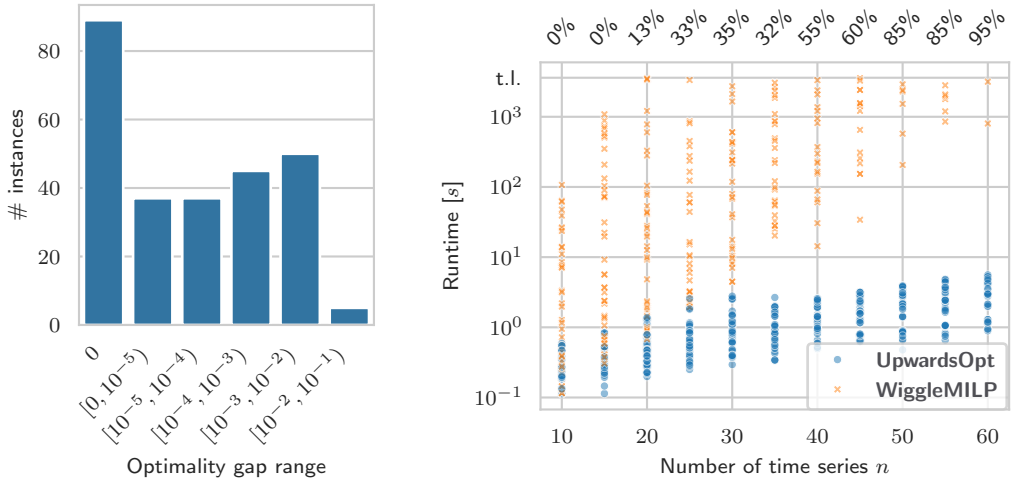
Tested algorithms. We compared two algorithms for Weighted-1-WiggleMin. The first, **WiggleMILP**, is an implementation of the MILP. The second, **UpwardsOpt**, is the best state-of-the art algorithm from [13]. It iteratively performs *moves* that improve the objective value. In a single move, a time series is removed from the current ordering and reinserted into the position that is best with regard to the objective value.

Instances. We obtained instances from real-world data, and these instances were also used in [13]. For example, the eight instances from [13] include unemployment rates of countries for a span of months, Covid values of countries for a span of days, and movie revenues for a span of weeks. These eight instances are very large, with 33–1,000 time series and 33–243 time points. We cannot expect that such instances can be solved to optimality by **WiggleMILP**, thus we sampled from these instances sub-instances as follows. For each instance I and each $k = 10, 15, \ldots, 60$, we picked uniformly at random k time series from I (if I contains at least k time series). These k time series then constitute an instance. We did this five times for each combination of k and I, resulting in 465 instances overall.

Hardware and setup. Both algorithms were implemented in Python 3.12.4. We used the original implementation of **UpwardsOpt** from [13] with the same parameters. **WiggleMILP** was implemented using the Python interface of Gurobi, using Gurobi v11.0.2. Due to large floating point values in the computations, the `NumericFocus` parameter of Gurobi was set to 3. The `MIPGap` parameter was set to 0, in order to find optimal solutions. Without this setup we ran into numeric issues in preliminary experiments. The experiments were run on a cluster using Intel Xeon E5-2640 v4, 2.40GHz 10-core processors, running Ubuntu 18.04.6 LTS. To simulate an end-user machine, the memory limit was set to 8GB and each algorithm was executed on a single thread. The time limit was set to one hour.

5.2 Results

In this section, we answer two questions: first, how does **UpwardsOpt** compare to **WiggleMILP** with respect to the solution quality; second, how scalable are both approaches with respect to the size of an instance. For the first question, we only considered 263 out of 465 instances where **WiggleMILP** produced the optimal solution – it neither timed out nor ran into the memory limit. For each instance where **WiggleMILP** produced the optimal solution, we compute the *optimality gap* g that is defined as $g = \mathrm{sol}_{\textsf{UpwardsOpt}}/\mathrm{sol}_{\textsf{WiggleMILP}} - 1$, where sol_x is the solution value of the algorithm x. Thus, a value of zero corresponds to **UpwardsOpt**



(a) Number of instances by the optimality gap achieved by **UpwardsOpt**.

(b) Scatter plot of runtimes by the number of time series n. The numbers at the top represent the percentage of instances with the respective value n that timed out for **WiggleMILP**.

Figure 3 Results of the experiments.

computing the optimal solution and larger values represent "how far away" **UpwardsOpt** is from the optimum. Figure 3a shows the distribution of optimality gaps. We observe that **UpwardsOpt** could solve 89 out of the 263 instances optimally, while the remaining optimality gaps are relatively low. This confirms that that **UpwardsOpt** is a good heuristic for Weighted-p-WiggleMin for real-world instances.

Next, in Figure 3b we present a scatter plot of runtimes. The y-axis shows runtime in a logscale. The x-axis corresponds to the instance size in terms of the number of time series. Each point in the plot corresponds to an instance-algorithm combination. The values at the top denote the percentage of instances with respective value n that timed out for **WiggleMILP** (the memory limit was never reached). There were no timeouts for **UpwardsOpt**. We observe that **UpwardsOpt** is relatively fast, while **WiggleMILP** runs for much longer and times out already for some instances with 20 time series.

Generally, we conclude that the state-of-the art heuristic **UpwardsOpt** is well-suited for Weighted-1-WiggleMin for real-world instances, even though the problem is computationally very hard in the theoretical sense.

6 Conclusion and Open Problems

We have investigated variants of wiggle minimization in stacked area charts from the theoretical side and showed computational lower bounds. Not only is it NP-hard to minimize wiggle, but it is also unlikely that approximations with good approximation guarantees exist. Nonetheless, we could show in an experimental evaluation that an existing heuristic is very good at minimizing weighted wiggle for real-world instances. Due to this being the first theoretical work on minimizing wiggle in stacked area charts, there remain some open problems.

- Are there constant-factor approximations for Min-$\sum$|Prefixsum|?
- Are there stronger inapproximability results for p-WiggleMin and Weighted-p-WiggleMin, i.e., under the assumption that P $\neq$ NP.

- We think that improvements to **WiggleMILP** are possible, and it might be interesting to investigate other exact wiggle minimization approaches.
- Finally, our results might be extended to wiggle minimization in streamgraphs. It seems likely that the hardness results in this paper carry over to streamgraphs, though, we were not able to show that yet.

References

1 Christoph Ambühl, Monaldo Mastrolilli, and Ola Svensson. Inapproximability results for maximum edge biclique, minimum linear arrangement, and sparsest cut. *SIAM J. Comput.*, 40(2):567–596, 2011. doi:10.1137/080729256.

2 Marco Di Bartolomeo and Yifan Hu. There is more to streamgraphs than movies: Better aesthetics via ordering and lassoing. *Comput. Graph. Forum*, 35(3):341–350, 2016. doi:10.1111/CGF.12910.

3 Chuan Bu, Quanjie Zhang, Qianwen Wang, Jian Zhang, Michael Sedlmair, Oliver Deussen, and Yunhai Wang. Sinestream: Improving the readability of streamgraphs by minimizing sine illusion effects. *IEEE Trans. Vis. Comput. Graph.*, 27(2):1634–1643, 2021. doi:10.1109/TVCG.2020.3030404.

4 Lee Byron and Martin Wattenberg. Stacked graphs - geometry & aesthetics. *IEEE Trans. Vis. Comput. Graph.*, 14(6):1245–1252, 2008. doi:10.1109/TVCG.2008.166.

5 Alexander Dobler and Martin Nöllenburg. On minimizing wiggle in stacked area charts. *CoRR*, abs/2506.21175, 2025. doi:10.48550/arXiv.2506.21175.

6 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

7 Nicolas Greffard and Pascale Kuntz. Visualizing a set of multiple time series with an aggregate stacked graph. In Ebad Banissi, Mark W. McK. Bannatyne, Fatma Bouali, Remo Burkhard, John Counsell, Urska Cvek, Martin J. Eppler, Georges G. Grinstein, Weidong Huang, Sebastian Kernbach, Chun-Cheng Lin, Feng Lin, Francis T. Marchese, Chi Man Pun, Muhammad Sarfraz, Marjan Trutschl, Anna Ursyn, Gilles Venturini, Theodor G. Wyeld, and Jian J. Zhang, editors, *Proc. International Conference on Information Visualisation (IV'2015)*, pages 68–74. IEEE Computer Society, 2015. doi:10.1109/IV.2015.23.

8 Yutian He and Hongjun Li. Optimal layout of stacked graph for visualizing multidimensional financial time series data. *Inf. Vis.*, 21(1):63–73, 2022. doi:10.1177/14738716211045005.

9 Renée Huggett. Multiple line graphs (2). In *Graphs and Charts*, pages 43–46. Palgrave Macmillan UK, London, 1990. doi:10.1007/978-1-349-11245-6_10.

10 Hans Kellerer, Vladimir Kotov, Franz Rendl, and Gerhard J. Woeginger. The stock size problem. *Oper. Res.*, 46(3-Supplement-3):S1–S12, 1998. doi:10.1287/OPRE.46.3.S1.

11 Subhash Khot. On the power of unique 2-prover 1-round games. In John H. Reif, editor, *Proc. ACM Symposium on Theory of Computing (STOC'2002)*, pages 767–775. ACM, 2002. doi:10.1145/509907.510017.

12 Tsai Li-Hui. Sequencing to minimize the maximum renewal cumulative cost. *Oper. Res. Lett.*, 12(2):117–124, 1992. doi:10.1016/0167-6377(92)90073-C.

13 Steffen Strunge Mathiesen and Hans-Jörg Schulz. Aesthetics and ordering in stacked area charts. In Amrita Basu, Gem Stapleton, Sven Linker, Catherine Legg, Emmanuel Manalo, and Petrucio Viana, editors, *Proc. Diagrammatic Representation and Inference (Diagrams'2021)*, volume 12909 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2021. doi:10.1007/978-3-030-86062-2_1.

14 Prasad Raghavendra and David Steurer. Graph expansion and the unique games conjecture. In Leonard J. Schulman, editor, *Proc. ACM Symposium on Theory of Computing (STOC'2010)*, pages 755–764. ACM, 2010. doi:10.1145/1806689.1806792.

15 Prasad Raghavendra, David Steurer, and Madhur Tulsiani. Reductions between expansion problems. In *Proc. Conference on Computational Complexity (CCC'2012)*, pages 64–73. IEEE Computer Society, 2012. doi:10.1109/CCC.2012.43.

16 Yu (Ledell) Wu, Per Austrin, Toniann Pitassi, and David Liu. Inapproximability of treewidth and related problems. *J. Artif. Intell. Res.*, 49:569–600, 2014. doi:10.1613/JAIR.4030.

17 Wenci Yu, Han Hoogeveen, and Jan Karel Lenstra. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard. *J. Sched.*, 7(5):333–348, 2004. doi:10.1023/B:JOSH.0000036858.59787.C2.

Novel Complexity Results for Temporal Separators with Deadlines

Riccardo Dondi
Università degli Studi di Bergamo, Italy

Manuel Lafond
Université de Sherbrooke, Canada

Abstract

We consider two variants, (s,z,ℓ)-Temporal Separator and (s,z,ℓ)-Temporal Cut, respectively, of the vertex separator and the edge cut problem in temporal graphs. The goal is to remove the minimum number of vertices (temporal edges, respectively) in order to delete all the temporal paths that have time travel at most ℓ between a source vertex s and target vertex z. First, we solve an open problem in the literature showing that (s,z,ℓ)-Temporal Separator is NP-hard even when the underlying graph has pathwidth bounded by four. We complement this result showing that (s,z,ℓ)-Temporal Separator can be solved in polynomial time for graphs of pathwidth bounded by three. Then we consider the approximability of (s,z,ℓ)-Temporal Separator and we show that it cannot be approximated within factor $2^{\Omega(\log^{1-\varepsilon}|V|)}$ for any constant $\varepsilon > 0$, unless $NP \subseteq ZPP$ (V is the vertex set of the input temporal graph) and that the strict version is approximable within factor $\ell - 1$ (we show also that it is unliklely that this factor can be improved). Then we consider the (s,z,ℓ)-Temporal Cut problem, we show that it is APX-hard and we present a $2\log_2(2\ell)$ approximation algorithm.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph theory

Keywords and phrases Temporal Graphs, Graph Algorithms, Graph Separators, Parameterized Complexity, Approximation Complexity

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.23

1 Introduction

A central problem in checking network robustness is finding the minimum number of vertices or edges that need to be removed in order to disconnect the network. In classic (static) graphs this is modeled by computing a minimum cut or a minimum vertex separator between a source vertex s and a target vertex z. The static graph model however does not consider how a network may change over time. In transportation networks, for example, the time schedule is a fundamental aspect that has to be taken into account for analyzing several properties, like connectedness and robustness. The need to incorporate time information of edge availability has led to the introduction of the *temporal graph* model [12, 9, 16, 10], where edges are assigned timestamps in a discrete set that define when each edge is available (thus when a specific transport is available in a transportation network).

The robustness problems (minimum cut and minimum vertex separator) have been considered also for temporal graphs, where the paths to be removed have to be *temporal*, that is they have to satisfy a time constraint. Given a source s and a target z, the (s,z)-Temporal Cut problem asks for the minimum number of temporal edges that have to be removed so that s and z are disconnected, while (s,z)-Temporal Separator asks for the minimum number of vertices that have to be removed so that s and z are disconnected. (s,z)-Temporal Cut is known to be in solvable in polynomial time [2], (s,z)-Temporal Separator is known NP-hard [12, 17], and its fixed-parameter tractability and approximability (and variants thereof) have been studied [7, 15, 8, 14, 11].

© Riccardo Dondi and Manuel Lafond;
licensed under Creative Commons License CC-BY 4.0

A variant of the (s,z)-Temporal Separator problem that has been considered in [8] to model the robustness of transportation system, defines s and z as separated if the time needed to move from s to z is above a time threshold. The motivation is that if the time to move from s to z is increased considerably, then this makes the possibility of moving from s to z unlikely. This is modeled in [8] by defining the (s,z,ℓ)-Temporal Separator problem, that asks for a smallest subset of vertices whose removal deletes each temporal path that takes at most time ℓ between s and z. Several results have been given for (s,z,ℓ)-Temporal Separator in [8]. The (s,z,ℓ)-Temporal Separator problem is NP-hard when $\ell = 1$ and the temporal graph is defined over (at least) two timestamps. On the other hand, the problem is solvable in polynomial time when the underlying graph is a tree (after deleting s and z), or it has branchwidth at most two. As for the approximation complexity, it is shown to be not approximable within factor $\Omega(\ln|V| + \ln\tau)$ assuming that $NP \not\subset DTIME(|V|^{\log\log|V|})$ (V is the set of vertices of the temporal graph, τ the number of timestamps); moreover, a τ^2-approximation algorithm is presented (also a τ-approximation algorithm for (s,z)-Temporal Separator). Finally, it is shown that solving (s,z,ℓ)-Temporal Separator when the underlying graph has bounded pathwidth is at least as difficult as solving a problem called Discrete Segment Covering, whose complexity is unsolved [1]. In Section 3, we settle the status of (s,z,ℓ)-Temporal Separator on graphs of bounded pathwidth: we show that the decision version of problem is NP-hard even with $\ell = 1$ and when the underlying graph has pathwidth at most 4, and we give a polynomial-time algorithm when the pathwidth is at most 3. Then in Section 4, we show that (s,z,ℓ)-Temporal Separator cannot be approximated within factor $2^{\Omega(\log^{1-\varepsilon}|V|)}$ for any constant $\varepsilon > 0$, unless $NP \subseteq ZPP$ and we present an $\ell - 1$-approximation algorithm for the strict variant of (s,z,ℓ)-Temporal Separator[1]. We show also that improving this factor is a challenging problem, since the strict variant of (s,z,ℓ)-Temporal Separator is hard to approximate as Vertex Cover k-Uniform Hypergraph, where $k = \ell - 1$. In Section 5, we consider the (s,z,ℓ)-Temporal Cut problem and we show that it is APX-hard, which contrasts with the polynomial-time solvability of the problem when deadlines are not considered, and we present a $2\log_2(2\ell)$-approximation algorithm. In Section 2 we give some definitions and we formally define the two problems we are interested into. Some of the proofs are omitted due to space constraint.

2 Preliminaries

For an integer n, we use the notation $[n] = \{1, 2, \ldots, n\}$. A *temporal graph* $G = (V, E, \tau)$ is defined over a set V of vertices and a set $E \subseteq V \times V \times [\tau]$ of temporal edges, where $\tau \in \mathbb{N}$. An *undirected* edge in a temporal graph is then a triple (u, v, t), where $u, v \in V$ and $t \in [\tau]$ is a timestamp[2]. Note that we denote by uv an edge in an undirected (static) graph and (u, v) an arc in a directed (static) graph.

We say that u, v are *neighbors* in G if they share some temporal edge, and denote the set of neighbors of u by $N_G(u)$ (we drop the subscript if it is clear from the context). Given a set $V' \subseteq V$, we denote by $G[V']$ the subgraph induced by V', which contains vertex set V' and every temporal edge whose two endpoints are in V'. We also denote by $G - V' = G[V \setminus V']$ the temporal graph obtained by removing each vertex in V'. Given a set $E' \subseteq E$, we denote by $G - E'$ the temporal graph obtained by removing each temporal edge in E'.

[1] In the strict variant a temporal path must uses temporal edges with increasing timestamps

[2] As in [8] we assume that (u, v, t) is a temporal edge of G if and only if (v, u, t) is a temporal edge of G.

An interval $[t_1, t_2]$, with $t_1, t_2 \in [\tau]$ and $t_1 \leq t_2$, is the sequence of consecutive timestamps between t_1 and t_2, including t_1 and t_2. Given interval $[t_1, t_2]$, $G([t_1, t_2])$ is the temporal subgraph of G that has temporal edges having timestamps in $[t_1, t_2]$.

A *temporal path* P in a temporal graph G is a sequence of temporal edges such that

$$P = [(u_1, v_1, t_1), (u_2, v_2, t_2), \ldots, (u_h, v_h, t_h)],$$

with $v_i = u_{i+1}$ and $t_i \leq t_{i+1}$, for each $i \in [h-1]$, and $u_i \neq u_j$, $v_i \neq v_j$, for each $i, j \in [h]$ with $i \neq j$; P is called an (u_1, v_1)-temporal path, since it starts from vertex u_1 and ends in vertex v_h. If $t_i < t_{i+1}$, with $i \in [h-1]$, then the temporal path is *strict*. Given two vertices s and z Given a temporal path $P = [(u_1, v_1, t_1), (u_2, v_2, t_2), \ldots, (u_h, v_h, t_h)]$, the *travelling time* $tt(P)$ of P is defined as $tt(P) = t_h - t_1 + 1$. The set of vertices of a temporal path P is denoted by $V(P)$. We may refer to a temporal path of travelling time *at most* ℓ as an ℓ-temporal path. Two temporal paths are *temporal edge disjoint* if they don't share any temporal edge.

Given a temporal graph $G = (V, E, \tau)$, we assume that there are two special vertices $s, z \in V$, which are the source and the target vertex of G. A set $V' \subseteq (V \setminus \{s, z\})$ is an (s, z)-temporal separator ((s, z)-strict temporal separator, respectively) in G if there is no temporal path (strict temporal path, respectively) in $G[V \setminus V']$ between s and z. Given $\ell \in [\tau]$, a set $V' \subseteq (V \setminus \{s, z\})$ is an (s, z, ℓ)-temporal separator ((s, z, ℓ)-strict temporal separator, respectively) in G if there is no temporal path (strict temporal path, respectively) between s and z in $G[V \setminus V']$ of travelling time at most ℓ.

A set $E' \subseteq E$ of temporal edges is an (s, z)-temporal cut ((s, z)-strict temporal cut, respectively) in G if there is no temporal path (strict temporal path, respectively) in $G - E'$ between s and z. Given $\ell \in [\tau]$ a set $E' \subseteq E$ is an (s, z, ℓ)-temporal cut ((s, z, ℓ)-strict temporal cut, respectively) in G if there is no temporal path (strict temporal path, respectively) in $G - E'$ between s and z of travelling time at most ℓ.

Now, we are ready to define the combinatorial problems we are interested in. The first, introduced in [8], is the following.

▶ **Problem 1** ((s,z,ℓ)-Temporal Separator).
Input: *a temporal graph $G = (V, E, \tau)$, two vertices $s, z \in V$, a positive integer $\ell \in [\tau]$.*
Output: *An (s,z,ℓ)-temporal separator $V' \subseteq (V \setminus \{s, z\})$ in G of minimum size.*

We denote by (s,z,ℓ)-Strict Temporal Separator the variant of (s,z,ℓ)-Temporal Separator where we look for an (s,z,ℓ)-strict temporal separator.

We consider now the second problem we are interested into.

▶ **Problem 2** ((s,z,ℓ)-Temporal Cut).
Input: *a temporal graph $G = (V, E, \tau)$, two vertices $s, z \in V$, a positive integer $\ell \in [\tau]$.*
Output: *An (s,z,ℓ)-temporal cut $E' \subseteq E$ in G of minimum size.*

Note that if $\ell = \tau$, then (s,z,ℓ)-Temporal Separator ((s,z,ℓ)-Temporal Cut, respectively) is exactly the (s,z)-Temporal Separator problem (the (s,z)-Temporal Cut) problem, respectively) where we look for an (s, z)- temporal separator ((s, z)-temporal cut, respectively) in G.

3 Graphs of bounded pathwidth

Let us first recall the notion of pathwidth. A *nice path decomposition* of a graph G is a sequence of set of vertices $(X_1, \ldots, X_q)$, with each $X_i \subseteq V(G)$ referred to as a *bag*, such that all of the following holds:

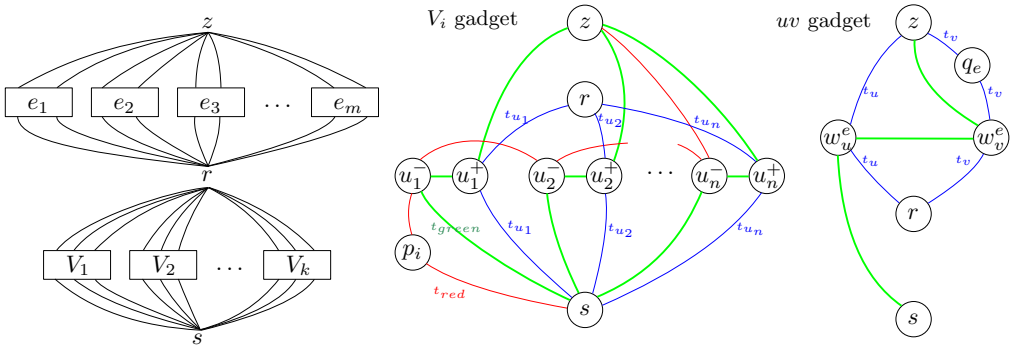


Figure 1 Top left: main structure of the reduction: there is a vertex-chossing phase leading to paths to r, followed by an edge vertification phase (note, not all edges are shown). Middle: a V_i gadget, with t_{green} edges in green, t_{red} edges in red, and t_u edges in blue. Right: an edge gadget for $e = uv$, with t_{green} edges in green and t_u, t_v edges in blue.

- $X_1 = X_q = \emptyset$;
- for $i \in \{2, \ldots, q\}$, either $X_i = X_{i-1} \cup \{v\}$, with $v \notin X_{i-1}$, in which case X_i is called an *introduce bag*; or $X_i = X_{i-1} \setminus \{v\}$, with $v \in X_{i-1}$, in which case X_i is a *forget bag*.
- for every pair of vertices u and v that share an edge in G, some bag X_i contains both u and v (regardless of the time of the edge).
- for every vertex $v \in V(G)$, there are $i, j \in [q-1]$ such that the set of bags that contain v is precisely $X_i, X_{i+1}, \ldots, X_j$.

The *width* of the nice path decomposition is $\max_{i \in [q-1]}(|X_i| - 1)$. The *pathwidth* of G is the minimum width of a nice path decomposition of G.

We show that (s,z,ℓ)-Temporal Separator is NP-hard even on graphs of pathwidth 4 and when $\ell = 1$. Our reduction is from the Multicolored Independent Set problem, where we are given a (static) graph G and a partition $\{V_1, \ldots, V_k\}$ of $V(G)$ into k sets. The V_i sets are called *color classes*. The question is whether there is an independent set I of G such that $|I \cap V_i| = 1$ for each $i \in [k]$, that is, we must choose one vertex per color class to form an independent set. Note that this problem is typically used to prove W[1]-hardness, but it is also NP-hard [6][3].

Let G be an instance of Multicolored Independent Set, with vertex partition $V_1, \ldots, V_k$. We assume, without loss of generality, that $|V_i| = n$ for every $i \in [k]$.

Let us construct from G an instance of the (s,z,ℓ)-Temporal Separator problem consisting of temporal graph H, vertices s and z to separate, and $\ell = 1$. In the construction, we assign to temporal edges of H a time in the set $T = \{t_{green}, t_{red}\} \cup \{t_u : u \in V(G)\}$, with the understanding that all elements of T correspond to a distinct integer. Since $\ell = 1$, one can view the temporal edges as being colored by an element of T and the problem as having to destroy all monochromatic paths – so the subscripts of the elements of T can be seen as colors. So from G, first add to $V(H)$ the vertices s, z, and a new vertex r. We then have vertex-choosing gadgets and edge-verification gadgets (see Figure 1 for an illustration).

[3] Note that the problem considered in [6], Multicolored Clique, is equivalent to Multicolored Independent Set if we consider complementary relations (edges and no edges) between vertices of different color classes of the input graph.

Vertex-choosing gadgets. For each $i \in [k]$, build a gadget for V_i as follows. For every $u \in V_i$, add to H two vertices u^- and u^+. Then add a temporal path at time t_{green} from s to z formed by the temporal edges $(s, u^-, t_{green}), (u^-, u^+, t_{green}), (u^+, z, t_{green})$. Also add a temporal path at time t_u from s to r formed by the temporal edges $(s, u^+, t_u), (u^+, r, t_u)$.

To complete the gadget, add a new vertex p_i, then add temporal edges so that there is a path with a time of t_{red} from s to z that first goes through p_i, then through all the u^- vertices exactly once. More precisely, denote $V_i = \{u_1, \ldots, u_n\}$, where the ordering is arbitrary. Then, add the temporal edges $(s, p_i, t_{red}), (p_i, u_1^-, t_{red})$ and (u_n^-, z, t_{red}), and for each $h \in [n-1]$, add the temporal edge $(u_h^-, u_{h+1}^-, t_{red})$.

Edge verification gadgets. Next, we construct gadgets to verify that the chosen vertices correspond to an independent set of G. For each edge $e = uv$ of G, where $u \in V_i$ and $v \in V_j$ such that $i < j$, add to H two vertices w_u^e and w_v^e. Add a temporal path at time t_{green} from s to z formed by the temporal edges $(s, w_u^e, t_{green}), (w_u^e, w_v^e, t_{green})$, and (w_v^e, z, t_{green}), enforcing the deletion of at least one of the two vertices. Next, add a temporal path at time t_u formed by the temporal edges $(r, w_u^e, t_u), (w_u^e, z, t_u)$. Finally, add a new vertex q_e and a temporal path at time t_v going from r to z formed by the temporal edges $(r, w_v^e, t_v), (w_v^e, q_e, t_v), (q_e, z, t_v)$.

▶ **Lemma 1.** *The graph H constructed from G as described above has pathwidth at most* 4.

The idea is that we can get a path decomposition by putting s, z, r in every bag. Then, the V_i and the uv gadgets are easy to construct using two extra vertices per bag. We next show that NP-hardness holds.

▶ **Lemma 2.** *The graph G has a multicolored independent set if and only if H has an $(s, z, 1)$-temporal separator of size $|V(G)| + |E(G)|$.*

Proof sketch. Let $I = \{u_1, \ldots, u_k\}$ be a multicolored independent set of G, with each $u_i \in V_i$. In H, in the V_i gadget, delete every u^+ vertex, *except* u_i^+, and delete u_i^- instead. This removes all the t_{green} and the t_{red} temporal paths, but s can reach r with a temporal path at time t_{u_i}. Then for each edge $e = uv$ of G, in the uv gadget, delete w_u^e if s reaches r with time t_u, and delete w_v^e otherwise. This removes the t_{green} temporal path, and since we delete at least one of u^+ or v^+, there remains no temporal path at time t_u or t_v going through the gadget.

Conversely, suppose there is an $(s, z, 1)$-temporal separator in H of size $|V(G)| + |E(G)|$. The t_{green} temporal paths enforce deleting, for each $u \in V(G)$, one of u^- or u^+, and also for each $e = uv \in E(G)$ one of w_u^e or w_v^e. There is no room for other deletions. Because of the t_{red} temporal path in the V_i gadget, some u^- is deleted and some u^+ is kept. Also, we cannot keep u^+ and v^+ from different V_i, V_j gadgets if $uv \in E(G)$, as otherwise there will be a t_u or t_v temporal path going through the uv gadget. Hence, the kept u^+ vertices correspond to a multicolored independent set. ◀

▶ **Theorem 3.** *The (s,z,ℓ)-Temporal Separator problem is NP-hard, even with $\ell = 1$ and on temporal graphs of pathwidth at most* 4*, and even if each edge is present in only one timestamp.*

We mention that it should be possible to modify the proof to prove a similar hardness for the *strict* variant of the problem, for larger values of ℓ, as it suffices to replace each label t_{green}, t_{red}, t_u with time intervals that are far enough from each other. This modification requires a large value of ℓ as the red path in our construction traverses $n + 1$ vertices, in addition to s and z. We leave the details for a future version, and the complexity of the strict variant for fixed pathwidth and ℓ remains open.

Graphs of pathwidth 3

Here we show that (s,z,ℓ)-Temporal Separator can be solved in polynomial time on graphs of pathwidth 3, for any ℓ, which shows that the above hardness is tight, assuming $P \neq NP$. Note that now, we allow temporal edges to have multiple activation times, and we allow them to be directed or not. Let G be a temporal graph. We first apply two reduction rules, which can easily seen to be safe, that is, a solution of size at most k exists in the temporal graph before the application of the rule if and only if it exists also after its application.

▶ **Rule 1.** *If G has a vertex v such that $(s, v, t)(v, z, t')$ is an ℓ-temporal path, then delete v.*

▶ **Rule 2.** *If $G - \{s, z\}$ has multiple connected components $C_1, C_2, \ldots, C_p$, then solve each subgraph $G[C_1 \cup \{s, z\}], \ldots, G[C_p \cup \{s, z\}]$ separately.*

We say that G is *clean* if none of the above rules is applicable to G. Note that for $n = |V(G)|$ and $m = |E(G)|$ (note that $E(G)$ is the set of temporal edges of G), one can determine in time $O(n + m)$ whether one of the rules applies, and each rule can be applied at most n times, and so a graph can be made clean in time $O(n^2 + nm)$. Note, for Rule 1 we assume that the time between the consecutive temporal arcs $(s, v, t), (v, z, t')$ can be compared in constant time, and Rule 2 does not use arc times, so there is no time dependency on τ.

We now turn to (clean) graphs of pathwidth at most 3. We assume that we have checked that G has pathwidth at most 3 and we can compute a decomposition of G in $O(n)$ time [3]. The idea is to first check whether there is an (s, z)-separator of size at most 3 (so, a separator that ignores edge times). If there is one, then there is an (s, z, ℓ)-temporal separator of size at most 3. In this case we can compute a solution by trying every combination of at most three vertices. If there is no such separator, we can rely on the structural lemma below, which is illustrated in Figure 2. First, we can show that there is a sequence of bags that contain both s and z; this allows to find B, consisting of the vertices introduced and forgotten within this sequence. This B induces a caterpillar[4], as removing s and z from these bags yields a subgraph of pathwidth at most 2. The vertices other than s or z in bags that precede this sequence form A, and the vertex s or z introduced last has at most one neighbor in A (e.g., in Figure 2, z has only y as a neighbor in A). Analogously, C contains vertices of bags that occur after the sequence.

▶ **Lemma 4.** *Suppose that G is clean, has pathwidth at most 3, and has no (s, z)-separator of size 3 or less. Then in $O(n^4 m)$ time we can compute a partition of $V(G) \setminus \{s, z\}$ into three non-empty sets A, B, C such that all of the following holds:*

- *either s has at most one neighbor in A (resp. C), or z has at most one neighbor in A (resp. C).*
- *the subgraph induced by B is a caterpillar with main path $w_1 - w_2 - \ldots - w_k$. Moreover, w_1 is the only vertex of $B \cup C$ whose neighborhood can intersect with A, and w_k is the only vertex of $B \cup A$ whose neighborhood can intersect with C.*

One can infer from this structure that "most" deletions occur on the caterpillar path.

[4] Recall that a *caterpillar* is a tree in which there is a *main path* $w_1 - \ldots - w_k$, and every vertex not on that path is a leaf adjacent to a vertex of the path.

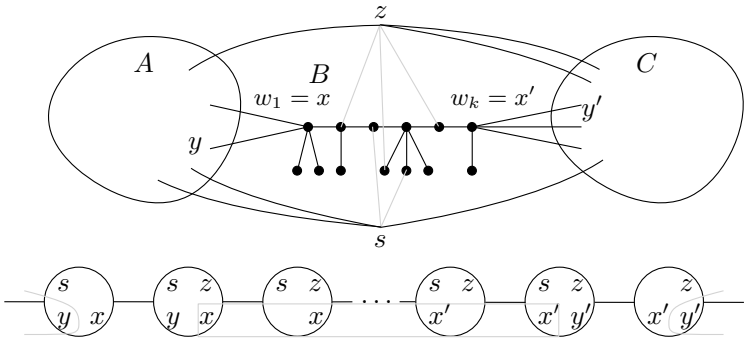


Figure 2 An illustration of the structure described by Lemma 4. The top part shows the temporal graph, the bottom part shows the bags that contain both s and z (the left and right areas represent A and C, and the middle box represents B).

▶ **Lemma 5.** *Suppose that G is clean and that $V(G) \setminus \{s, z\}$ can be partitioned into the sets A, B, C as in Lemma 4. Then there exists a minimum (s, z, ℓ)-temporal separator in which at most one vertex of A is deleted, at most one vertex of C is deleted, and all other deleted vertices are on the main path $w_1 - \ldots - w_k$ of the B caterpillar.*

Algorithm 1 Main algorithm.

```
1  function getTemporalSeparator(G, s, z, (A, B, C))
2      //We assume that G is clean
3      Let 𝒟 = {D ⊆ V(G) : D ∩ A ≤ 1, D ∩ C ≤ 1, and D ∩ B = ∅}
4      for each D ∈ 𝒟 do
5          D' = extendSeparator(G, s, z, (A, B, C), D)        // pass copy of D
6      end
7      return a smallest (s, z, ℓ)-separator D' found
8
9  function extendSeparator(G, s, z, (A, B, C), D)
10     if G − (D ∪ B) has an (s, z, ℓ)-temporal path then
11         return "impossible"
12     Let w_1 − ... − w_k be the main path of the caterpillar B
13     for i = 1, 2, ..., k − 1 do
14         Define W_i = A ∪ {s, z} ∪ (N[w_1, ..., w_i] \ {w_{i+1}})
15         if G[W_i] − D contains an (s, z, ℓ)-temporal path then
16             Add w_i to D
17     end
18     if G − D contains an (s, z, ℓ)-temporal path then
19         Add w_k to D
20     return D;
```

Our algorithm proceeds as follows. We first make G clean, possibly handling multiple connected components of $G - \{s, z\}$ separately, and construct the sets A, B, C from Lemma 4. We then run Algorithm 1, which first guesses every way that a solution could delete at most one vertex from A and at most one vertex from C, noting that such a solution exists by Lemma 5. There are $O(n^2)$ possible guesses, and for each of them, we solve a restricted version of the problem where we can only delete vertices from the caterpillar B.

This is done by the *extendSeparator* function, which attempts to extend the guess by finding the minimum number of deletions to do on the main B path $w_1 - \ldots - w_k$, traversing it from left to right. When at a specific w_i, we restrict the graph to $G[W_i] - D$, where D contains the deletions made so far and W_i contains A, w_i and the predecessors of w_i, and their neighbors. In a greedy manner, we delete w_i only if it creates a temporal path in this restricted graph, considering the deletions D applied so far. We can apply a greedy approach, since we can show that no vertex in $N(w_i)$ is removed, if needed w_i is removed instead. This ensures that there is no temporal path that goes through the w_i's in increasing order (possibly going in A as well), and we make one last check at the end to ensure that no temporal path goes through C and then on the path in the reverse order (if so, we delete w_k to prevent that).

▶ **Theorem 6.** *The (s,z,ℓ)-Temporal Separator problem can be solved in time $O(n^4 m)$ on graphs on pathwidth at most 3, where $n = |V(G)|$ and $m = |E(G)|$.*

Note that the above algorithm works for the strict variant of the problem. Indeed, the algorithm only needs to query whether a forbidden path exists, so it suffices to have access to a routine that determines whether a strict ℓ-temporal path exists. Again we leave the details for a future version.

4 Approximation of (s,z,ℓ)-Temporal Separator

4.1 Hardness of Approximation

In this section we strengthen the inapproximability of (s,z,ℓ)-Strict Temporal Separator (and later we extend the inapproximability to (s,z,ℓ)-Temporal Separator). We prove the result by giving an approximation preserving reduction from Directed Multicut to (s,z,ℓ)-Strict Temporal Separator. Directed Multicut is known to be inapproximable within factor $2^{\Omega(\log^{1-\varepsilon}|N|)}$, for any constant $\varepsilon > 0$, even for a directed acyclic graph with a set N of vertices, unless $NP \subseteq ZPP$ [4]. We recall here that Directed Multicut (we consider it is defined on a directed acyclic graph).

▶ **Problem 3** (Directed Multicut).
Input: *given a directed acyclic graph $D = (N, A)$, a set R of pairs $(s_1, z_1), \ldots, (s_h, z_h)$ of vertices, with $s_i, z_i \in N$, $i \in [h]$.*
Output: *asks for a minimum cardinality subset $A' \subseteq A$ so that each pair (s_i, z_i), $i \in [h]$, is disconnected in $D - A'$.*

Note that we assume that, for each $(s_i, z_i) \in R$, $i \in [h]$, it holds that $s_i \neq t_i$, otherwise the pair cannot be separated by edge removal. Given an instance (D, R) of Directed Multicut, the vertices that belong to a pair in R are called terminals; the set R_S (R_Z, respectively) contains those terminals v such that $v = s_i$ and $(s_i, z_j) \in R$ ($v = z_j$ and $(s_i, z_j = v) \in R$, respectively).

Consider an instance (D, R) of Directed Multicut, in the following we construct a corresponding instance of (s,z,ℓ)-Strict Temporal Separator. We first present the idea of the construction. Essentially G contains $|A| + 1$ copies of each vertex, plus one vertex for each arc in A, in addition to s and z. This ensures that only the vertices associated with arcs will be removed, as removing $|A| + 1$ copies of each vertex of D requires to delete too many vertices. As for the temporal edges of G, for each arc from u to v in D, there is a path of length two, consisting of a temporal edge from the vertex associated with u to vertex associated with arc (u, v) and a temporal edge from the vertex associated with

arc (u, v) to the vertex associated with v. Each vertex v_i is associated with a interval of ℓ timestamps $[\ell \cdot (2i-2), \ell \cdot (2i-1) - 1]$. The timestamps are assigned to temporal edges of G so that if there exists a path in D from v_i to v_j, then there exists a temporal path from two corresponding vertices of G, that is $x_{i,q}$ to $x_{j,r}$, $q, r \in [|A| + 1]$, with timestamps in the interval $[\ell \cdot (2i-2), \ell \cdot (2i-1) - 1]$.

Now, we present the formal definition of the instance of (s,z,ℓ)-Strict Temporal Separator. First, $\ell = 2|N|$. Given $N = \{v_1, \ldots, v_{|N|}\}$, since D is a DAG, we assume that the vertices of D are sorted according to a topological sorting of D, that is for $(v_i, v_j) \in A$, $i, j \in [|N|]$, it holds that $i < j$. We define the set V of vertices:

$$V = \{x_{i,q} : v_i \in N, q \in [|A| + 1]\} \cup \{y_{i,j} : (v_i, v_j) \in A, i < j\} \cup \{s, z\}.$$

Now, we define the set of temporal edges. For a vertex $v_i \in N$, $i \in [|N|]$, $Reach(D, v_i)$ denotes the set of vertices that can be reached with a path that starts from v_i in D.

$$\begin{aligned} E = &\{(s, x_{i,q}, \ell \cdot (2i-2)) : v_i \in R_S, q \in [|A| + 1]\} \cup \\ &\{(x_{j,q}, z, \ell \cdot (2i-1) - 1) : v_j \in R_Z, v_j \in Reach(D, v_i), \exists (v_i, v_j) \in R, q \in [|A| + 1]\} \cup \\ &\{(x_{i,q}, y_{i,j}, \ell \cdot (2h-2) + 2i - 1) : v_i \in N, v_i \in Reach(D, v_h), (v_i, v_j) \in A, h \in [i], q \in [|A| + 1]\} \cup \\ &\{(y_{i,j}, x_{j,q}, \ell \cdot (2h-2) + 2(j-1)) : v_j \in N, v_j \in Reach(D, v_h), (v_i, v_j) \in A, h \in [i], q \in [|A| + 1]\} \end{aligned}$$

Now, we prove the relations between Directed Multicut and (s,z,ℓ)-Strict Temporal Separator.

▶ **Lemma 7.** *Consider an instance (D, R) of Directed Multicut and a the instance (G, s, t, ℓ) of (s,z,ℓ)-Strict Temporal Separator, computed by the construction above. Then (1) given a solution A' of Directed Multicut on instance (D, R) consisting of k arcs we can compute in polynomial time a solution of (s,z,ℓ)-Strict Temporal Separator on instance (G, s, t, ℓ) consisting of k vertices; (2) given a solution of (s,z,ℓ)-Strict Temporal Separator on instance (G, s, z, ℓ) consisting of k vertices we can compute in polynomial time a solution A' of Directed Multicut on instance (D, R) consisting of most k arcs.*

Proof sketch. For (1), given a solution A' of Directed Multicut on instance (D, R) consisting of k arcs, we can compute in polynomial time a solution of (s,z,ℓ)-Strict Temporal Separator on the corresponding instance (G, s, t, ℓ) as follows: $V' = \{y_{i,j} : (v_i, v_j) \in A'\}$. Indeed, if $G[V \setminus V']$ contains a strict temporal path p it must pass through two vertices $x_{i,r}, x_{j,r}$ and there is a path between the corresponding vertices v_i, v_j of $D - A'$ such that $(v_i, v_j) \in R$. For (2), given a solution V' of (s,z,ℓ)-Strict Temporal Separator on instance (G, s, t, ℓ), where $|V'| = k$, first we can show that that there is a solution V^* of (s,z,ℓ)-Strict Temporal Separator on instance (G, s, t, ℓ), with $|V^*| \leq k$, that contains only vertices $y_{i,j}$. We can define a solution A' of Directed Multicut on instance (D, R) as follows $A' = \{(v_i, v_j) : y_{i,j} \in V^*\}$. ◀

It follows from Lemma 7 that we have designed an approximation preserving reduction from Directed Multicut to (s,z,ℓ)-Strict Temporal Separator. Since Directed Multicut is not approximable within factor $2^{\Omega(\log^{1-\varepsilon} |N|)}$, for any constant $\varepsilon > 0$, unless $NP \subseteq ZPP$ [4], we have the following result.

▶ **Theorem 8.** *The (s,z,ℓ)-Strict Temporal Separator problem is $2^{\Omega(\log^{1-\varepsilon} |V|)}$-hard to approximate for any constant $\varepsilon > 0$, unless $NP \subseteq ZPP$.*

The same result holds also for (s,z,ℓ)-Temporal Separator, we need to slightly modify the input temporal graph so that each ℓ-temporal path is forced to be strict.

▶ **Corollary 9.** *The (s,z,ℓ)-Temporal Separator problem is* $2^{\Omega(\log^{1-\varepsilon}|V|)}$*-hard to approximate for any constant* $\varepsilon > 0$*, unless* $NP \subseteq ZPP$*.*

4.2 Approximating (s,z,ℓ)-Strict Temporal Separator

We consider now the (s,z,ℓ)-Strict Temporal Separator problem and we present an $(\ell-1)$-approximation algorithm for it. First, we prove the following easy claim.

▷ Claim 10. A strict ℓ-temporal path between s and z contains at most $\ell-1$ vertices different from s and z.

Now, we present the approximation algorithm (Algorithm 2). The algorithm greedily looks for a strict ℓ-temporal path P between s and z. If there exists such a P, Algorithm 2 removes all the vertices of P from G, except for s and z, and adds its vertices to the vertex separator V'. If there exists no such path, then the algorithm stops and return V'.

Algorithm 2 The approximation algorithm for (s,z,ℓ)-Strict Temporal Separator.

```
1 V' ← ∅;
2 while there exists a strict ℓ-temporal path P, between s and z do
3 |   V' ← V' ∪ (V(P) \ {s, z});
4 |   Remove V(P) \ {s, z} from G;
5 end
6 return (V')
```

Algorithm 2 has approximation factor $\ell-1$ since the ℓ-temporal paths considered in each iteration are vertex-disjoint, thus any solution of (s,z,ℓ)-Strict Temporal Separator contains at least one vertex for each of these temporal path.

▶ **Theorem 11.** *Algorithm 2 is an* $(\ell-1)$*-approximation algorithm for (s,z,ℓ)-Strict Temporal Separator.*

Next, we prove that, using an approximation preserving reduction from Vertex Cover k-Uniform Hypergraph similar to the one presented in [8] from Set Cover, improving this approximation factor is a challenging problem. In particular, Vertex Cover k-Uniform Hypergraph is not approximable within factor $k-\varepsilon$, for any positive constant ε, assuming the Unique Game Conjecture [13].

▶ **Corollary 12.** *(s,z,ℓ)-Strict Temporal Separator is hard to approximate within factor* $(\ell-1)-\varepsilon$*, for any positive constant* ε*, assuming the Unique Game Conjecture.*

5 Tractability and approximation of (s,z,ℓ)-Temporal Cut

In this section we consider the (s,z,ℓ)-Temporal Cut problem. First, we prove the APX-hardness of the problem, then we present a $2\log_2(2\ell)$-approximation algorithm.

5.1 Hardness of (s,z,ℓ)-Temporal Cut

We show the APX-hardness of (s,z,ℓ)-Temporal Cut, by presenting an approximation preserving reduction from the Multiway Cut problem, which is APX-hard [5]. In this latter problem, the input is an undirected graph G with $k \geq 3$ distinguished vertices $v_1, \ldots, v_k$

called *terminals*. The goal is to remove the minimum number of edges from G to cut all paths between terminals, that is, compute $F \subseteq E(G)$ of minimum size such that in $G - F$ there is no path between v_i and v_j for every distinct $i, j \in [k]$.

Consider an instance $(G, v_1, \ldots, v_k)$ of Multiway Cut and construct a temporal graph H as follows. First define $\ell = k - 1$. We assume that some edges of H are *undeletable*. This can easily be achieved by replacing an undeletable edge (u, v, t) with a large number of parallel paths between u and v at time t. We construct H as follows. First, start with a copy of G in which every temporal edge is defined at time ℓ, and every such temporal edge is *deletable*: for each $uv \in E(G)$ we add (u, v, ℓ) to H. Then add vertices s and z, and add the following *undeletable* edges:

$$(s, v_1, 1), \quad (v_1, z, \ell + 1); \quad (s, v_2, 2), \quad (v_2, z, \ell + 2); \quad \ldots$$
$$(s, v_{k-1}, k - 1), \quad (v_{k-1}, z, \ell + k - 1); \quad (v_k, z, \ell).$$

In other words we add (s, v_i, i) and $(v_i, z, \ell + i)$ for $i \in [k - 1]$, and add (v_k, z, ℓ). Note that s is a neighbor of every v_i except v_k. Also observe that the edge $(v_{k-1}, z, \ell + k - 1)$ is useless, but we include it to preserve the pattern.

▶ **Theorem 13.** *The (s,z,ℓ)-Temporal Cut problem is APX-hard, even for $\ell = 2$.*

Proof sketch. Let $F \subseteq E(G)$ be a multiway cut of G, so that all $v_i - v_j$ paths are removed in $G - F$. In H, remove the set $F' = \{(u, v, \ell) : uv \in F\}$. Note that s needs to use two distinct v_i and v_j terminals to reach z, because it first needs to use some (s, v_i, i) temporal edges, but cannot use the temporal edge $(v_i, z, i + \ell)$. But F' cuts any path between two terminals, so F' is a temporal cut of H.

Conversely, any temporal cut $F' \subseteq E(H)$ contains only deletable edges which are copied from G. Consider $F = \{uv \in E(G) : (u, v, \ell) \in F'\}$. A case analysis shows that if $G - F$ has a path between v_i and v_j, then $H - F'$ has an ℓ-temporal path from s to z, going through v_i then v_j or vice-versa using that path. Thus F must be a multiway cut for G. ◀

Note for $\ell = 1$, (s,z,ℓ)-Temporal Cut is in P: the subgraphs that occur at different times can be treated independently, and thus $\ell = 1$ can be solved by computing a minimum edge cut for each possible time.

Also note that unlike most of our previous results, there is no obvious extension of the above reduction to the the strict variant, where one must delete a minimum number of edges to remove all strict temporal paths of travel time at most ℓ. We leave the complexity of latter as an open problem.

5.2 A $2\log_2(2\ell)$-Approximation for (s,z,ℓ)-Temporal Cut

We present a $2\log_2(2\ell)$-approximation algorithm for (s,z,ℓ)-Temporal Cut. We start by recalling a generalization of Menger's Theorem on temporal graphs [2].

▶ **Theorem 14.** *The maximum number of edge-disjoint (s, z)-temporal paths in a temporal graph G equals the size of a minimum temporal (s, z)-cut of G.*

Also given a temporal graph G, it is possible to compute in polynomial time a minimum temporal (s, z)-cut of G [2] and we denote such algorithm by `TempCut`(G). We present now a result that will be useful to prove the approximation factor.

▶ **Lemma 15.** *Given $t \in [2, \tau - 1]$, consider temporal graph $G([t, t + \ell - 1])$ and let E'_t be an (s, z)-cut of $G([t, t + \ell - 1])$. Consider a temporal path p of $G([t + i, t + \ell + i - 1]) - E'_t$, for some $i > 0$, and a temporal path p' of $G([t - j, t - j + \ell - 1]) - E'_t$, for some $j > 0$. Then p and p' are temporal edge disjoint.*

We describe now an algorithm that has approximation factor $\log_2 \tau$, then we show how to use it to achieve approximation factor $2\log_2(2\ell)$. We assume for simplicity that ℓ is even. The algorithm is recursive: given an interval $[a, b]$, at each step if the difference $b - a$ is at

Algorithm 3 $\log_2 \tau$-approximation algorithm for (s,z,ℓ)-Temporal Cut.

```
1 G' ← G,    E' ← ∅,    a ← 1, b ← τ;
2 Function ℓ − TempEdgeCut(G, r, l)
3 if b − a ≥ ℓ − 1 then
4 |   t ← ⌊(a+b)/2⌋;
5 |   E' ← E' ∪ TempCut(G([t − ℓ/2, t + ℓ/2 − 1)]);
6 |   ℓ − TempEdgeCut(G, a, t + ℓ/2 − 2);
7 |   ℓ − TempEdgeCut(G, t − ℓ/2 + 1, b);
8 end
```

least $\ell - 1$, thus $[a, b]$ contains at least ℓ timestamps, it defines a timestamp t that partitions in two (almost) equal parts the intervals of length ℓ contained in $[a, b]$. Then it computes a minimum temporal cut of $G[t - \ell/2, t + \ell/2 - 1]$ and recursively computes a solution on the first part of intervals and a solution on the second part of intervals (independently). Since the number of intervals of length ℓ is bounded by $\tau - \ell < \tau$, and each recursive call partitions in two equal parts this set of intervals, it follows that after $\log_2 \tau$ levels of recursion, the size of an interval is at most ℓ, thus there can be at most $\log_2 \tau$ levels of recursion.

Denote by h the number of recursion levels and by w_i the number of timestamps defined (as t in the pseudocode) at level i, $i \in [h]$. Denote by $t_{i,j}$ the j-th timestamp chosen by the algorithm at the i-th level of the recursion, $i \in [h]$ and $j \in [w_i]$. Now, consider an approximate solution E' and optimal solution Opt of (s,z,ℓ)-Temporal Cut. We partition E' and Opt as follows. Given a timestamp $t_{i,j}$, $i \in [h]$ and $j \in [w_i]$, we denote by $E'(t_{i,j})$ the set of temporal edges added to E' by the approximation algorithm when it defines $t = t_{i,j}$. By construction and by Lemma 15, the sets $E'(t_{i,j})$, $i \in [h]$, $j \in [w_i]$, define a partition of E'. Similarly, we define $Opt(t_{i,j})$, $i \in [h]$, $j \in [w_i]$, as the set of temporal edges $e \in Opt$ such that (1) there is an ℓ-temporal path in $G([t_{i,j} - \ell/2, t_{i,j} + \ell/2 - 1])$ from s to z that contains e and (2) e does not belong to any $Opt(t_{r,q})$, with $r < i$ for some $q \in [w_r]$. Note that by Lemma 15 this is indeed a partition, and in particular for each $i \in [h]$ it holds that $Opt(t_{i,j}) \cap Opt(t_{i,q}) = \emptyset$ for each $j, q \in [w_i]$ and $j \neq q$.

Since each temporal edge $e \in Opt$ belongs to exactly one set $Opt(t_{i,j})$ and since $Opt(t_{i,j}) \cap Opt(t_{i,q}) = \emptyset$, for each $j, q \in [w_i]$, we have that

$$Opt = \bigcup_{i \in [h], j \in [w_i]} Opt(t_{i,j})$$

and

$$|Opt| = \sum_{i \in [h], j \in [w_i]} |Opt(t_{i,j})|.$$

Now, we prove the following result.

▶ **Lemma 16.** *Consider $E'(t_{i,j})$, $i \in [h]$, $j \in [w_i]$, the set of temporal edges cut by Algorithm 3 when it defines $t = t_{i,j}$. For each level $i \in [h]$ of the recursion, it holds that*

$$\sum_{j \in [w_i]} |E'(t_{i,j})| \leq \sum_{j \in [w_i]} |Opt(t_{i,j})| + \sum_{r \in [h] : r < i, q \in [w_r]} |Opt(t_{r,q})|.$$

Based on the previous lemma, since the number of recursion levels is bounded by $\log_2 \tau$, we can prove the following.

▶ **Theorem 17.** *Algorithm 3 returns an (s, z, ℓ)-temporal cut of size at most $\log_2 \tau \cdot Opt$.*

Next we prove that the previous approximation algorithm can be used to obtain an approximation algorithm of factor $2\log_2(2\ell)$ for (s,z,ℓ)-Temporal Cut. We assume that τ is a multiple of 2ℓ (if not, increase τ to the next multiple of 2ℓ, with no temporal edges existing in the extra times appended). Consider the time domain $[1, \tau]$ and computes the following sets of disjoint intervals of $[1, \tau]$:

- $P_1 = \{[1, 2\ell], [2\ell + 1, 4\ell], ..., [\tau - 2\ell + 1, \tau]\}$;
- $P_2 = \{[\ell, 3\ell], [3\ell + 1, 5\ell], ..., [\tau - 3\ell, \tau - \ell]\}$.

Each interval of length ℓ is contained in at least one element of P_1 or P_2. and, moreover, two graphs defined on distinct intervals of a set P_i are temporal edge disjoint. Recall that $G(I)$ is the temporal graph defined on interval I.

▶ **Lemma 18.** *Consider the sets P_1 and P_2, then (1) for each $I = [t, t + \ell - 1]$ of length ℓ, there is an interval of P_1 or P_2 that contains it; (2) Given I and I' be two distinct interval of P_i, then a temporal path of $G(I)$ and a temporal path of $G(I')$ are temporal edge disjoint.*

Now, we run Algorithm 3 on each $G(I)$, with $I \in P_i$ and we compute a feasible solution since by Lemma 18 each interval I of length ℓ is contained in an interval of P_1 or P_2. The $2\log_2(2\ell)$-approximation factor is due to the fact that each interval I has length 2ℓ and we approximate each with a factor of $\log_2(2\ell)$. Then by Lemma 18 two intervals I and I' of P_i, are temporal edge disjoint, thus each temporal edge removed by an optimal solution belongs to at most one interval of P_1 and at most one interval of P_2. Because we remove edges from both the intervals in P_1 and P_2, our approximation factor is multiplied by two.

▶ **Corollary 19.** *(s,z,ℓ)-Temporal Cut admits a $2\log_2(2\ell)$-approximation algorithm.*

6 Conclusion

We have studied (s,z,ℓ)-Temporal Separator and (s,z,ℓ)-Temporal Cut, two variants of the vertex separator and the edge cut problem in temporal graphs. We have shown that (s,z,ℓ)-Temporal Separator is NP-hard even when the underlying graph has pathwidth bounded by four and it can be solved in polynomial time for graphs of pathwidth bounded by three. We have shown that (s,z,ℓ)-Temporal Separator cannot be approximated within factor $2^{\Omega(\log^{1-\varepsilon}|V|)}$ for any constant $\varepsilon > 0$, unless $NP \subseteq ZPP$, and that the strict version is approximable within factor $\ell - 1$; moreover, we have shown that it is unliklely that this factor can be improved. Finally, We have shown that (s,z,ℓ)-Temporal Cut is APX-hard and that it is approximable within factor $2\log_2(2\ell)$.

We conclude the paper with a few open problems:

- Is the (s,z,ℓ)-Temporal Separator problem NP-hard on graphs of treewidth at most three?
- Is there a $O(\ell)$-approximation for (s,z,ℓ)-Temporal Separator (non strict variant)?
- Is there a $O(1)$-approximation for the (s,z,ℓ)-Temporal Cut problem?
- Is the *strict* variant of the (s,z,ℓ)-Temporal Cut NP-hard?

References

1 Dan Bergren, Eduard Eiben, Robert Ganian, and Iyad Kanj. On covering segments with unit intervals. *SIAM J. Discret. Math.*, 36(2):1200–1230, 2022. doi:10.1137/20M1336412.

2 Kenneth A. Berman. Vulnerability of scheduled networks and a generalization of menger's theorem. *Networks*, 28(3):125–134, 1996. doi:10.1002/(SICI)1097-0037(199610)28:3\%3C125::AID-NET1\%3E3.0.CO;2-P.

3 Hans L Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996. doi:10.1006/JAGM.1996.0049.

4 Julia Chuzhoy and Sanjeev Khanna. Polynomial flow-cut gaps and hardness of directed cut problems. *J. ACM*, 56(2):6:1–6:28, 2009. doi:10.1145/1502793.1502795.

5 Elias Dahlhaus, David S Johnson, Christos H Papadimitriou, Paul D Seymour, and Mihalis Yannakakis. The complexity of multiway cuts. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 241–251, 1992.

6 Michael R Fellows, Danny Hermelin, Frances Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical computer science*, 410(1):53–61, 2009. doi:10.1016/J.TCS.2008.09.065.

7 Till Fluschnik, Hendrik Molter, Rolf Niedermeier, Malte Renken, and Philipp Zschoche. Temporal graph classes: A view through temporal separators. *Theor. Comput. Sci.*, 806:197–218, 2020. doi:10.1016/J.TCS.2019.03.031.

8 Hovhannes A. Harutyunyan, Kamran Koupayi, and Denis Pankratov. Temporal separators with deadlines. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation, ISAAC 2023, December 3-6, 2023, Kyoto, Japan*, volume 283 of *LIPIcs*, pages 38:1–38:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ISAAC.2023.38.

9 Petter Holme. Modern temporal network theory: a colloquium. *The European Physical Journal B*, 88(9):234, 2015.

10 Mohammad Mehdi Hosseinzadeh, Mario Cannataro, Pietro Hiram Guzzi, and Riccardo Dondi. Temporal networks in biology and medicine: a survey on models, algorithms, and tools. *Netw. Model. Anal. Health Informatics Bioinform.*, 12(1):10, 2023. doi:10.1007/S13721-022-00406-X.

11 Allen Ibiapina and Ana Silva. Mengerian graphs: Characterization and recognition. *J. Comput. Syst. Sci.*, 139:103467, 2024. doi:10.1016/J.JCSS.2023.103467.

12 David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *J. Comput. Syst. Sci.*, 64(4):820–842, 2002. doi:10.1006/jcss.2002.1829.

13 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2- ε. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

14 Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Interference-free walks in time: temporally disjoint paths. *Auton. Agents Multi Agent Syst.*, 37(1):1, 2023. doi:10.1007/S10458-022-09583-5.

15 Nicolas Maack, Hendrik Molter, Rolf Niedermeier, and Malte Renken. On finding separators in temporal split and permutation graphs. *J. Comput. Syst. Sci.*, 135:1–14, 2023. doi:10.1016/J.JCSS.2023.01.004.

16 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Math.*, 12(4):239–280, 2016. doi:10.1080/15427951.2016.1177801.

17 Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The complexity of finding small separators in temporal graphs. *J. Comput. Syst. Sci.*, 107:72–92, 2020. doi:10.1016/J.JCSS.2019.07.006.

Computational Geometry with Probabilistically Noisy Primitive Operations

David Eppstein
University of California, Irvine, CA, USA

Michael T. Goodrich
University of California, Irvine, CA, USA

Vinesh Sridhar
University of California, Irvine, CA, USA

Abstract

Much prior work has been done on designing computational geometry algorithms that handle input degeneracies, data imprecision, and arithmetic round-off errors. We take a new approach, inspired by the *noisy sorting* literature, and study computational geometry algorithms subject to noisy Boolean primitive operations in which, e.g., the comparison "is point q above line ℓ?" returns the wrong answer with some fixed probability. We propose a novel technique called *path-guided pushdown random walks* that generalizes the results of noisy sorting. We apply this technique to solve point-location, plane-sweep, convex hulls in 2D and 3D, and Delaunay triangulations for noisy primitives in optimal time with high probability.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Random walks and Markov chains

Keywords and phrases Computational geometry, noisy comparisons, random walks

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.24

Related Version *Full Version*: https://arxiv.org/abs/2501.07707

Funding *David Eppstein*: Research supported in part by NSF grant CCF-2212129.
Michael T. Goodrich: Research supported in part by NSF grant CCF-2212129.

1 Introduction

In 1961, Rényi [37] introduced a binary search problem where comparisons between two values return the wrong answer independently with probability $p < 1/2$; see also, e.g., Pelc [33, 34]. Subsequently, in 1994, Feige, Raghavan, Peleg, and Upfal [15] showed how to sort n items in $O(n \log n)$ time, with high probability,[1] in the same noisy comparison model. Building on this classic work, in this paper we study the design of efficient computational geometry algorithms using Boolean geometric primitives, such as orientation queries or sidedness tests, that randomly return the wrong answer (independently of previous queries and answers) with probability at most $p < 1/2$ (where p is known) and otherwise return the correct answer.

The only prior work we are aware of on computational geometry algorithms tolerant to such non-persistent errors is work by Groz, Mallmann-Trenn, Mathieu, and Verdugo [19] on computing a d-dimensional skyline of size k and probability $1 - \delta$ in $O(nd \log(dk/\delta))$ time. We stress that, as in our work, this noise model is different than the considerable prior work on geometric algorithms that are tolerant to uncertainty, imprecision, or degeneracy in their inputs, some of which we review below. Indeed, the motivation for our study does

[1] In this paper, we take "with high probability" (w.h.p.) to mean that the failure probability is at most $1/n^c$, for some constant $c \geq 1$.

© David Eppstein, Michael T. Goodrich, and Vinesh Sridhar;
licensed under Creative Commons License CC-BY 4.0

not come from issues that arise, for example, from round-off errors, geometric measurement errors, or degenerate geometric configurations. Instead, our motivation comes from potential applications involving quantum computing, where a quantum computer is used to answer primitive queries, which return an incorrect answer with a fixed probability at most $p < 1/2$; see, e.g., [2, 23, 24]. We see our contribution as a complement to prior work that applies quantum algorithms in a black-box fashion to quickly solve geometric problems, e.g., see [3]. Our technique gives greater flexibility to algorithms that rely on noisy primitives or subprocessors and may be useful in further developing this field. A second motivation for this noise model comes from communication complexity, in which hash functions can be used to reduce communication costs in distributed algorithms at the expense of introducing error. Viola models this behavior with noisy primitives to efficiently construct higher-level, fault-tolerant algorithms [40].

A simple observation, made for sorting by Feige, Raghavan, Peleg, and Upfal [15], is that we can use any polynomial-time algorithm based on correct primitives by repeating each primitive operation $O(\log n)$ times and taking the majority answer as the result. This guarantees correctness w.h.p., but increases the running time by a $\log n$ factor. In this paper, we design computational geometry algorithms with noisy primitive operations that are correct w.h.p. without incurring this overhead. In the full version, we show that the logarithmic overhead is unavoidable for certain problems, including closest pairs and detecting collinearities.

1.1 Related Work

There is considerable prior work on sorting and searching with noisy comparison errors. For example, Feige, Raghavan, Peleg, and Upfal [15] show that one can sort in $O(n \log n)$ time w.h.p. with probabilistically noisy comparisons. Dereniowski, Lukasiewicz, and Uznanski [9] study noisy binary searching, deriving time bounds for constant factors involved. A similar study has also been done by Wang, Ghaddar, Zhu, and Wang [41]. Klein, Penninger, Sohler, and Woodruff solve linear programming in 2D under a different model of primitive errors [25], in which errors persist regardless of whether primitives are recomputed.

Other than the work by Groz *et al.* [19] mentioned above, we are not aware of prior work on computational geometry algorithms with random, independent noisy primitives. Nevertheless, considerable prior work has designed algorithms that can deal with input degeneracies, data imprecision, and arithmetic round-off errors. For example, several researchers have studied general methods for dealing with degeneracies in inputs to geometric algorithms, e.g., see [13, 42]. Researchers have designed algorithms for geometric objects with imprecise positions, e.g., see [27, 28]. In addition, significant prior work has dealt with arithmetic round-off errors and/or performing geometric primitive operations using exact arithmetic, e.g., see [16, 30]. While these prior works have made important contributions to algorithm implementation in computational geometry, they are orthogonal to the probabilistic noise model we consider in this paper.

Emamjomeh-Zadeh, Kempe, and Singhal [14] and Dereniowski, Tiegel, Uznański, and Wolleb-Graf [10] explore a generalization of noisy binary search to graphs, where one vertex in an undirected, positively weighted graph is a target. Their algorithm identifies the target by adaptively querying vertices. A query to a node v either determines that v is the target or produces an edge out of v that lies on a shortest path from v to the target. As in our model, the response to each query is wrong independently with probability $p < 1/2$. This problem is different than the graph search we study in this paper, however, which is better suited to applications in computational geometry. For example, in computational geometry applications, there is typically a search path, P, that needs to be traversed to a target vertex,

but the search path P need not be a shortest path. Furthermore, in such applications, if one queries using a node, v, that is not on P, it may not even be possible to identify a node adjacent to v that is closer to the target vertex.

Viola [40] uses a technique similar to ours to handle errors in a communication protocol. In one problem studied by Viola, two participants with n-bit values seek to determine which of their two values is largest. This can be done by a noisy binary search for the highest-order differing bit position. Each search step performs a noisy equality test on two prefixes of the inputs, by exchanging single-bit hash values. The result is an $O(\log n)$ bound on the randomized communication complexity of the problem. Viola uses similar protocols for other problems including testing whether the sum of participant values is above some threshold. The noisy binary search protocol used by Viola directs the participants down a decision tree, with an efficient method to test whether the protocol has navigated down the wrong path in order to backtrack. One can think of our main technical lemma as a generalization of this work to apply to any DAG.

1.2 Our Results

This work centers around our novel technique, *path-guided pushdown random walks*, described in Section 3. It extends noisy binary search in two ways: it can handle searches where the decision structure of comparisons is in general a DAG, not a binary tree, and it also correctly returns a non-answer in the case that the query value is not found. These two traits allow us to implement various geometric algorithms in the noisy comparison setting.

However, to apply path-guided pushdown random walks, one must design an oracle that, given a sequence of comparisons, can determine if one of them is incorrect in constant time. Because different geometric algorithms use different data structures and have different underlying geometry, we must develop a unique oracle for each one. The remainder of the paper describes noise-tolerant implementations with optimal running times for plane-sweep algorithms, point location, convex hulls in 2D and 3D, and Delaunay triangulations. We also present a dynamic 2D hull construction that runs in $O(\log^2 n)$ time w.h.p. per operation. See Table 1 for a summary of our results.

Our algorithms to construct the trapezoidal decomposition, Delaunay triangulation, and 3D convex hull are adaptions of classic randomized incremental constructions. A recent paper by Gudmundsson and Seybold [20] published at SODA 2022 claimed that such constructions produce search structures of depth $O(\log n)$ and size $O(n)$ w.h.p. in n, implying that the algorithms run in $O(n \log n)$ time w.h.p. Unfortunately, the authors have contacted us privately to say that their analysis of the size of such structures contains a bug. In this paper, we continue to use the standard expected time analysis.

2 Preliminaries

Noisy Boolean Geometric Primitive Operations

Geometric algorithms typically rely on one or more Boolean geometric primitive operations that are assumed to be computable in $O(1)$ time. For example, in a Delaunay triangulation algorithm, this may be determining if some point p is located in some triangle Δ; in a 2D convex hull algorithm, this may be an orientation test; etc. Here we assume that a geometric algorithm relies on primitive operations that each outputs a Boolean value and has a fixed probability $p < 1/2$ of outputting the wrong answer. As in earlier work for the sorting problem [15], we assume non-persistent errors, in which each primitive test can be viewed as an independent weighted coin flip.

Table 1 Our main results in the noisy setting. All algorithms succeed w.h.p. in n. Runtimes marked with a * are in expectation. Runtimes marked with a † are optimal despite having faster algorithms in the non-noisy setting.

Algorithm	Runtime	Section
Trapezoidal Map	$\Theta(n \log n)^\star$	Section 4
Trapezoidal Map with k Crossings	$O((n+k)\log n)$	Section 5.2
2D Closest Points	$\Theta(n \log n)^\dagger$	Section 5.3
2D Convex Hull	$\Theta(n \log n)^\dagger$	Section 6.1
Dynamic 2D Convex Hull	$O(\log^2 n)$ per update	Section 6.2
3D Convex Hull	$\Theta(n \log n)^{\star\dagger}$	Section 6.3
Delaunay Triangulation	$\Theta(n \log n)^\star$	Section 7

In each section below, we specify the Boolean geometric primitive operation(s) relevant to the algorithm in consideration. We note here that, while determining whether two objects a and b have equal value may be a noisy operation, determining whether two pointers both point to the same object a is not a noisy operation. We also note that manipulating and comparing non-geometric data, such as pointers or metadata of nodes in a tree (e.g., for rotations), are not noisy operations. This is true even if the tree was constructed using noisy comparisons.

The Trivial Repetition Strategy

As mentioned above, Feige, Raghavan, Peleg, and Upfal [15] observed in the context of the noisy sorting problem that by simply repeating a primitive operation $O(\log n)$ times and choosing the decision returned a majority of the time, one can select the correct answer w.h.p. The constant in this logarithmic bound can be adjusted as necessary to make a polynomial number of correct decisions, w.h.p., as part of any larger algorithm. Indeed, this naive method immediately implies $O(n \log^2 n)$ algorithms for a majority of the geometric constructions we discuss below. The goal of our paper is to improve this to an optimal running time using the novel technique described in Section 3.

General Position Assumptions

For the sake of simplicity of expression, we make standard general position assumptions throughout this paper: no two segment endpoints in a trapezoidal decomposition and no two events in a plane sweep algorithm have the same x-coordinate, no three points lie in a line and no four points lie on a plane for 2D and 3D convex hulls respectively, and no four points lie in a circle for Delaunay triangulations. Applying perturbation methods [29] or implementing special cases in each algorithm would allow the relaxation of these assumptions.

3 Path-Guided Pushdown Random Walks

In this section, we provide an analysis tool that we use repeatedly in this paper and which may be of independent interest (e.g., to analyze randomized routing protocols). Specifically, we introduce *path-guided pushdown random walks*, which are related to biased random walks on a graph (e.g., see [5, 17]) and generalize the noisy binary search problem [15, 18]. See Figure 1 for a depiction of path-guided pushdown random walks.

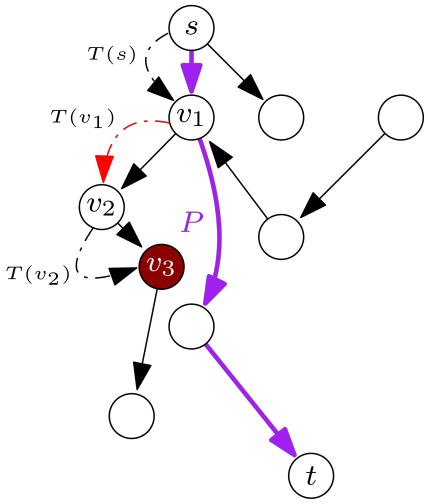


Figure 1 Here we show a sample execution of path-guided pushdown random walks in some DAG G. The transition oracle acts *arbitrarily* when it lies, so we may end up with the sequence shown here, a correct move prior to our current node but an incorrect move to v_2. To apply path-guided pushdown random walks, we must be able to determine whether we are on P in $O(1)$ comparisons, no matter where we are in G.

A path-guided pushdown random walk is defined in terms of a directed acyclic graph (DAG), G, that has a starting vertex, s, a target vertex, t, and a path, $P = (s, v_1, v_2, \ldots, t)$, from s to t, in G (we assume $s \neq t$). We start our walk from the start vertex, $v \leftarrow s$, and we use a stack, S, which initially contains only s, and a transition oracle, $T(v)$, to determine our walk. For each vertex, v, during our walk, we consult the transition oracle, $T(v)$, which first tells us whether $v \in P$ and if so, then $T(v)$ tells us the next vertex in P to visit to make progress towards t. $T(v)$ can return v, which means we should stay at v, e.g., if $v = t$.

Our model allows T to "lie." We assume a fixed error probability,[2] $p_e < 1/15$, such that T gives the correct answer with probability $1 - p_e$, independently each time we query T. With probability p_e, $T(v)$ can lie, i.e., $T(v)$ can indicate "$v \in P$" when $v \notin P$, $T(v)$ can indicate "$v \notin P$" when $v \in P$, or $T(v)$ can return a "next" vertex that is not an actual next vertex in P (including returning v itself even though $v \neq t$). Importantly, this next vertex must be an outgoing neighbor of v. This allows us to maintain the invariant that S holds an actual path in G (with repeated vertices). Our traversal step, for current vertex v, is as follows:

- If $T(v)$ indicates that $v \notin P$ (and $v \neq s$), then we set $v \leftarrow S.\mathsf{pop}()$, which may be v again, for the next iteration. This is a backtracking step.
- If $T(v)$ indicates that $v \in P$, then let w be the vertex indicated by $T(v)$ as next in P, such that $v = w$ or (v, w) is an edge in G.[3]
  - If $v = w$, then we call $S.\mathsf{push}(v)$ and repeat the iteration with this same v, since this is evidence we have reached the target, t.
  - Else ($v \neq w$) if $v = S.\mathsf{top}()$, then we set $v \leftarrow S.\mathsf{pop}()$. That is, in this case, we don't immediately transition to w, but we take this as evidence that we should not stay at v, as we did in the previous iteration. This is another type of backtracking step.
  - Otherwise, we call $S.\mathsf{push}(v)$ and set $v \leftarrow w$ for the next iteration.

[2] The threshold of 1/15 simplifies our proof. We can tolerate any higher error probability bounded below 1/2, by repeating any query a constant number of times and taking the majority answer.

[3] If $w \neq v$ and (v, w) is not an edge in G, we immediately reject this call to T and repeat the call to T.

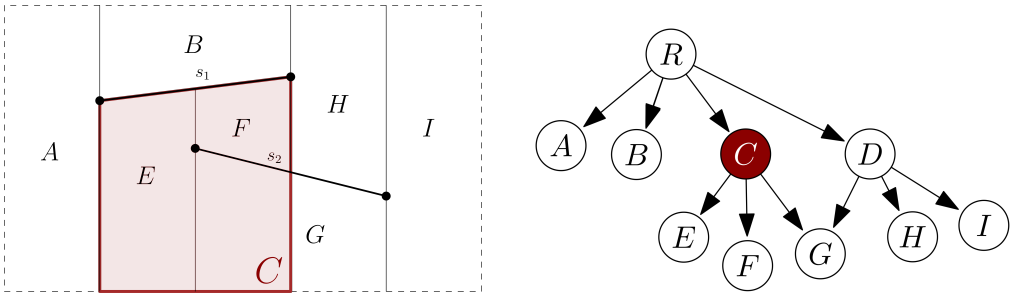


Figure 2 An instance of a path-guided pushdown random walk in the trapezoidal map history DAG. R represents the initial bounding box, the leaf nodes represent currently visible trapezoids. The remaining nodes represent destroyed trapezoids. We are at node C in a path-guided pushdown random walk, so we test if our query point lies in trapezoid C, the shaded region in the decomposition.

We repeat this step until we are confident we can stop, which occurs when enough copies of the same vertex occur at the top of the stack.

▶ **Theorem 1.** *Given an error tolerance, $\varepsilon = n^{-c}$ for $c > 0$, and a DAG, G, with a path, P, from a vertex, s, to a distinct vertex, t, the path-guided pushdown random walk in G starting from s will terminate at t with probability at least $(1 - \varepsilon)$ after $N = \Theta(|P| + \log(1/\varepsilon))$ steps, for a transition oracle, T, with error probability $p_e < 1/15$.*

The proof of Theorem 1 can be found in Section A. In short, we show that each "good" action by T can undo any "bad" action by T. By applying a Chernoff bound, we then show that, w.h.p., we reach t after N steps and terminate after another $O(\log(1/\varepsilon))$ steps. The requirement that ε be polynomially small is used to ensure that premature termination is unlikely. For the remainder of the paper, we assume that $\varepsilon = 1/n^c$ for some constant $c > 0$. A majority of the algorithms below invoke path-guided pushdown random walks at most $O(n)$ times. Taking the union bound over all $O(n)$ invocations still shows that path-guided pushdown random walks fails with probability $O(1/n^{c-1})$. Thus, for $c \geq 2$, all invocations succeed w.h.p. This choice of ε adds $O(\log n^c) = O(\log n)$ to the time for each walk. However, in our applications it can be shown that $|P| = O(\log n)$, so this does not change the asymptotic complexity of the operation.

4 Noisy Randomized Incremental Construction for Trapezoidal Maps

In this section, we show how a history DAG in a randomized incremental construction (RIC) algorithm can be used as the DAG of Theorem 1. Suppose we are given a set of n non-crossing line segments in the plane and wish to construct their trapezoidal decomposition, $\mathcal{D}$. We outline below how the history DAG for a (non-noisy) RIC algorithm for constructing $\mathcal{D}$ can be used as the DAG of Theorem 1. We show that, even in the noisy setting, such a DAG can perform point location in the history of trapezoids created and/or destroyed within the algorithm to locate the endpoints of successively inserted line segments. In particular, we consider a history DAG where any trapezoid destroyed in a given iteration points to the new trapezoids that replace it. This is in contrast to another variant of a history DAG that represents the segments themselves as nodes in the DAG [8], which seems less usable when primitives are noisy. To make our RIC algorithm noise-tolerant, we must solve two issues. The first is to navigate the history DAG in $O(\log n)$ time w.h.p.; the second is to walk along each segment to merge and destroy trapezoids when it is added.

To navigate down the history DAG, we apply a path-guided pushdown random walk. To do so we must test, with a constant number of operations, whether we are on the path that a non-noisy algorithm would search to find the query endpoint q of a new segment, s_i. See Figure 2 for an example of this process. Each node of our history DAG represents a trapezoid (either destroyed if an internal node or current if a leaf node). Importantly, each of the at most four children of a node cover the parent's trapezoid, but do not overlap. Therefore, we can define a unique path, P w.r.t query q, to be a sequence of trapezoids beginning at the root of the history DAG and ending at the leaf-node trapezoid containing q. Each node on the path is the unique child of the previous node that contains q. Thus, all our transition oracle must do to determine if we are on the correct path is to check whether q is contained by the trapezoid mapped to our current node v. If this test succeeds, the oracle determines (rightly or wrongly) that v is on a valid path, and it proceeds to compare q against the segment whose addition split trapezoid v in order to return the next node of the walk. If one or more of these tests fails, the oracle says that v is not on a valid path. Let ℓ_i be the sum of the lengths of the two unique paths corresponding to the endpoints of s_i. If we set our error probability for path-guided pushdown random walks to be w.h.p. in n, the point-location cost to insert the ith segment is $O(\ell_i + \log n)$.

For the second issue, suppose there are d_i trapezoids between the left and right endpoint of the ith segment to be inserted, and that we need to walk left-to-right in the current subdivision to find them. To find the next trapezoid in this walk, we simply test if the segment endpoint that defines the right boundary of the current trapezoid lies above or below segment s_i, e.g., determining whether to choose its upper-right or lower-right neighbor. Combining this above-below test with the trivial repetition strategy from Section 2, we can compute the correct sequence of trapezoids in $O(d_i \log n)$ time w.h.p.

Via the standard backwards analysis, $\sum_{i=1}^{n} \ell_i = O(n \log n)$ and $\sum_{i=1}^{n} d_i = O(n)$ in expectation [8]. It has been shown [8, 38] that the search depth of the final history DAG is $O(\log n)$ w.h.p. Therefore, after constructing the decomposition, we can use path-guided pushdown random walks to answer planar point-location queries in $O(\log n)$ time w.h.p. We conclude the following.

▶ **Theorem 2.** *We can successfully compute a trapezoidal decomposition map of n non-crossing line segments w.h.p. in expected $O(n \log n)$ time, even with noisy primitives, and thereby construct a data structure (the history DAG) that answers point location queries in $O(\log n)$ time w.h.p.*

It is natural to hope that this can be extended to line segment arrangements with crossings, for which the best non-noisy time bounds are $O(n \log n + k)$, achieved with a similar randomized incremental approach. We show in the full version that an extra logarithmic factor may be necessary, however.

5 Plane-Sweep Algorithms

In this section, we show that many plane-sweep algorithms can be adapted to our noisy-primitive model.

5.1 Noisy Balanced Binary Search Trees

We begin by noting that we can implement binary search trees storing geometric objects to support searches and updates in $O(\log n)$ time w.h.p. by adapting the noisy balanced binary search trees recently developed for numbers in quantum applications by Khadiev,

Savelyev, Ziatdinov, and Melnikov [24] to our noisy-primitive framework for geometric objects. Plane-sweep algorithms require us to store an ordered sequence of events that are visited by the sweep-line throughout the course of the algorithm while maintaining an active set of geometric objects for the sweep line in a balanced binary tree. Some plane-sweep algorithms, such as the one of Lee and Preparata that divides a polygon into monotone pieces [26], have a static set of events that can simply be maintained as a sorted array that is constructed using an existing noisy sorting algorithm [15]. Others, however, add or change events to the event queue as the algorithm progresses. In addition, the algorithms must also maintain already-processed data in a way that allows for new data swept over to be efficiently incorporated. The two most common dynamic data structures in plane-sweep algorithms are dynamic binary search trees and priority queues. Throughout this section, noise is associated with the comparison "$a \leq b$?" where a and b are geometric objects.

Once a data structure is built, its underlying structure can be manipulated without noise. For example, we need no knowledge of the values held in a tree to recognize that it is imbalanced and to perform a rotation, allowing implementation of self-balancing binary search trees. For example, Khadiev, Savelyev, Ziatdinov, and Melnikov [24] show how to implement red-black trees [21] in the noisy comparison model for numbers. We observe here that the same method can be used for ordered geometric objects compared with noisy geometric primitives, with a binary search tree serving as the search DAG and a root-to-leaf search path as the path. See Section B for an explanation for how to instantiate path-guided pushdown random walks on a binary search tree.

5.2 Noisy Trapezoidal Decomposition with Segment Crossings

We can implement an event queue as a balanced binary search tree with a pointer to the smallest element to perform priority queue operations in the noisy setting.

▶ **Theorem 3.** *Given a set, S, of n x-monotone pseudo-segments*[4] *in the plane, we can construct a trapezoidal map of S in $O((n+k)\log n)$ time w.h.p., where k is the number of pairs of crossing segments, even with noisy primitive operations.*

Proof. Bentley and Ottmann [6] compute a trapezoidal map of line segments S via a now well-known plane-sweep algorithm, which translates directly to x-monotone pseudo-segments. Pseudo-segment endpoints and intersection points are kept in an event queue ordered by x-coordinates and line segments intersecting the sweep line are kept in a balanced binary search tree, both of which can be implemented to support searches and updates in $O(\log n)$ time w.h.p. in the noisy model, as described above. Given the $O((n+k)\log n)$-time performance of the Bentley-Ottmann algorithm, this implies that we can construct the trapezoidal map of S in $O((n+k)\log n)$ time w.h.p. ◀

We note that this running time matches the construction time in Theorem 2 for non-crossing line segments, but it does not give us a point-location data structure as in Theorem 2. Further, the upper bound of Theorem 3 is optimal with noisy primitive operations for $k = \Theta(n^2)$ via a reduction from computing line arrangements to computing trapezoidal decompositions. We show in the full version that computing an arrangement of n lines in the noisy setting takes $\Omega(n^2 \log n)$ time.

[4] A set of x-monotone psuedo-segments is a set of x-monotone curve segments that do not self-intersect and such that any two of them intersect at most once [1, 7].

5.3 Noisy Closest Pair

Here, we present another plane-sweep application in the noisy setting.

▶ **Theorem 4.** *We can find a closest pair of points, from n points in the plane, with noisy primitives, in time $O(n \log n)$ w.h.p.*

Proof. Hinrichs, Nievergelt, and Schorn show how to find a pair of closest points in the plane in $O(n \log n)$ time [22] by sorting the points by their x-coordinate and then plane-sweeping the points in that order. They maintain the minimum distance δ seen so far, and a *y-table* of *active points*. A point is active if has been processed and its x-coordinate is within δ of the current point; once this condition stops being true, it is removed from the y-table. The y-table may be implemented using a balanced binary tree. When a point is processed, the y-table is updated to remove points that have stopped being active, by checking the previously-active points sequentially according to the sorted order until finding a point that remains active. The new point is inserted into the y-table, and a bounded number of its nearby points in the table are selected. The distances between the new point and these selected points are compared to δ, and δ is updated if a smaller distance is found.

In our noisy model, sorting the points takes $O(n \log n)$ time w.h.p. [15]. Checking whether a point has stopped being active and is ready to be removed from the y-table may be done using the trivial repetition strategy of Section 2; its removal is a non-noisy operation. Inserting each point into the y-table takes $O(\log n)$ time w.h.p. Selecting a fixed number of nearby neighbors is a non-noisy operation, and comparing their distances to δ may be done using the trivial repetition strategy with a noisy primitive that compares two distances determined by two pairs of points. In this way, we perform $O(\log n)$ work for each point when it is processed, and $O(\log n)$ work again later when it is removed from the y-table. Overall, the time is $O(n \log n)$ w.h.p. ◀

We give this result mostly as a demonstration for performing plane sweep in the noisy model, since a closest pair (or all nearest neighbors) may be found by constructing the Delaunay triangulation (Section 7) and then performing min-finding operations on its edges.

An algorithm by Rabin [36] finds 2D closest points in $O(n)$ time, beating the $O(n \log n)$ plane sweep implementation that our solution is based on, in a model of computation allowing integer rounding of numerical values derived from input coordinates. However, computing the minimum of n elements takes $O(n \log n)$ time for noisy comparison trees [15], and Rabin's algorithm includes steps that find the minimum among $O(n)$ distances, so it is not faster than our algorithm in our noisy model. In the full version, we prove that if data is accessed only through noisy primitives that combine information from $O(1)$ data points (allowing integer rounding), then finding the closest pair w.h.p. requires time $\Omega(n \log n)$. Thus, the plane-sweep closest-pair algorithm is optimal in this model.

6 Convex Hulls

In this section, we describe algorithms for constructing convex hulls that can tolerate probabilistically noisy primitive operations. Here, primitive operations are orientation tests and visibility tests for 2D and 3D convex hulls respectively. Sorting is also used throughout, so comparing the x or y coordinates of two points is also a noisy primitive.

6.1 Static Convex Hulls in 2D

We begin by observing that it is easy to construct two-dimensional convex hulls in $O(n \log n)$ time w.h.p. Namely, we simply sort the points using a noise-tolerant sorting algorithm [15] and then run Graham scan [8]. The Graham scan phase of the algorithm uses $O(n)$ calls to primitives, so we can simply use the trivial repetition strategy from Section 2 to implement this phase in $O(n \log n)$ time w.h.p. We note that this is the best possible, however, since just computing the minimum or maximum of n values has an $\Omega(n \log n)$-time lower bound for a high-probability bound in the noisy primitive model [15].

6.2 Dynamic Convex Hulls in 2D

Overmars and van Leeuwen [32] show how to maintain a dynamic 2D convex hull with $O(\log^2 n)$ insertion and deletion times, $O(\log n)$ query time, and $O(n)$ space. They maintain a binary search tree, T^*, of points stored in the leaves ordered by y-coordinates and augment each internal node with a representation of half the convex hull of the points in that subtree. They define an lc-hull of P as the convex hull of $P \cup \{(\infty, 0)\}$, which is the left half of the hull of P (rc-hull is defined symmetrically). It is easy to see that performing the same query on both the lc- and rc-hull of P allows us to determine if a query point is inside, outside, or on the complete convex hull. We follow this same approach, showing how their approach can be implemented in the noisy-primitive model using path-guided pushdown random walks.

Overmars and van Leeuwen show that two lc-hulls A and C separated by a horizontal line can be merged in $O(\log n)$ time. Rather than being a single binary search, however, their method involves a joint binary search on the two trees that represent A and C with the aim of finding a tangent line that joins them. At each step, we follow one of ten cases to determine how to navigate the trees of A and/or C. Because each decision may only advance in one tree, noisy binary search is not sufficient. Fortunately, as we show, the path-guided pushdown random walk framework is powerful enough to solve this problem in the noisy model. The challenge, of course, is to define an appropriate transition oracle. Say that the transition oracle has the ability to identify which case corresponds to our current position in A and C. This decision determines which direction to navigate in A and/or C and corresponds to choosing a child to navigate down an implicit decision DAG wherein each node has ten children. It remains to show how a transition oracle acting truthfully can determine if we are on the correct path.

We have each node of A and C maintain $v.l$ and $v.r$ pointers as described in Section B. The values at $v.l$ and $v.r$ are ancestors of v whose keys are just smaller and just larger than v respectively. They determine the interval of possible values of any node in v's subtree, which corresponds to an interval of points on each hull. See Figure 3 for an example.

Say we are currently at node $a \in A$ and node $c \in C$. We can perform four case comparisons: $a.r$ with $c.r$, $a.r$ with $c.l$, $a.l$ with $c.r$, and $a.l$ with $c.l$. Each outputs a region of A and C that the two tangent points must be located in. If the intersection of all four case comparisons contains the regions bounded by $a.r$ and $a.l$ as well as $c.r$ and $c.l$, then we are on a valid path by the correctness of Overmars and van Leeuwen's case analysis [32]. Since the noise-free process takes $O(\log n)$ time, with a path-guided pushdown random walk using the transition oracle described above, this process takes $O(\log n)$ time w.h.p.

To query a hull, we can perform noisy binary search on the lc- and rc-hull structures using the transition oracle of Section B. Updating the convex hull utilizes the technical lemma discussed above along with split and join operations on binary search trees [32]. We conclude the following.

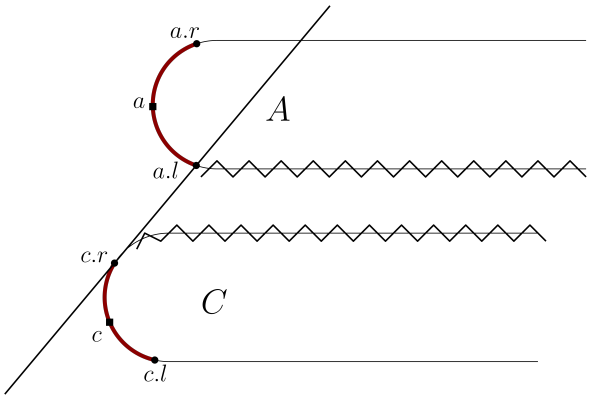


Figure 3 Here we show one of four comparisons computed by the transition oracle. The points a and c represent the node we are at in each structure. The highlighted regions represent all possible values in that node's subtree, bordered by points $a.r$ and $a.l$ (resp. $c.r$ and $c.l$). This comparison is valid as it does not eliminate the highlighted regions. If all four are valid, then we must be on the right path in our double binary search.

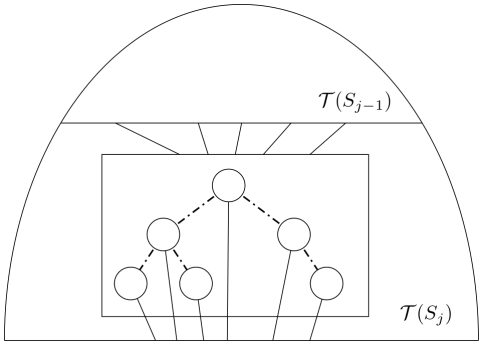


Figure 4 A depiction of how the radial-search structure is embedded into the history DAG. $\mathcal{T}(S_{j-1})$ represents the history DAG before the jth point was added to the hull. Every node that was a leaf in $\mathcal{T}(S_{j-1})$ now points to each node created in $\mathcal{T}(S_j)$. Destroyed nodes have pointers to all created nodes. The dashed lines between the nodes in $\mathcal{T}(S_j)$, represent the connections of the radial search structure, represented as a BST.

▶ **Theorem 5.** *We can insert and delete points in a planar convex hull in $O(\log^2 n)$ time per update w.h.p., even with noisy primitives.*

6.3 Convex Hulls in 3D

In this section, we show how to construct 3D convex hulls in $O(n \log n)$ time w.h.p. even with noisy primitive operations. The main challenges in this case are first to define an appropriate algorithm in the noise-free model and then define a good transition oracle for such an algorithm. For example, it does not seem possible to efficiently implement the divide-and-conquer algorithm of Preparata and Hong [35] in the noisy primitive model as each combine step performs $O(n)$ primitive operations. We begin by constructing a tetrahedron through any four points; we need the orientation of these four points, which can be found by the trivial repetition strategy in $O(\log n)$ time. The points within this tetrahedron can be discarded, again using the trivial repetition strategy in $O(\log n)$ time per point. The main challenge to designing a transition oracle for the history DAG in this method is its nested

nature. We have the nodes in the DAG themselves representing facets and pointing to the facets that replaced them when they were deleted. We also have the radial search structures, which maintain the set of new facets produced in a given iteration in sorted clockwise order around the point that generated them. See Figure 4 for a depiction of this.

Our transition oracle needs to be able to determine if we are on the correct path. Say we are given a query point, q_i, being inserted in the ith iteration of the RIC algorithm and are at node v of the history DAG. To see if we are performing the right radial search, we first check if the ray from the origin to q passes through v's parent (in the history DAG, not the radial search structure). Then we must determine if we are performing radial search correctly. To do so, we can augment the radial search tree as described in Section B and perform two more orientation tests to determine if v is on the right path in our current radial search structure. If a comparison returns false, then we use our stack to backtrack up the radial search structure or, if we are at its root, back to the parent node in the history DAG and its respective radial search tree. Thus, in just three comparisons, we can determine if we are on the right path in the history DAG. Similar to our trapezoidal decomposition analysis, we observe that the point-location cost in each iteration is $O(\ell_i + \log n)$, where ℓ_i is the length of the unique path in the history DAG corresponding to our query q. An analysis by Mulmuley [31] shows that, over all n iterations, $\sum_{i=1}^{n} \ell_i = O(n \log n)$ in expectation. As a result, total point-location cost is $O(n \log n)$ in expectation.

Once a conflicting facet is found through the random walk, we can again walk around the hull and perform the trivial repetition strategy of Section 2 to determine the set of all conflicting facets, X_i, in $O(|X_i| \log n)$ time. Each facet must be created before it is destroyed, so $\sum_{i=1}^{n} |X_i|$ is upper-bounded by the total size of the history DAG. The same analysis by Mulmuley [31] shows that the expected size of the history DAG is $O(n)$. We conclude the following.

▶ **Theorem 6.** *We can successfully compute a 3D convex hull of n points w.h.p. in expected $O(n \log n)$ time, even with noisy primitives.*

7 Delaunay Triangulations and Voronoi Diagrams

In this section, we describe an algorithm that computes the Delaunay triangulation (DT) of a set of points in the plane in the noisy-primitives model. Noise is associated with determining whether a point p_i lies within some triangle Δ and whether a point p_i lies in the circle defined by some Delaunay edge e. Here, we describe the algorithm under the Euclidean metric. In the full version, we show how to generalize our algorithm for other metrics.

Once again, we use a history DAG RIC similar to what we described in the previous section. This time, each node in the DAG represents the triangles that exist throughout the construction of the triangulation (see [31] for details). A leaf node is a triangle that exists in the current version, and an internal node is a triangle that was destroyed in a previous iteration. When we wish to insert a new point p_i, we first use the history DAG to locate where p_i is in the current DT. If p_i is within a triangle Δ, we split Δ into three triangles and add them as children of Δ. Otherwise p_i is on some edge e, and we split the two adjacent triangles Δ_1 and Δ_2 into two triangles each. Once this is done, we repeatedly flip edges that violate the empty circle property so that the triangulation becomes a DT again (see [8] Chapter 9.3). As described in [31], we store the new triangles in a radial search structure sorted CCW around the added point p_i. Using a path-guided pushdown random walk in the history DAG to find the triangle containing p_i is broadly similar to the process used to find conflicting faces in the 3D convex hull RIC. Once again, we use the ray-shooting and radial

search queries described in Section 6.3. With a similar transition oracle to 3D convex hulls, we can determine where p_i is in the current DT in $O(\ell_i + \log n)$ time, where ℓ_i is the length of p_i's unique path in the history DAG. Unlike for 3D convex hulls, there is a unique triangle or edge that is in conflict with p_i, which simplifies the process of determining a unique path through the history DAG. We observe that if p_i is not located in the triangle Δ represented by the current node in the DAG, then we are not on the right path. Once we have found the triangle containing p_i and added in new edges, we begin edge flipping. Using the trivial repetition strategy, each empty circle test costs $O(\log n)$ time. The number of empty circle tests is proportional to the number of edges flipped, which is less than the total number of triangles created throughout the algorithm. Once again, Mulmuley's [31] analysis shows that $\sum_{i=1}^{n} \ell_i = O(n \log n)$ in expectation and the number of empty circle tests needed to update the DAG after point-location is $O(n)$ in expectation. We conclude the following.

▶ **Theorem 7.** *We can successfully compute a Delaunay triangulation of n points w.h.p. in expected $O(n \log n)$ time, even with noisy primitives.*

We remark that this also leads to an algorithm for constructing a Euclidean minimum spanning tree, in which we construct the Delaunay triangulation, apply a noisy sorting algorithm to its edge lengths, and then (with no more need for noisy primitives) apply Kruskal's algorithm to the sorted edge sequence. In the following section, we generalize our Delaunay triangulation algorithm to work for a wider class of metrics, including all L_p metrics for $1 < p < \infty$.

7.1 Generalized Delaunay Triangulations

In this section, we show the following:

▶ **Theorem 8.** *Given n points in the plane, we can successfully compute a generalized Delaunay triangulation generated by homothets of any smooth shape w.h.p. in expected time $O(n \log n)$, even with noisy primitives.*

A *shape Delaunay tessellation* is a generalization of a Delaunay triangulation first defined by Drysdale [12]. Rather than a circle, we instantiate some convex compact set C in $\mathbf{R}^2$ and redefine the empty circle property on an edge $\overline{pq}$ like so: given a set of points S and some shape C, the edge $\overline{pq}$ exists in the shape Delaunay tessellation $DT_C(S)$ iff there exists some homothet of C with p and q on its boundary that contains no other points of S [4]. We will consider smooth shapes, as arbitrary non-smooth shapes may cause $DT_C(S)$ to not be a triangulation under certain point sets [4]. Note that all L_p metrics for $1 < p < \infty$ correspond to smooth shapes (rather than a unit circle, they are unit rounded squares) and furthermore, all smooth shapes produce triangulations of the convex hull of S [4, 39]. Similar to above, we use the general position assumption that no four points lie on a homothet of C.

After inserting a point, p_r, in the incremental algorithm, we draw new edges that connect p_r to the points that comprise the triangle(s) that p landed in. These new edges are Delaunay as either they are fully enclosed by the homothets of C that covered the triangle(s) p was located in or they are enclosed by the union of previously-empty shapes that covered each edge of the triangles. After this, the algorithm finds adjacent edges, tests if they obey the empty shape property, and flips them if not. In Theorem 3 of a work by Aurenhammer and Paulini [4], the authors prove that for any convex shape C, local Delaunay flips lead to $DT_C(S)$ by showing that there exists some lexicographical ordering to these triangulations similar to how the Delaunay triangulation in L_2 maximizes the minimum angle over all triangles. Thus, the flipping algorithm used after an incremental insertion will terminate at a triangulation that is $DT_C(P_r)$, where P_r is the set $\{p_1, ..., p_r\}$.

Because the flipping process works as before, the point location structure can be easily adapted to this setting. As stated above, using a smooth shape C guarantees that $DT_C(S)$ is an actual triangulation as opposed to a tree. Thus, in every iteration, there exists a triangle that the new point is enclosed by. The history DAG data structure is agnostic to the specific edge flips being made during the course of the algorithm and so works as expected. Our transition oracle also behaves the same.

In the Euclidean case, the backwards analysis of [8] uses the fact that a Delaunay triangulation of r points has $O(r)$ edges, any new triangle created in iteration r is incident to the newly inserted point p_r, and triangles are defined by at most three points. By planarity and by construction of the algorithm, all three hold true in this generalized case. Thus, we have the desired result.

8 Discussion

We have shown that a large variety of computational geometry algorithms can be implemented in optimal time or, for dynamic 2D hull, in time that matches the original work's complexity with high probability in the noisy comparison model. We believe that this is the first work that adapts the techniques of noisy sorting and searching to the classic algorithms of computational geometry, and we hope that it inspires work in other settings, such as graph algorithms.

References

1 Pankaj K Agarwal and Micha Sharir. Pseudo-line arrangements: Duality, algorithms, and applications. *SIAM Journal on Computing*, 34(3):526–552, 2005. doi:10.1137/S0097539703433900.

2 Jonathan Allcock, Jinge Bao, Aleksandrs Belovs, Troy Lee, and Miklos Santha. On the quantum time complexity of divide and conquer, 2023. doi:10.48550/arXiv.2311.16401.

3 Andris Ambainis and Nikita Larka. Quantum algorithms for computational geometry problems. *arXiv preprint arXiv:2004.08949*, 2020. arXiv:2004.08949.

4 Franz Aurenhammer and Günter Paulini. On shape Delaunay tessellations. *Inf. Process. Lett.*, 114(10):535–541, 2014. doi:10.1016/j.ipl.2014.04.007.

5 Yossi Azar, Andrei Z. Broder, Anna R. Karlin, Nathan Linial, and Steven Phillips. Biased random walks. *Combinatorica*, 16(1):1–18, 1996. doi:10.1007/BF01300124.

6 Jon L. Bentley and Thomas A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, 100(9):643–647, 1979. doi:10.1109/TC.1979.1675432.

7 Chan. On levels in arrangements of curves. *Discrete & Computational Geometry*, 29:375–393, 2003. doi:10.1007/S00454-002-2840-2.

8 Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

9 Dariusz Dereniowski, Aleksander Lukasiewicz, and Przemyslaw Uznanski. Noisy searching: Simple, fast and correct, 2023. arXiv:2107.05753.

10 Dariusz Dereniowski, Stefan Tiegel, Przemyslaw Uznanski, and Daniel Wolleb-Graf. A framework for searching in graphs in the presence of errors. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symp. on Simplicity in Algorithms (SOSA)*, volume 69, pages 4:1–4:17, 2019. doi:10.4230/OASICS.SOSA.2019.4.

11 Michael Dillencourt, Michael T. Goodrich, and Michael Mitzenmacher. Leveraging parameterized Chernoff bounds for simplified algorithm analyses. *Inf. Process. Lett.*, 187:106516, 2025. doi:10.1016/j.ipl.2024.106516.

12 Robert L. (Scot) Drysdale, III. A practical algorithm for computing the Delaunay triangulation for convex distance functions. In David S. Johnson, editor, *ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 1990. URL: http://dl.acm.org/citation.cfm?id=320176.320194.

13 Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graphics*, 9(1):66–104, 1990. doi:10.1145/77635.77639.

14 Ehsan Emamjomeh-Zadeh, David Kempe, and Vikrant Singhal. Deterministic and probabilistic binary search in graphs. In Daniel Wichs and Yishay Mansour, editors, *48th ACM Symp. on Theory of Computing (STOC)*, pages 519–532, 2016. doi:10.1145/2897518.2897656.

15 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, 1994. doi:10.1137/S0097539791195877.

16 Steven Fortune and Christopher J Van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Trans. Graphics*, 15(3):223–248, 1996. doi:10.1145/231731.231735.

17 Agata Fronczak and Piotr Fronczak. Biased random walks in complex networks: The role of local navigation rules. *Phys. Rev. E*, 80(1):016107, 2009. doi:10.1103/PhysRevE.80.016107.

18 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Optimal sorting with persistent comparison errors. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th European Symp. on Algorithms (ESA)*, volume 144 of *LIPIcs*, pages 49:1–49:14, 2019. doi:10.4230/LIPICS.ESA.2019.49.

19 Benoît Groz, Frederik Mallmann-Trenn, Claire Mathieu, and Victor Verdugo. Skyline computation with noisy comparisons. In Leszek Gasieniec, Ralf Klasing, and Tomasz Radzik, editors, *Combinatorial Algorithms*, pages 289–303, Cham, 2020. Springer. doi:10.1007/978-3-030-48966-3_22.

20 Joachim Gudmundsson and Martin P. Seybold. A tail estimate with exponential decay for the randomized incremental construction of search structures. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 610–626, 2022. doi:10.1137/1.9781611977073.28.

21 Leonidas J. Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *19th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 8–21, 1978. doi:10.1109/SFCS.1978.3.

22 Klaus Hinrichs, Jürg Nievergelt, and Peter Schorn. Plane-sweep solves the closest pair problem elegantly. *Inf. Process. Lett.*, 26(5):255–261, 1988. doi:10.1016/0020-0190(88)90150-0.

23 Kazuo Iwama, Rudy Raymond, and Shigeru Yamashita. General bounds for quantum biased oracles. *IPSJ Digital Courier*, 1:415–425, 2005. doi:10.2197/ipsjdc.1.415.

24 Kamil Khadiev, Nikita Savelyev, Mansur Ziatdinov, and Denis Melnikov. Noisy tree data structures and quantum applications. *Mathematics*, 11(22), 2023. doi:10.3390/math11224707.

25 Rolf Klein, Rainer Penninger, Christian Sohler, and David P. Woodruff. Tolerant algorithms. In Camil Demetrescu and Magnús M. Halldórsson, editors, *19th European Symp. on Algorithms (ESA)*, volume 6942 of *LNCS*, pages 736–747. Springer, 2011. doi:10.1007/978-3-642-23719-5_62.

26 D. T. Lee and F. P. Preparata. Location of a point in a planar subdivision and its applications. *SIAM J. Comput.*, 6(3):594–606, 1977. doi:10.1137/0206043.

27 Maarten Löffler. *Data Imprecision in Computational Geometry*. PhD thesis, Utrecht University, 2009.

28 Maarten Löffler and Jack Snoeyink. Delaunay triangulation of imprecise points in linear time after preprocessing. *Comput. Geom.*, 43(3):234–242, 2010. doi:10.1016/j.comgeo.2008.12.007.

29 Kurt Mehlhorn, Ralf Osbild, and Michael Sagraloff. Reliable and efficient computational geometry via controlled perturbation. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Int. Colloq. Automata, Languages and Programming (ICALP)*, volume 4051 of *LNCS*, pages 299–310. Springer, 2006. doi:10.1007/11786986_27.

30 Kurt Mehlhorn and Stefan Schirra. Exact computation with leda_real – theory and geometric applications. In Götz Alefeld, Jiri Rohn, Siegfried M. Rump, and Tetsuro Yamamoto, editors, *Symbolic Algebraic Methods and Verification Methods*, pages 163–172. Springer, 2001. doi:10.1007/978-3-7091-6280-4_16.

31 Ketan Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, 1994.

32 Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *J. Comput. System Sci.*, 23(2):166–204, 1981. doi:10.1016/0022-0000(81)90012-X.

33 Andrzej Pelc. Searching with known error probability. *Theoret. Comput. Sci.*, 63(2):185–202, 1989. doi:10.1016/0304-3975(89)90077-7.

34 Andrzej Pelc. Searching games with errors—fifty years of coping with liars. *Theoret. Comput. Sci.*, 270(1-2):71–109, 2002. doi:10.1016/S0304-3975(01)00303-6.

35 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.

36 Michael O. Rabin. Probabilistic algorithms. In J. F. Traub, editor, *Symp. New Directions in Algorithms and Complexity*, pages 21–39. Academic Press, 1976.

37 Alfréd Rényi. On a problem in information theory. *Magyar Tud. Akad. Mat. Kutató Int. Közl.*, 6:515–516, 1961.

38 Sandeep Sen. A unified approach to tail estimates for randomized incremental construction. In *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, pages 58–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

39 Sven Skyum. A sweepline algorithm for generalized Delaunay triangulations. *DAIMI Report Series*, 20(373), 1991.

40 Emanuele Viola. The communication complexity of addition. *Combinatorica*, 35(6):703–747, 2015. doi:10.1007/s00493-014-3078-3.

41 Ziao Wang, Nadim Ghaddar, Banghua Zhu, and Lele Wang. Noisy sorting capacity. *IEEE Transactions on Information Theory*, 70(9):6121–6138, 2024. doi:10.1109/TIT.2024.3425281.

42 Chee-Keng Yap. Symbolic treatment of geometric degeneracies. *J. Symbolic Comput.*, 10(3-4):349–370, 1990. doi:10.1016/S0747-7171(08)80069-7.

A Path-guided Pushdown Random Walks

A.1 Correctness and analysis

In this section we prove that w.h.p. the path-guided random walk terminates with the correct answer to its search problem (Theorem 1). To do so we first need to define a termination condition for the walk. It is not adequate to terminate merely after an appropriate number of steps: with constant probability, the final step of the walk will be taken after an erroneous oracle result, and may be incorrect. On the stack used to guide the algorithm, we store along with each vertex a *repetition count*, equal to one if the vertex is different from the previous vertex on the stack, and equal to the previous repetition count otherwise. We terminate the algorithm when this repetition count reaches an appropriately large value, $\Theta(\log(1/\varepsilon))$.

▶ **Lemma 9.** *If it does not terminate earlier, the path-guided random walk will reach the correct goal vertex t, within $\Theta(|P| + \log(1/\varepsilon))$ steps, with high probability, with a constant factor determined by the analysis below.*

Proof. Define a call to the transition oracle, T, in our random walk as "good" if it returns the correct answer (i.e., T does not lie) and "bad" otherwise, so that each call is bad independently with probability at most p_e. Note that if we are at a node, $v \in P$, and $v \neq t$, then a good call with either undo a previous bad call to stay at v or it will move to the next vertex in P. Also, if we are already at t, then a good call will keep us at t, adding another

copy of t to the top of the stack, S. Alternatively, if we are a node, $v \notin P$, then a good call with either undo a previous bad call to stay at v or it will move back to a previous node in our traversal, which undoes a previous bad call to move to v. Moreover, if we repeat this latter case, we will eventually return back to a node in P. Thus, every good call either undoes a previous bad call or makes progress towards the target vertex, t. Admittedly, if we are at a node, $v \in P$, then a bad call can undo a previous good call (e.g., which was to move to v or stay at v if $v = t$). Also, a sequence of bad calls can even deviate from P and return back to it – but because G is a DAG, a series of bad calls cannot return back to a previously visited vertex of P. Thus, a path-guided pushdown random walk will successfully reach the target vertex, t, if the difference between the number of good calls and bad calls is at least $|P|$, the length of the path, P. Let X_i be an indicator random variable that is 1 iff the ith call to the transition oracle is bad, and let $X = \sum_{i=1}^{N} X_i$. Since each call is bad independent of all other calls, we can apply a Chernoff bound to determine an upper bound on the probability that the difference between the number of good calls and bad calls is less than $|P|$, i.e., if $(N - X) - X < |P|$, that is, if $X > (N - |P|)/2$. Further, note that $\mu = E[X] = p_e N \leq (1/15)N$. Then, for $N = 3(|P| + \log(1/\varepsilon))$, the failure probability is

$$\begin{aligned}
\Pr\left(X > \frac{N - |P|}{2}\right) &= \Pr\left(X > |P| + \frac{3}{2}\log(1/\varepsilon)\right) \\
&= \Pr\left(X > \frac{1}{3}\left(3|P| + \frac{9}{2}\log(1/\varepsilon)\right)\right) \\
&\leq \Pr\left(X > \frac{1}{3}N\right).
\end{aligned}$$

Further, by a Chernoff bound from Dillencourt, Goodrich, and Mitzenmacher [11] (Theorem 7),

$$\Pr\left(X > \frac{1}{3}N\right) < 2^{-N/3} \leq 2^{-\log(1/\varepsilon)} = \varepsilon.$$

This establishes the proof. ◀

As mentioned above, the condition $p_e < 1/15$ can, with the same asymptotic performance as in Theorem 1 be replaced with any error probability bounded away from $1/2$. We caution, however, that while Theorem 1 provides a high-probability guarantee we reach the target vertex, t, it does not provide a high-probability guarantee we visit every vertex of the path, P. There is an exception to this, which we describe now.

▶ **Corollary 10.** *If G is a tree, then the path-guided pushdown random walk in G will visit every node in P.*

Proof. By definition, there is a unique path P^* from the root of G to the target vertex t. Thus, according to Theorem 1, with high probability after N steps, the path represented by the stack S must equal the intended path P^*, which also must equal P. ◀

▶ **Lemma 11.** *If it does not terminate earlier, the path-guided pushdown random walk will accumulate $\Theta(\log(1/\varepsilon))$ copies of the goal vertex t on its stack, after $\Theta(|P| + \log(1/\varepsilon))$ steps, and therefore terminate, with high probability.*

Proof. This follows by applying Lemma 9 to a modified DAG G' in which we expand each vertex v of G into a chain of copies $(v, 1)$, $(v, 2)$, $(v, 3)$, ... of v with a repetition count, as used in the termination condition of the algorithm. ◀

▶ **Lemma 12.** *The path-guided pushdown random walk will not terminate with any other vertex than t, with high probability.*

Proof. By another application of Chernoff bounds, and by the assumption that ε is polynomially small, the probability that the algorithm terminates at step i with an incorrect vertex is itself polynomially small, with an exponent that can be made arbitrarily large (independent of our original choice of ε) by an appropriate choice of the constant factor in the $\Theta(\log(1/\varepsilon))$ repetition count threshold used for the termination condition. By choosing this constant factor appropriately, we can make the probability of termination at any fixed step smaller than ε by a factor at least as large as the high-probability bound on the number of steps of the walk in Lemma 11. The result follows by applying the union bound to all the steps of the walk. ◀

Proof of Theorem 1. Termination of the algorithm w.h.p. follows from Lemma 11, and correctness once terminated follows from Lemma 12. For a given high-probability bound $1-\varepsilon$, we apply Lemma 11 and Lemma 12 with $\varepsilon/2$ and then apply the union bound to get a bound on the probability that the algorithm terminates with a correct answer. ◀

A.2 Counterexample to Generalizations

It is very tempting to consider a generalization of the path-guided pushdown random walk defined by a set of *valid nodes* and *goal nodes*, with the property that a non-noisy search starting from any valid node will reach a goal node and then stop. With a noisy oracle that determines whether a node is valid and if so follows a noisy version of the same search, one could hope that this generalized algorithm would quickly reach a goal state. The path-guided pushdown random walk would then be a special case where the valid nodes are exactly the nodes on the non-noisy search path from the root node. However, as we show in this section, this generalization does not work with the same fast high-probability termination bounds, unless additional assumptions are made (such as the assumptions giving the special case of the path-guided pushdown random walk).

Consider the following search problem: the DAG to be searched is simply a complete binary tree with height $\log_2 n$. One root-to-leaf path on this DAG is marked as valid, with the nodes on this path alternating between non-goal and goal nodes, so that the non-noisy search from the root stops after one step but other later stops would also produce a valid result. We have a noisy oracle with the following behavior:

- At a leaf node of the binary tree, or an invalid node, it always produces the correct result.
- At a valid non-leaf node, it follows the same behavior as a non-noisy oracle with probability 14/15: that is, it correctly identifies whether the node is a goal, and if not returns the unique valid child. With probability $K \log\log n/\log n$ (for some suitably large constant K) it returns the valid child regardless of whether the node is a goal. And with the remaining probability $1/15 - K\log\log n/\log n$ it returns the invalid child.

Now consider the behavior on this problem of a path-guided random walk, as defined above with a repetition-count termination condition. It will follow the valid path in the binary tree, with brief diversions whenever the noisy oracle causes it to walk to a node not on the path. At any goal node, it will wait at that node, accumulating more repetitions of the node, until either it achieves $C\log n$ repetitions (with a constant factor C determined by the desired high probability bound) or the noisy oracle tells it to take one more step along the valid path. The probability of reaching $C\log n$ repetitions without taking one more step is $(1 - K\log\log n/\log n)^{C\log n} \approx \exp -(K/C)\log\log n$, and by adjusting K relative

to C we can make this probability less than $1/(2 \log n)$. With this choice, by the union bound, the random walk will eventually take one more step from each goal node until finally reaching the leaf goal node. The expected number of steps that it waits at each goal node will be $\Theta(\log n / \log \log n)$, so the expected number of steps in the overall random walk, before reaching the leaf goal node and then accumulating enough repetitions to terminate, will be $\Theta(\log^2 n / \log \log n)$.

It also would not work to use a termination condition of making enough steps to have high probability of reaching a goal node, and then stopping. If this number of steps is not enough to reach the leaf goal node, the probability of stopping at a non-goal node would be too high.

Thus, without either a change of algorithm or more assumptions on the valid and goal node sets, this generalized random walk can be made to take a number of steps that is longer than $|P| + \log 1/\varepsilon$ by a non-constant factor.

B Transition Oracle for Binary Search Trees

While path-guided pushdown random walks applies to many DAGs, many fundamental computational geometry algorithms rely on binary search trees. In this appendix, we present a transition oracle for BSTs.

Say that we are attempting to find some value x in a binary search tree T. By properties of BSTs, there is a unique path from the root to the node containing x. If there is no such node, there is still a unique path to the nonexistent leaf that would contain x. When a call to the transition oracle T is good, we expect it to correctly determine if we are on said valid path only using a constant number of comparisons. To do so, we add a constant amount of extra data to each node.

For each node v of the BST, let P_v be the unique path from the root to v. Let $v.l$ be the lowest ancestor of v whose right child is in P_v (or a sentinel representing $-\infty$). Likewise, let $v.r$ be the lowest ancestor of v whose left child is in P_v (or a sentinel representing ∞). One of $\{v.l, v.r\}$ is v's parent. Observe that, if we have correctly navigated to v, x must be in between the values held by $v.l$ and $v.r$. Thus, by adding two pointers to each node, the transition oracle can determine if we are on the correct path using two comparisons. See Figure 5 for a visual depiction of the pointers $v.l$ and $v.r$ and their effect. Augmenting the tree with these pointers can be done in $O(\log n)$ extra time per insertion and deletion. With slight modification, we can also support inexact queries such as for predecessors or successors.

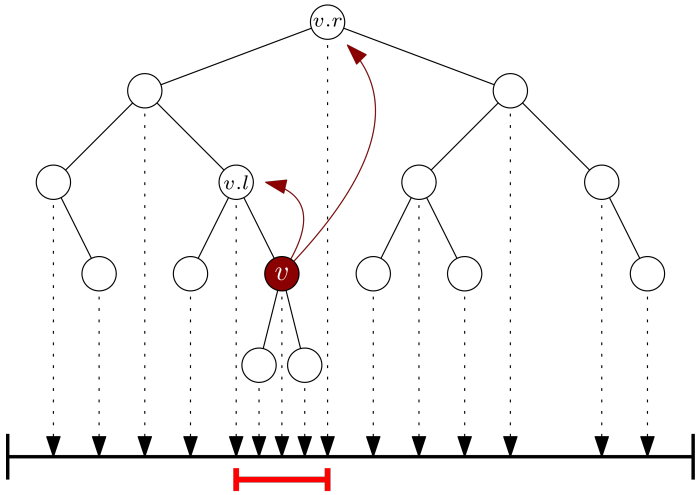


Figure 5 Here we have depicted the $v.l$ and $v.r$ pointers for the node v. Notice that $v.l$ is v's first ancestor whose value is smaller than v's. Likewise, $v.r$ is its first ancestor whose value is larger than v's. The dotted lines map nodes to their relative ordering on a number line. It is clear that the values held by $v.l$ and $v.r$ form the interval of possible values that could be held by nodes in v's subtree. If the query is located strictly in this interval and if it is in the tree at all, then it must be within v's subtree. If the query is not in this interval, then it cannot be in v's subtree and we have made an errant comparison earlier.

An Efficient Polynomial Time Approximation Scheme for Minimizing the Total Weighted Completion Time on Uniformly Related Machines

Leah Epstein
Department of Mathematics, Faculty of Natural Sciences, University of Haifa, Israel

Asaf Levin
Faculty of Data and Decisions Science, The Technion, Haifa, Israel

Abstract

We study a classic scheduling problem on *uniformly related machines* for which we show an efficient polynomial time approximation scheme (EPTAS), where an EPTAS is a fast and practical approximation scheme. For a desired approximation ratio of $1+\varepsilon$ for $\varepsilon > 0$, the running time of an EPTAS is a function of ε multiplied by a polynomial function of the input length. New methods and techniques are essential in developing such improved approximation schemes, and their design is a primary goal of this research agenda. We present an EPTAS for the scheduling problem of a set of jobs on uniformly related machines so as to minimize the total weighted completion time. The problem is NP-hard in the strong sense, and therefore an EPTAS is the best possible approximation scheme for the problem, unless P=NP. Prior to our work, only a PTAS was known for the problem, while an EPTAS was known only for the special case of identical machines.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Scheduling algorithms, Approximation schemes, Min-sum objective

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.25

Funding *Asaf Levin*: Partially supported by ISF – Israel Science Foundation grant number 1467/22.

1 Introduction

We consider one of the most basic multiprocessor scheduling problems: scheduling on uniformly related machines with the goal of minimizing the total weighted completion time of jobs. More precisely, our scheduling problem is defined as follows. We are given a set of n jobs, where each job has a positive size and a positive weight associated with it. We are also given a set of m machines to be used for the processing of jobs, such that each machine has a given positive speed. Running job j on machine i requires the allocation of a time interval on this machine, whose length is precisely the size of j divided by the speed of i, which is called the processing time of j on i.

We consider non-preemptive schedules and thus every job is assigned to a machine and to a single (continuous) time slot (or interval) on that machine, under the following conditions. The length of the time slot assigned to job j (on one of the machines) has to be the processing time of j on that machine. A machine can process at most one job at each time so the time intervals assigned to two jobs that are assigned to a common machine do not intersect in an inner point. Given such a schedule, the completion time of job j is defined as the ending point of the time interval of j, and the weighted completion time of j is the product of its weight and its completion time. The goal is to find a schedule for which the sum of the weighted completion times of all jobs.

In the scheduling community there is a consensus that the objective of weighted sum of job completion times is the most important min-sum objective, and together with the makespan minimization these are the two most important objective functions in the scheduling literature.

© Leah Epstein and Asaf Levin;
licensed under Creative Commons License CC-BY 4.0

A special case of our problem is that of identical machines, where each machine has speed 1. A generalization of these problems is the one where each job also has a release date associated with it, which restricts its starting time such that it cannot start running before its release date. Here, all release dates are equal to zero. Another well-motivated special case of our problem is the unweighted one, where all weights are equal to 1.

We study here a core problem in scheduling theory. We refer to [13, 27, 16, 29, 35, 5] for some recent work on the problem with respect to various aspects.

Now, we define the notions of approximation algorithms and the different types of approximation schemes. An $\mathcal{R}$-approximation algorithm for a minimization problem is a polynomial time algorithm that always finds a feasible solution of cost at most $\mathcal{R}$ times the cost of an optimal solution. The infimum value of $\mathcal{R}$ for which an algorithm is an $\mathcal{R}$-approximation is called the approximation ratio or the performance guarantee of the algorithm. A polynomial time approximation scheme (PTAS) is a family of approximation algorithms such that the family has a $(1+\varepsilon)$-approximation algorithm for any $\varepsilon > 0$. An efficient polynomial time approximation scheme (EPTAS) is a PTAS whose time complexity is of the form $f(\frac{1}{\varepsilon})$ multiplied by $poly(n)$ where f is some (not necessarily polynomial) computable function, and $poly(n)$ is a polynomial function of the length of the (binary) encoding of the input. A fully polynomial time approximation scheme (FPTAS) is a stronger concept, defined like an EPTAS, but the function f must be a polynomial in $\frac{1}{\varepsilon}$.

In this paper, we are interested in EPTAS's and we say that an algorithm (for some problem) has a *polynomial running time complexity* if its time complexity is of the form $f(\frac{1}{\varepsilon}) \cdot poly(n)$. Note that while a PTAS may have time complexity of the form $n^{g(\frac{1}{\varepsilon})}$, where g can be polynomial or even super-exponential, this cannot be the case for an EPTAS. The notion of an EPTAS is modern and finds its roots in the FPT (fixed parameter tractable) literature (see [7, 9, 14, 31]). It was introduced in order to distinguish practical from impractical running times of PTAS's, for cases where an FPTAS does not exist (unless P=NP). In this work, we design an EPTAS for the scheduling problem defined above for which an FPTAS cannot exist unless P=NP.

The seminal work of Smith [34] established the existence of a polynomial time algorithm (of time $O(n \log n)$) for solving the problem of minimizing the total weighted completion time on a single machine. This algorithm can be described as follows. The jobs are scheduled according to a non-increasing order of their densities, starting at time zero, and without any idle time (where the density of job j is the ratio between its weight and its size). This ratio is called Smith's ratio, and any tie breaking policy leads to an optimal solution. The correctness of this algorithm follows by a simple exchange argument. This algorithm generalizes the SPT (shortest-processing-time) approach for the case of equal weights. In our settings, we can conclude the following property for the problem. Once the jobs have been assigned to machines (but not to time slots), the permutation of jobs assigned to a given machine is fixed according to Smith's algorithm.

For a constant number of machines (at least two machines), the problem is NP-hard in the ordinary sense, but it is solvable in pseudo-polynomial time and has an FPTAS [32]. For the case where the number of machines is a part of the input, the problem is strongly NP-hard (see e.g. problem SS13 in [15]). The case of equal weights is polynomially solvable even for the more general case of unrelated machines [22]. The property that our problem is strongly NP-hard excludes the possibility to design an FPTAS for it, and thus an EPTAS is the best possible result (unless P=NP).

The development of good approximation algorithms for problems studied here was fairly slow. Till the late 1990's, only constant approximation algorithms were developed for min-sum scheduling problems such as the ones we study. We refer to the papers cited in

[33, 1, 8] for a survey of such results. Here we elaborate on the approximation schemes in the literature. The first approximation scheme for a special case of our problem was introduced by Skutella and Woeginger [33], which was designed for the special case of identical machines. Moreover, the last scheme is in fact an EPTAS for this special case. Shortly afterwards, Afrati et al. [1] presented a PTAS for the problem on identical machines with release dates. Their approach was generalized by Chekuri and Khanna [8], who showed the existence of a PTAS for uniformly related machines with release dates (see also [2]). The discussed schemes of [1, 8] are not EPTAS's, and they cannot be converted into EPTAS's (using known methods) even for the case without release dates studied here.

Before explaining the methods and techniques of these last schemes, as well as the limitations of those approaches, we mention the state of the art of approximation schemes for general load balancing problems on identical machines and uniformly related machines. The relation between the problem studied here and this family of problems will become clear later. In load balancing problems, the goal is to minimize functions of the vector of *machine* completion times (and not job completion times as we study here). Hochbaum and Shmoys presented the dual approximation framework and used it to show that the makespan minimization problem has a PTAS for identical machines [20] and for uniformly related machines [21]. In the dual approximation framework, the problem is converted into a bin packing type problem by fixing the machine loads. It was noted in [18] that the PTAS of [20] for identical machines can be converted into an EPTAS by using an integer program in fixed dimension instead of dynamic programming. Jansen [23] developed an EPTAS for the makespan minimization problem on uniformly related machines (see [24] for an alternative EPTAS for this problem and [25, 6] for improved time complexity EPTAS's for this problem). The ℓ_p norm minimization problem (of the vector of machine completion times) has an EPTAS for identical machines [3, 4], and a PTAS and an EPTAS for uniformly related machines [10, 11]. The EPTAS for the ℓ_p norm minimization problem on uniformly related machines presented in [11] provides a method for constructing an EPTAS for our problem if all job densities are equal. However, this does not translate into an EPTAS for the problem even if there are only a constant number of distinct job densities. After the preliminary version of our work here was posted [12], Kones and Levin [28] have used the framework we present here for the case of constant number of job densities in order to establish an alternative EPTAS for the ℓ_p norm minimization problem on uniformly related machines, and they used this framework to approximate a generalization of this problem.

Previous directions. Next, we elaborate on the known approximation schemes for related problems. Skutella and Woeginger [33] observed that for identical machines the ℓ_2 norm minimization of the vector of machine loads is equivalent to minimizing the total weighted completion time, if the jobs have equal densities. This equivalence means that an optimal solution to one problem is an optimal solution to the other as the values of the two objective functions differ by an additive constant (which is common to all feasible solutions). Using this approach, they showed that if the ratio between the maximum density and the minimum density of jobs is upper bounded by a constant, then one can adapt the ideas of Alon et al. [3, 4] and obtain an EPTAS for this restricted setting.

The scheme [33] for the problem on identical machines follows the next ideas. First, round all the job sizes and job weights (up or down) to integer powers of $1+\varepsilon$. Next, apply randomized geometric partitioning of the jobs based on their (rounded) density, solve each sub-instance consisting of all jobs of one partition using the scheme (which is similar to the one of [3, 4]), and schedule the partial solutions for every machine sorted by non-decreasing

job densities. This last step of combining the solutions for the different parts in the partition was more delicate in [33], but as noted in [1], the last step could be made much simpler. Afrati et al. [1] also noted that this approach can be extended to obtain an approximation scheme (a PTAS) for the problem without release dates in other machine environments (see the last remark in [1]). Chekuri and Khanna [8] extended the techniques of [1] to the setting of uniformly related machines (with release dates). The methods of [1, 8] fail completely when one tries to obtain an EPTAS for these problems with or without release dates (even for identical machines).

Our results. An EPTAS for the total weighted completion time on uniformly related machines was not known prior to our work, and it was only known for the much simpler case of identical machines. We design such an EPTAS here, using a number of methods explained in the next section. In an accompanying article (see the second part of [12]), we design an EPTAS for total weighted completion time on uniformly related machines with release dates. The current EPTAS is designed for the important special case without release dates, and while this EPTAS requires new ideas, this scheme is significantly simpler than the one of the general case. Before presenting the detailed description of the scheme and its analysis, we present an overview of the scheme. A forthcoming full version will contain additional details and omitted proofs.

2 An Overview of Our Scheme

In our scheme, all parameters are rounded to integer powers of a parameter $1+\delta$ (where $\delta = \frac{\varepsilon}{\Upsilon}$ for a constant $\Upsilon > 0$ that is independent of ε) [33, 1]. Thus, the first step is rounding the job sizes, the job weights, and the speeds of the machines to integer powers of $1+\delta$. Observe that this step has a minor effect on the performance guarantee of the approximation algorithm, and it is used to structure the input in the later steps. This is a standard approach which is suitable here. We say that a set of values is discrete if for every interval $[L, U]$ (for every $0 < L < U$) we have at most $O(\log \frac{U}{L})$ distinct values of the set in the interval. Then, we would like to consider discrete values for sizes, weights, and speeds, which gives us also discrete densities and discrete job processing times on the machines.

Next, we use the shifting technique [19] by modifying the weights of some jobs to separate the input into *bounded* instances in terms of job sets. For each bounded instance, it will hold that its jobs have sufficiently similar densities. We eliminate intervals of densities from the input (and not intervals of weights). This is done by increasing weights of jobs whose densities belong to intervals that will be forbidden, so not just one interval of densities is removed but a regular pattern of densities is removed, creating gaps between density intervals. Finding the best sequence of gaps requires running our algorithm for all options.

Once there are regular gaps between density intervals, we show that each instance can be solved separately and independently from other instances, and the solutions can be combined without further modifications. That is, we apply an approach that is similar to [33], where we replace the randomized geometric partitioning by the stronger shifting technique (resulting in a slightly simpler analysis as one of the cases in [33] of jobs of similar densities in different instances is avoided, because there will be no such cases). In our shifting procedure, we ensure the following for a pair of jobs j and j' such that the density of j is not smaller than the density of j'. If j and j' belong to a common bounded instance, then the ratio between the density of j and the density of j' is upper bounded by a constant (that depends on ε), while if j and j' belong to different bounded instances the ratio between the two

densities is very large (at least a larger constant that also depends on ε). When we obtain an approximated solution for each such bounded instance, we simply concatenate the solutions by processing each job on the machine it is assigned to (in the solution to its bounded instance) and processing the jobs assigned to each machine according to their Smith's ratios. By the separation of densities, we justify this, and conclude that the cost of the resulting solution is only slightly larger than the sum of costs of the solutions to the bounded instances. Therefore, if we are able to ensure a $(1+\delta)$-approximation for each bounded instance, we will get an EPTAS for the instance resulting from this shifting technique.

The new part, where we cannot follow previous work, is where we solve each bounded instance (due to shifting, there is a polynomial number of such instances). Unlike the approach in [33], since we deal with machines of possibly different speeds, we use configuration mixed-integer linear programs (MILPs), where we require a variable of a machine configuration to be integral if the machine is heavily loaded, that is, if it receives a large total size of jobs. A configuration is a list of jobs assigned to one machine of a certain speed, where the list of relatively small jobs is not defined precisely, but the total size is within a given short interval. In order to define the notion of a heavily loaded machine, we "guess" the (approximate) load of the fastest machine (breaking ties in favor of low indices) counting the job set of the current bounded instance. Then, for the correct guess we can upper bound the load of each one of other machines (where the load of a machine is its completion time, that is the work divided by the speed, where the work of a machine is the total size of jobs of the current bounded instance assigned to it). In this way, we obtain a valid upper bound on the work of every other machine. This upper bound on the load of every machine or on its work could be seen as an alternative to the *dual approximation method* of Hochbaum and Shmoys [20, 21] for cases where this dual approximation method cannot be applied. In that case there is one upper bound for the completion time of every machine, which is not possible here since our objective is not the makespan.

We say that a machine is heavily loaded if its assigned work is at least a constant fraction of this upper bound. By splitting the machine set into slow and fast machines, we show that a slow machine is never heavily loaded in optimal solutions. Moreover, as mentioned earlier there is a limit to how much a fast machine can be loaded (in terms of the load of the first fastest machine, which we guess approximately), and as a result, the number of heavy configurations is a constant.

A machine configuration consists of the speed of the machine having this configuration, the load of this machine, and a complete list of relatively large jobs together with their densities (the jobs are large compared to the total size of jobs assigned to this machine) where a job is specified by its size and weight but not by its identity, and additionally, a configuration contains an approximate total size of smaller jobs of each density. The partition of jobs to small and large (for a given configuration) is based on both the load and on the speed of that machine (it is not based only on the speed, in the way that it is defined in many approximation schemes). This property is essential since this problem does not allow the use of the dual approximation method. The assignment of jobs that are scheduled as small jobs is carried out by another set of assignment variables. These last assignment variables can be fractional in a feasible solution to the MILP. The job sets that are small for their configurations are different for distinct configurations due to different load and speed. Such jobs are assigned (fractionally) to configurations. Thus, for every job type (which consists of a rounded size and a rounded density), there are variables counting how many copies of the job are assigned as large jobs to each configuration (so every machine scheduled based on such a configuration receives this number of jobs of this type) and how many jobs are

assigned as small jobs to such a configuration (in which case the job is not assigned to a specific copy of the configuration). We find that no jobs are treated as "sand" and every job is assigned to a configuration since (due to speeds) jobs cannot be just partitioned into small and large, but assigning jobs to specific copies of configurations will not allow us to obtain the required running time. It is required to know the speed of the machine that will receive a certain job, because the objective of the MILP has only a part of the cost incurred by small jobs, and the residue has to be added (where it depends on the speed).

As in [33], we approximate the increase of cost due to the contribution of slightly larger set of small jobs using the total size of jobs of the configuration (in the sense that the increase is minor compared to this total size), and thus, the usage of a fractional assignment becomes possible, and it can be converted in turn into an integral one by slightly increasing the cost of each configuration. As for fractional configurations, we round down the number of machines with each such configuration. The total size of jobs that remain unassigned is sufficiently small to be combined into the schedule of a heavily loaded machine (at least one such fastest machine must exist, and this is sufficient for our purposes). Using the fact that we solve a bounded instance, we conclude that slightly increasing the load of a machine with respect to its value in the optimal solution to the MILP has a minor impact on the cost of the resulting solution.

Our scheme for the bounded instance is of independent interest as it extends the methods of [23] for the ℓ_2-norm minimization problem (the case of one density for all jobs) to obtain an EPTAS for a constant number of densities (where the constant depends on δ), whereas extending the original EPTAS for this ℓ_2-norm minimization problem [11] results in (only) a PTAS for the case of constant number of densities. Thus, presenting a new EPTAS for the ℓ_2-norm minimization problem is indeed necessary for our generalization. We refer to Section 3 for a complete description of the approximation scheme explained in this overview, and we conclude that there is an EPTAS for our problem.

3 The Detailed Description and Formal Analysis of Our EPTAS

In this section we present our scheme and prove its correctness.

Properties. Obviously, since any job can start running at any time, an optimal solution (or schedule) never introduces idle time, and moreover, every machine runs its assigned jobs in an optimal order, that is, the jobs are sorted by Smith's ratios. As any tie-breaking policy leads to the same objective function value, we use a specific tie-breaking policy in this section. More specifically, we will always break ties in favor of running larger jobs first, and in the case of equal size jobs (of the same weight), jobs of smaller indices are scheduled earlier. We call this ordering *the natural ordering*. Thus, we can define a *solution* or a schedule as a partition of the jobs to m subsets, one subset for each machine, and the order for each machine will always be the natural one. In some cases, we will compute the total weighted completion time of another permutation (not of the natural ordering). This will be done for the sake of analysis when this calculation is easier for cases where we are interested in an upper bound on the objective value and this upper bound is sufficient.

For an input X and a solution B, we let $B(X)$ denote the output and the objective value of B for the input X. Recall that for a job j, the density of j is the ratio between its weight and its size. Thus, running the jobs sorted according to the natural ordering is equivalent to first sorting them by non-increasing density, and for each density, the jobs are sorted by non-increasing size, breaking ties (among equal size jobs) in favor of scheduling jobs of smaller indices earlier.

For a fixed schedule, the work of machine i is the total size of jobs assigned to it, and its load is its completion time, that is, the work divided by the speed (as we only consider schedules without any idle time). Let the original input instance be denoted by A, where job j has size $a_j > 0$ and weight $\omega_j > 0$, and machine i has speed $v_i > 0$.

Let C_j denote the completion time of j under a given schedule, that is, the total size of jobs that run before j on the same machine (including j), divided by the speed of this machine. We let Γ_j be defined as $\omega_j(C_j - \frac{a_j}{2v_i})$, where $C_j - \frac{a_j}{2v_i}$ is the time when half of job j is completed (known as the mean busy time of job j [17]). We call these values Γ-values, and obviously, the cost of a solution is at least the sum of Γ-values. Moreover, for identical machines, the difference between the cost of a solution and the sum of the Γ-values is independent of the solution (see also [33]) whereas for uniformly related machines, this difference depends on the speeds as we establish in the following lemma.

▶ **Lemma 1.** *Consider a set of jobs $\tilde{I} \subseteq A$ assigned to machine i. Let $\Phi = \sum_{j\in\tilde{I}} a_j$ and let $\phi = \min_{j\in\tilde{I}} \frac{\omega_j}{a_j}$ be the minimum density of any job of $\tilde{I}$. The sum of Γ-values (and the cost) of any solution that selects to run all jobs of $\tilde{I}$ on machine i (possibly with other jobs) is at least $\frac{\phi\cdot\Phi^2}{2v_i}$.*

Rounding. Given the original instance A, where job j has size a_j and weight ω_j, and machine i has speed v_i, we create a (rounded) instance A' as follows. Let $0 < \delta \leq 1/8$ be an accuracy factor, that is a function of ε (where $\delta < \varepsilon$), and such that $\frac{1}{\delta}$ is an integer. The sets of jobs and machines in A' are the same as in A. Let $s_i = (1+\delta)^{\lfloor \log_{1+\delta} v_i \rfloor}$ be the speed of machine i in instance A'. Let $w_j = (1+\delta)^{\lceil \log_{1+\delta} \omega_j \rceil}$, and $p_j = (1+\delta)^{\lceil \log_{1+\delta} a_j \rceil}$ be the weight and size (respectively) of job j in instance A'. That is, we round up the weights and the sizes of jobs while we round down the speeds of the machines. We will sometimes use $\Theta_j = \log_{1+\delta} \frac{w_j}{p_j}$, which is always an integer due to the rounding.

Next, we bound the increase of the approximation ratio due to this step. Our goal is to conclude that we can safely consider A' instead of A. Let $\mathcal{O}$ and $\mathcal{O}'$ denote optimal solutions for A and A' respectively. Let SOL denote a given solution for both instances, where a solution consists of a partition of the set of jobs to the machines. The ordering for each machine may be different for the two inputs since the natural ordering is used. Using the new notation, for a given schedule, the completion time of j is still denoted by C_j, and its weighted completion time is $w_j \cdot C_j$.

▶ **Proposition 2.** *We have $SOL(A) \leq SOL(A') \leq (1+\delta)^3 \cdot SOL(A)$ and $\mathcal{O}(A) \leq \mathcal{O}'(A') \leq (1+\delta)^3 \cdot \mathcal{O}(A)$. Furthermore, if a solution SOL satisfies $SOL(A') \leq (1+k\delta)\cdot\mathcal{O}'(A')$ for some $k > 0$, then it holds that $SOL(A) \leq (1+(2k+4)\delta)\cdot\mathcal{O}(A)$.*

By the last proposition, we only consider the instance A', and use the obtained schedule as an output for A. Thus, it suffices to present an EPTAS for the rounded instance A'.

On the use of Γ-values for A'. Given the input A', if j is executed on machine i, then $\Gamma_j = w_j(C_j - \frac{p_j}{2s_i})$ where $C_j - \frac{p_j}{2s_i}$ is the time when half of the job is completed. We say that for a given schedule, a block of jobs is a set of jobs of equal density that are assigned to one machine to run consecutively on that machine (there may be additional jobs of the same density, each assigned to run later or earlier than these jobs). The next well-known lemma shows that the sum of Γ-values for a block of jobs is a function of their total size, their common density, and the starting time of the block, and it is independent of the other properties of these jobs (but it depends on the speed of the machine). We prove the lemma for completeness similarly to the proof of Lemma 1.

▶ **Lemma 3.** *Let $\bar{I} \subseteq A'$ be a set of jobs, all having densities equal to some $\Delta > 0$, and assume that these jobs are scheduled to run consecutively on machine i starting at time τ. Then $\sum_{j\in\bar{I}} \Gamma_j = \Delta \cdot \left(\tau + \frac{1}{2s_i}\sum_{j\in\bar{I}} p_j\right) \cdot \left(\sum_{j\in\bar{I}} p_j\right)$.*

Shifting. Based on the value of δ we define a new parameter ξ for the shifting step. Its value is a constant (as a function of δ). Let $\xi = \lceil \ell \cdot \log_{1+\delta} \frac{1}{\delta} \rceil = \lceil \log_{1+\delta} (\frac{1}{\delta})^{\ell} \rceil$ for a fixed integer $3 \leq \ell \leq 5$. Thus, $\frac{1}{\delta^3} \leq \frac{1}{\delta^\ell} \leq (1+\delta)^\xi < \frac{1+\delta}{\delta^\ell} < \frac{2}{\delta^\ell} < \frac{1}{\delta^{\ell+1}} \leq \frac{1}{\delta^6}$. Next, we use the integrality of $\frac{1}{\delta^2}$ and of ℓ. We have $(1+\delta)^{\frac{\ell}{\delta^2}} > (1+\delta\cdot\frac{1}{\delta^2})^\ell > (\frac{1}{\delta})^\ell$, and we have $\xi \leq \frac{\ell}{\delta^2} < \frac{1}{\delta^3}$. However, $(\frac{9}{8})^\xi \geq (1+\delta)^\xi \geq \frac{1}{\delta^\ell} \geq 8^3$, so $\xi \geq 53$.

We form a partition of integers based on the value of ξ, such that every subset has ξ consecutive integers. For an integer $c \in \mathbb{Z}$, let $\Omega_c = \{c\cdot\xi + 1, \ldots, (c+1)\cdot\xi\}$. The next step will be to partition the collections of integers in a cyclic way. There will be $\frac{1}{\delta^{\ell+1}}$ possible collections. Let $0 \leq \zeta \leq \frac{1}{\delta^{\ell+1}} - 1$ be an integer. The collection for the value of ζ will consist of indices of sets for which the index modulo $\frac{1}{\delta^{\ell+1}}$ is equal to ζ. We will be interested in densities that are powers of $1+\delta$ that belong to Ω_c where it holds that $(c \bmod \frac{1}{\delta^{\ell+1}}) = \zeta$, and the densities for jobs with densities that belong to such a set will be increased by a constant multiplicative factor.

We define the instance A_ζ by modifying the weights of some jobs in A'. For a job j, if for some (not necessarily positive) integer v, $\Theta_j \in \Omega_{v/\delta^{\ell+1}+\zeta}$ (recall that the density of j is an integer power of $1+\delta$ and $\Theta_j = \log_{1+\delta}\frac{w_j}{p_j}$), then $w_j^\zeta = w_j \cdot (1+\delta)^\xi$, and otherwise $w_j^\zeta = w_j$. We let $\Theta_j^\zeta = \log_{1+\delta}\frac{w_j^\zeta}{p_j}$ which means that in the first case $\Theta_j^\zeta = \xi\cdot\Theta_j$ and in the second case $\Theta_j^\zeta = \Theta_j$. In the first case, $\Theta_j^\zeta \in \Omega_{v/\delta^{\ell+1}+\zeta+1}$ (where $v/\delta^{\ell+1}+\zeta+1 < (v+1)/\delta^{\ell+1}+\zeta$, so it was moved to a different density collection out of the $\frac{1}{\delta^{\ell+1}}$ collections). Furthermore, weights (and densities) are increased by a multiplicative factor of at most $(1+\delta)^\xi \leq \frac{1+\delta}{\delta^\ell}$. For any ζ, let $\mathcal{O}^\zeta$ be an optimal solution for A_ζ. As the set of jobs and machines is the same in A, A', and A_ζ, the sets of feasible solutions for the three instances are the same.

Bounding the increase of the cost due to modifying A' into A_ζ. Next, we bound the increase of the cost due to the transformation from A' to A_ζ. As a result of increasing weights for jobs with densities in a fixed density collection, no job $j \in A_\zeta$ has a density such that $\Theta_j^\zeta = \log_{1+\delta}\frac{w_j^\zeta}{p_j} \in \Omega_{v'/\delta^{\ell+1}+\zeta}$ for any integer v'. Any value $(1+\delta)^\beta$ where $\beta \in \Omega_{v/\delta^{\ell+1}+\zeta}$ for an integer v is called a forbidden density for ζ, and other values $(1+\delta)^{\beta'}$ for integer values of β' are called allowed density for ζ. The next lemma compares costs for $\mathcal{O}'(A')$ (which was defined to be an optimal solution for A') and $\mathcal{O}^\zeta$ for specific values of ζ (for the corresponding input A_ζ). This will allow us to guess a suitable value of ζ (by enumeration) and approximate $\mathcal{O}^\zeta$ rather than $\mathcal{O}'$.

▶ **Lemma 4.** *Given a solution SOL, any $0 \leq \zeta \leq \frac{1}{\delta^{\ell+1}} - 1$ satisfies $SOL(A') \leq SOL(A_\zeta)$, and there exists a value $0 \leq \bar{\zeta} \leq \frac{1}{\delta^{\ell+1}} - 1$ such that we have $SOL(A_{\bar{\zeta}}) \leq (1+2\delta)\cdot SOL(A')$. Additionally, there exists a value $0 \leq \zeta' \leq \frac{1}{\delta^{\ell+1}} - 1$ such that $\mathcal{O}'(A') \leq \mathcal{O}^{\zeta'}(A_{\zeta'}) \leq (1+2\delta)\cdot \mathcal{O}'(A')$, and if a solution SOL_1 satisfies $SOL_1(A_{\zeta'}) \leq (1+k'\delta)\cdot\mathcal{O}^{\zeta'}(A_{\zeta'})$ for some $k' > 0$, then $SOL_1(A') \leq (1+(2k'+2)\delta)\cdot\mathcal{O}'(A')$.*

The exhaustive enumeration algorithm implementing the shifting step. Down below we present an algorithm that receives an input where the ratio between the maximum density of any job and the minimum density of any job is constant (as a function of δ), and outputs a solution of cost at most $1+\delta$ times the cost of an optimal solution for this input. The

ratio between densities will be at most $(1+\delta)^y$, where $y = \frac{\xi}{\delta^{\ell+1}} - \xi - 1$, The motivation for the value of y is that every density collection has ξ different densities, and since there are $\frac{1}{\delta^{\ell+1}}$ options for ζ, there are $\frac{1}{\delta^{\ell+1}} - 1$ options of ζ for which the density collections were not removed. Sequences of possible densities without gaps introduced by considering A_ζ for a fixed value of ζ will contain $\xi \cdot (\frac{1}{\delta^{\ell+1}} - 1) = y + 1$ densities which are integer powers of $1+\delta$, such that the largest one will be larger than the smallest one by a multiplicative factor of at most $(1+\delta)^y$.

We next use this algorithm as a black box. For every value of ζ, we apply the following process and create a schedule for the input A_ζ. Afterwards, we choose a solution of minimum cost among the $\frac{1}{\delta^{\ell+1}}$ resulting solutions.

Let $0 \leq \zeta \leq \frac{1}{\delta^{\ell+1}} - 1$. Decompose the allowed densities for ζ into collections of consecutive densities that are separated by intervals of forbidden densities for ζ. Since densities were rounded, two densities are consecutive if the large one is larger by a factor of exactly $1+\delta$ from the smaller one. By our construction, this results in subsets of allowed densities with very different densities, such that each subset has allowed densities in an interval of the form $[(1+\delta)^{-y}\rho, \rho]$, where $\rho = (1+\delta)^{(v/\delta^{\ell+1}+\zeta)\xi}$ for some integer v (this is the last density in the density collection just before $\Omega_{v/\delta^{\ell+1}+\zeta}$), y is as defined above ($y = \frac{\xi}{\delta^{\ell+1}} - \xi - 1$), and additionally there is a gap between the allowed densities of one set and another set (because there are no jobs with densities in $\Omega_{v/\delta^{\ell+1}+\zeta}$ in A_ζ). More precisely, if I and I' are two such subsets and the allowed densities for I are smaller than those of I', then the largest allowed density for I is smaller by a multiplicative factor of $(1+\delta)^{\xi+1} \geq \frac{1+\delta}{\delta^\ell}$ than the smallest allowed density of I'.

A sub-instance is defined to be a non-empty subset of the jobs corresponding to an interval of allowed densities together with the complete set of machines. Let q denote the number of such (sub-)instances (where $q \leq n$). Let the instances be denoted by $I_1, \ldots, I_q$, such that the maximum allowed density in I_p is ρ_p, and for $p > 1$, $\rho_p > \rho_{p-1}$, and in fact $\rho_p \geq (1+\delta)^{\frac{\xi}{\delta^{\ell+1}}} \cdot \rho_{p-1} \geq (\frac{1}{\delta^\ell})^{\frac{1}{\delta^{\ell+1}}} \cdot \rho_{p-1}$.

Given a set of solutions SOL_p, for $1 \leq p \leq q$ (where SOL_p is a solution for I_p), define a *combined solution* SOL, where the jobs assigned to machine i in SOL are all jobs assigned to this machine in all the solutions. Obviously, in SOL the jobs are scheduled sorted by non-increasing indices of their sets I_p.

Bounding the cost of the combined solution. We will prove that the effect of concatenating solutions is not harmful even though earlier solutions delay the processing of later solutions. This will hold due to the gap between densities of jobs of different instances.

▶ **Lemma 5.** *We have $OPT(A_\zeta) \geq \sum_{p=1}^{q} OPT(I_p)$. Furthermore, if for every $1 \leq p \leq q$ it holds that $SOL_p(I_p) \leq (1+\nu) \cdot OPT(I_p)$ for a fixed value $\nu > 0$, then the combined solution SOL satisfies $SOL(A_\zeta) \leq (1+\nu) \cdot (1+8\delta) \cdot OPT(A_\zeta)$.*

In what follows we design a $(1+\delta)$-approximation algorithm for the bounded ratio problem, that gives a $(1+\delta)(1+8\delta)$-approximation algorithm for A_ζ. Since $(1+\delta)(1+8\delta) < 1+10\delta$, for an appropriate choice of ζ, we get an $(1+22\delta)$-approximation algorithm for A', and a $(1+48\delta)$-approximation algorithm for A. Thus, letting $\delta = \frac{\varepsilon}{48}$ will give a $(1+\varepsilon)$-approximation algorithm for the general problem. The algorithm for the bounded ratio problem is applied at most n times for each choice of ζ, i.e., at most $\frac{n}{\delta^6}$ times in total. The reason is that we split every input A_ζ into parts with similar densities, and the number of parts cannot exceed the number of jobs.

An EPTAS for the bounded ratio problem. Let I be a bounded instance such that all densities are in $[1, \rho = \rho(\delta) = (1+\delta)^y]$, where y is a function of δ (y and ξ are as defined in the previous section). We will use not only the property that the number of distinct densities is a constant but also the constant maximum ratio between densities. By scaling (and possibly increasing the interval of densities), any valid input of the bounded ratio problem can be transformed into this form. We have the following properties. First, we have $y = \frac{\xi}{\delta^{\ell+1}} - \xi - 1 \geq \frac{\xi}{\delta^{\ell}} > 400$ (since $\frac{1}{\delta^{\ell+1}} - \frac{1}{\delta^{\ell}} - 1 = \frac{1-\delta-\delta^{\ell+1}}{\delta^{\ell+1}} > 1 > \frac{1}{\xi}$ as $2\delta^{\ell+1} + \delta < 1$ by $\ell \geq 3$ and $\delta \leq \frac{1}{8}$, and since $\xi \geq 53$). Second, $y + 2 < y + \xi + 1 \leq \frac{\xi}{\delta^6} \leq \frac{1}{\delta^9}$ since $\ell \leq 5$ and $\xi \leq \frac{1}{\delta^3}$ and due to integrality, $y + 3 \leq \frac{1}{\delta^9}$. Third, we also have (using the properties of ξ) that $(1+\delta)^y \geq (1+\delta)^{\frac{\xi}{\delta^\ell}} \geq \left(\frac{1}{\delta^\ell}\right)^{\frac{1}{\delta^\ell}} \geq \left(\frac{1}{\delta^3}\right)^{\frac{1}{\delta^3}} > \frac{1}{\delta^{1500}}$, and $(1+\delta)^y \leq (1+\delta)^{\xi/\delta^{\ell+1}} \leq \left(\frac{1}{\delta^{\ell+1}}\right)^{\frac{1}{\delta^{\ell+1}}} \leq \left(\frac{1}{\delta}\right)^{6/\delta^6} \leq \left(\frac{1}{\delta}\right)^{1/\delta^7 - 12}$, and last, we conclude that $y + 1 \leq \delta^2(1+\delta)^y$ since $y + 1 \leq \frac{1}{\delta^9}$ while $(1+\delta)^y \geq \frac{1}{\delta^{1500}}$.

Since the density of every job is in $[1, (1+\delta)^y]$, every job $j \in I$ has $p_j = (1+\delta)^k$, $w_j = (1+\delta)^{k'}$, for some integers k, k' such that $0 \leq k' - k \leq y$. Let $\gamma = \frac{\delta^{12}}{(1+\delta)^y}$. We have $\gamma \geq \delta^{1/\delta^7}$ and $\gamma \leq \delta^{1512}$. Let $\mathcal{I}$ denote the set of values i such that there is at least one job of size $(1+\delta)^i$. Clearly, $|\mathcal{I}| \leq n$. Let $n_{r,i}$ denote the number of jobs of size $(1+\delta)^i$ and density $(1+\delta)^r$. By scaling the machine speeds (that are integer powers of $1+\delta$) and possibly reordering the machines, let the speeds of machines be $s_1 \geq s_2 \cdots \geq s_m$, where without loss of generality, $s_1 = 1$, and for $i \geq 2$, $s_i = (1+\delta)^{k_i}$, for some non-positive integer $k_i \leq 0$. It is possible that there is more than one machine of speed 1, and we consider all such machines in the next paragraph.

First guessing step and its analysis. Recall that the work of a machine is the total size of jobs assigned to it. We would like to assume that job sizes are scaled such that the work of the most loaded machine (the most loaded machine in terms of work without taking speeds into account) of speed 1 is in $[1/(1+\delta), 1)$ (while since we have jobs of varying densities, the work of slower machines may be arbitrary). We will now show that this is possible. For a job j, and $1 \leq b \leq \frac{n}{\delta}$, let $D_{j,b}$ be the interval $[(1+\delta)^{b-1}p_j, (1+\delta)^b p_j)$ (we have that $(1+\delta)^{\frac{n}{\delta}} \geq n+1$ holds by a direct calculation since $\frac{n}{\delta}$ is integral).

Without loss of generality we consider solutions where at least one machine of speed 1 has at least one job. This holds since otherwise it is possible to move an arbitrary job to such a machine and get a solution of a smaller cost.

▶ **Lemma 6.** *For any solution, there exist a job $1 \leq j' \leq n$ and an integer $1 \leq b' \leq \frac{n}{\delta}$, such that the work of the most loaded machine out of works of machines of speed 1 is in $D_{j,b'}$.*

For every choice of a pair j, b ($\frac{n^2}{\delta}$ choices in total), we scale (i.e., divide) job sizes by $(1+\delta)^b p_j$ (which is an integer power of $1+\delta$), and apply the algorithm described later in this section (this algorithm enforces the existence of a fastest machine with a suitable load, where it is possible that there is no such solution for some choices). By the claim, given an optimal solution OPT, for at least one choice of j, b the assumption regarding the work of the most loaded machine of speed 1 (that the work is at least $(1+\delta)^{-1}$ and below 1) holds as a result of the scaling. We will pick the best solution, whose objective function value cannot be larger than that of the solution obtained for the correct choice of j, b. In what follows we analyze the properties of the correct choice of j, b for a given optimal solution OPT after the scaling. Moreover, the job sizes are according to the scaling.

Machine configurations – preliminaries. Let U be a threshold on job sizes separating jobs seen as relatively large and jobs seen as relatively small. Let $\hat{I}$ be the subset of jobs assigned to machine i in some solution. The U-cost of machine i in this solution is defined as the total weighted completion time of jobs whose sizes are at least U, plus the Γ-values of jobs whose sizes are below U. The cost for machine i is therefore its U-cost plus $\frac{1}{2s_i}\sum_{j\in\hat{I},p_j<U} w_j\cdot p_j$. Obviously, the U-cost never exceeds the cost, for any value of U. Additionally, similarly to the cost, the U-cost for a fixed value of U is monotone in the sense that removing a job from the machine decreases its U-cost, and thus, if we consider a block of jobs of a common density each of which shorter than U, decreasing the total size of jobs in a block also decreases the U-cost.

We use the following two functions of δ: $f(\delta)$ and $g(\delta)$, both integral, non-decreasing, and *negative*. A machine i is called *slow* if its speed is at most $(1+\delta)^{f(\delta)}$ (and otherwise we say that it is not slow or that it is fast). We say that a machine is lightly loaded if its work does not exceed $(1+\delta)^{g(\delta)}$ (and otherwise it is heavily loaded). Let

$$f(\delta)=g(\delta)-\frac{2}{\delta^2} \text{ and } g(\delta)=-(\frac{1}{\delta})^{\frac{1}{\delta^{300}}}.$$

Consider a fixed optimal solution OPT (for the input considered in this section which consists of a subset of jobs, and given the scaling of machine speeds and job sizes).

▶ **Lemma 7.** *No machine has a load strictly above $\frac{2}{\delta}$ in OPT, and this is an upper bound on the work of any machine as well. Furthermore, in OPT every slow machine is lightly loaded.*

Formal definition of machine configuration. Next, we define machine configurations. A configuration is a vector that defines most properties for the schedule of one machine. This consists of a set of jobs assigned to it in terms of the types of jobs, that is, a job is specified by its size and weight but not by its identity, and jobs that are relatively small are defined by their approximate total sizes and densities. The set of all configurations will be denoted by $\mathcal{C}$. For a configuration $C\in\mathcal{C}$, the first component is an integer $j_1(C)\in\mathbb{Z}$, such that the total work of the machine is in $((1+\delta)^{j_1(C)-2},(1+\delta)^{j_1(C)}]$. The second component is a non-positive integer $j_2(C)$ such that the speed of the machine is $(1+\delta)^{j_2(C)}$. The third component is an integer $j_3(C)\in\mathbb{Z}$, such that $j_3(C)=j_1(C)-j_2(C)$, and therefore the completion time (or load) of the machine is in $((1+\delta)^{j_3(C)-2},(1+\delta)^{j_3(C)}]$.

Recall that $\gamma=\frac{\delta^{12}}{(1+\delta)^y}$. For integers r,i such that $0\le r\le y$, $i\le j_1(C)$, and $\gamma(1+\delta)^{j_1(C)-1}\le(1+\delta)^i\le(1+\delta)^{j_1(C)}$ (i.e., $j_1(C)-1+\lceil\log_{1+\delta}\gamma\rceil\le i\le j_1(C)$), there is an integer component $n_{r,i}(C)\ge 0$ stating how many jobs of size $(1+\delta)^i$ and density $(1+\delta)^r$ are assigned to this machine. These jobs are called *large* (large for configuration C, or large for a machine that has configuration C). We let $n_{r,i}(C)=0$ for other (smaller or larger) values of i (these are constants that are not a part of the configuration). There are additional components for other jobs assigned to a machine whose configuration is C. These jobs (which are not taken into account in the components of the form $n_{r,i}(C)$) must have smaller values of i, as jobs with larger values of i (jobs of sizes above $(1+\delta)^{j_1(C)}$) cannot be assigned to a machine that has configuration C. This is so since they are too large, given the upper bound on the work of the machine. These remaining jobs are called *small* jobs for C, or small jobs for a machine whose schedule is according to configuration C. For every r, there is an integer component $t_r(C)\ge 0$ such that the total size of small jobs (of sizes in $(0,\gamma(1+\delta)^{j_1(C)-1})$) of density $(1+\delta)^r$ assigned to a machine with configuration C is in $((t_r(C)-1)\cdot\gamma(1+\delta)^{j_1(C)-1},t_r(C)\cdot\gamma(1+\delta)^{j_1(C)-1}]$. We have $t_r(C)=0$ if and only if there are no such jobs for a machine with this configuration.

Valid configurations. Recall that $\mathcal{I}$ is the set of indices i such that there is at least one job of size $(1+\delta)^i$. A configuration C is valid if the total size of jobs is sufficiently close to its required work, and its load does not exceed $\frac{2}{\delta}$. Formally, letting $\mathcal{I}(C) = \{i \in \mathcal{I} : j_1(C) - 1 + \lceil \log_{1+\delta} \gamma \rceil \leq i \leq j_1(C)\}$ it is required that $\sum_{0 \leq r \leq y, i \in \mathcal{I}(C)} (1+\delta)^i \cdot n_{r,i}(C) + \sum_{r=0}^{y} t_r(C) \cdot \gamma(1+\delta)^{j_1(C)-1} \in \left((1+\delta)^{j_1(C)-2}, (1+\delta)^{j_1(C)+1}\right]$. The reason for the increased exponent of the right endpoint is that for every density the total size of small jobs may be smaller by an additive term of $\gamma(1+\delta)^{j_1(C)-1}$ compared to its upper bound, and since $(y+1) \cdot \gamma < \delta$, it suffices to increase the right endpoint by a multiplicative factor of $1+\delta$. We also let $\mathcal{I}'(C) = \{i \in \mathcal{I} : i \leq j_1(C) - 2 + \lceil \log_{1+\delta} \gamma \rceil\}$, and require $(1+\delta)^{j_3(C)-2} < \frac{2}{\delta}$. In the remainder of this section, given C, we will use the threshold $U(C) = \gamma(1+\delta)^{j_1(C)-1}$ for computing the U-cost of a machine with configuration C (this is the value U of the configuration C).

Bounding the number of configurations. The number of components is therefore at most $3+(y+1)\cdot(\lceil \log_{1+\delta} \frac{1}{\gamma} \rceil + 2) + (y+1) \leq 3 + (y+1)(\log_{1+\delta} \frac{1}{\gamma} + 3)$. Since $\log_{1+\delta} \frac{1}{\gamma} = y + \log_{1+\delta} \frac{1}{\delta^{12}} \leq y + \frac{1}{\delta^{13}}$, and $y + 1 \leq \frac{1}{\delta^9}$, we have $3 + (y+1)(\log_{1+\delta} \frac{1}{\gamma} + 3) \leq 3 + \frac{1}{\delta^9}(\frac{1}{\delta^9} + \frac{1}{\delta^{13}} + 3) < \frac{1}{\delta^{23}}$. We will use the following bounds.

▶ **Lemma 8.** *For every pair of value j_1 and j_2, there are at most $(\frac{1}{\delta})^{1/\delta^{33}}$ configurations C with $j_1(C) = j_1$ and $j_2(C) = j_2$.*

▶ **Lemma 9.** *The number of options for j_1, j_2, j_3 (for which there are valid configurations) is $O(\frac{n^2 m}{\delta})$*

Let $J = \{\sigma_1, \sigma_2, \ldots, \sigma_\kappa\}$ be the set of all κ different speeds of machines (where $1 \leq \kappa \leq m$) such that $1 = \sigma_1 > \sigma_2 > \cdots > \sigma_\kappa$, and let $N_i \geq 1$ be the number of machines of speed σ_i.

An auxiliary mathematical program. Consider the following mathematical program $\mathcal{P}$. The goal of $\mathcal{P}$ is to determine a partial schedule via machine configurations. For every configuration C, X_C is a decision variable stating how many machines have this configuration. Obviously, the number of used configurations whose second component is σ_i cannot exceed N_i (it is not required to use all machines, but it is required to assign all jobs). For every triple of density, size, and a configuration (density $(1+\delta)^r$, size $(1+\delta)^i$, and configuration C), there is a decision variable $Y_{r,i,C}$ corresponding to the number of jobs of this size and density that are assigned to machines whose configurations are C as small jobs for this configuration. The variable $Y_{r,i,C}$ may be positive only if a job of size $(1+\delta)^i$ is small for configuration C, i.e., $i \leq j_1(C) - 2 + \lceil \log_{1+\delta} \gamma \rceil$, and in all other cases we set $Y_{r,i,C} = 0$ to be a constant rather than a variable. Each such variable is used for all copies of C together since jobs assigned as small jobs within the copies of the configuration may be of different sizes. Based on these decision variables we define a set of constraints and solve the resulting mixed-integer linear program $\mathcal{P}$.

A configuration C has a cost denoted by $cost(C)$ associated with it, which is the U-cost for the threshold $U(C) = \gamma(1+\delta)^{j_1(C)-1}$. Recall that for the purpose of calculating the U-cost of a machine, a list of its large jobs is needed. However, for its jobs of size below U (i.e., small jobs), the exact list of such small jobs is not needed. The only needed property of the subset of small jobs (for configuration C) of each one of the densities is their total size. This allows us to assign very different jobs that are small to machines with the same configuration, as long as these jobs are indeed small for the configuration and have the same density.

For large jobs, the exact identities of jobs are not needed as well, but a list of densities and sizes is needed. Thus, the U-cost for configuration C is calculated assuming that the machine has exactly $n_{r,i}(C)$ jobs of size $(1+\delta)^i$ and weight $(1+\delta)^{i+r}$ (i.e., density $(1+\delta)^r$) assigned to it, and the total size of small jobs (of sizes in $(0, \gamma(1+\delta)^{j_1(C)-1})$) with densities equal to $(1+\delta)^r$ is exactly $t_r(C)\cdot\gamma(1+\delta)^{j_1(C)-1}$. The objective of $\mathcal{P}$ is to minimize the sum of costs of configurations plus the missing parts of the costs of jobs that are assigned as small. For that, for a job of size $(1+\delta)^i$ and density $(1+\delta)^r$ that is assigned to a machine of speed s as a small job for its configuration, an additive term of $\frac{(1+\delta)^{r+2i}}{2s}$ is incurred (this is the difference between the actual cost for this job, and its Γ-value, which is the part of the cost already included in the U-cost). This last term only depends on the speed of the machine that runs the job rather than the specific machine. Thus, for each r and i we add

$$(1+\delta)^{r+2i}\cdot\sum_{C\in\mathcal{C}}\frac{Y_{r,i,C}}{2(1+\delta)^{j_2(C)}}$$

to the cost of configurations to get the total cost of the schedule, where $\sum_{C\in\mathcal{C}}Y_{r,i,C}$ is the number of jobs of size $(1+\delta)^i$ and density $(1+\delta)^r$ that are assigned as small jobs for their configurations.

$$\min\sum_{C\in\mathcal{C}}cost(C)\cdot X_C+\sum_{r=0}^{y}\sum_{i\in\mathcal{I}}(1+\delta)^{r+2i}\cdot\sum_{C\in\mathcal{C}}\frac{Y_{r,i,C}}{2(1+\delta)^{j_2(C)}}\qquad s.t.$$

$$\sum_{\boldsymbol{C\in\mathcal{C}:(1+\delta)^{j_2(C)}=\sigma_i}}\boldsymbol{X_C}\le\boldsymbol{N_i}\qquad\forall\sigma_i\in J\tag{1}$$

$$\sum_{\boldsymbol{C\in\mathcal{C}}}\boldsymbol{n_{r,i}(C)\cdot X_C}+\sum_{\boldsymbol{C\in\mathcal{C}}}\boldsymbol{Y_{r,i,C}}=\boldsymbol{n_{r,i}}\qquad\forall 0\le r\le y, i\in\mathcal{I}\tag{2}$$

$$\sum_{\boldsymbol{i\in\mathcal{I}'(C)}}\boldsymbol{(1+\delta)^i\cdot Y_{r,i,C}}-\boldsymbol{\left((t_r(C)+1)\cdot\gamma\cdot(1+\delta)^{j_1(C)-1}\right)\cdot X_C}\le 0\tag{3}$$

$$\forall 0\le r\le y, C\in\mathcal{C}$$

$$\sum_{\boldsymbol{C\in\mathcal{C}:j_1(C)\in\{-1,0,1\},j_2(C)=0}}\boldsymbol{X_C}\ge\boldsymbol{1}\tag{4}$$

$$\boldsymbol{Y_{r,i,C}}\ge\boldsymbol{0}\qquad\forall C\in\mathcal{C},0\le r\le y,i\in\mathcal{I}'(C)\tag{5}$$

$$\boldsymbol{X_C}\ge\boldsymbol{0}\qquad\forall C\in\mathcal{C}\tag{6}$$

Motivation for the constraints. Later, we will use the last mathematical program to direct our algorithm. Before we continue we explain the intuition behind its constraints. We stress that the formal proof does not use this intuition, so it is given to assist the reader. Condition (1) ensures that the number of used machines for each speed does not exceed the existing number of such machines. Condition (2) states that every job is assigned (either it is a large job or a small job). Condition (3) considers jobs of density $(1+\delta)^r$ that are assigned as small to machines scheduled according to configuration C, and verifies that sufficient space is allocated for them if the space for them is slightly extended (this extended space may be used as we show later). Condition (4) ensures that indeed there is a machine of the maximum speed that has a work that is close to 1, that is, above $\frac{1}{(1+\delta)^2}$ and at most $1+\delta$ (the condition that the work is in $[1/(1+\delta),1)$ is slightly relaxed).

Observe that we distinguish between large and small jobs for a machine based on the configuration and not solely by its speed. This feature is needed since our problem does not allow the use of the dual-approximation method (because the objective is not based on machine loads).

Formal definition of the subset of integer variables. We define the set of heavy configurations $\mathcal{C}_H$ as $\mathcal{C}_H = \{C \in \mathcal{C} : j_1(C) > g(\delta)\}$, (note that the definition of a heavy configuration or a heavily loaded machine depends only on the work, and it is independent of the speed) and the complement set of light configurations is $\mathcal{C}_L = \mathcal{C} \setminus \mathcal{C}_H$. By Lemma 7 it is sufficient to consider light configurations for slow machines. Therefore, we delete heavy configurations for slow machines and do not consider them (which is justified later in the comparison between the cost of an optimal schedule and the optimal solution of $\mathcal{P}$).

We see $\mathcal{P}$ as a mixed-integer linear program and thus we define next the set of variables that are forced to be integral and the set of variables that are allowed to be fractional. All *variables* $Y_{r,i,C}$ may be fractional. The variables of (light) configurations corresponding to slow machines and variables of light configurations of fast machines may be fractional, whereas the variables of heavy configurations corresponding to fast machines must be integral. The number of variables is polynomial, and the number of integral variables will be constant (a function of δ).

Recall that for every pair of speed and work (i.e, a pair j_1, j_2), the number of different valid configurations is constant (as a function of δ). The number of different speeds of fast machines is at most $|f(\delta)| = -f(\delta)$. The number of different values of $j_1(C)$ such that C is a heavy configuration is at most $-g(\delta) + \lceil \log_{1+\delta} \frac{2}{\delta} \rceil \leq \frac{2}{\delta^2} - g(\delta) = -f(\delta)$, and $j_2(C)$ of a heavy configuration C must be a fast speed (so there are at most $-f(\delta)$ values for it). As the number of integral variables is constant (as a function of δ), an optimal solution can be found in polynomial time [30, 26] that is a function of δ multiplied by a polynomial in the input encoding length.

Presenting a feasible solution to $\mathcal{P}$ whose cost is at most the cost of an optimal schedule. We will first compare the cost of the optimal schedule OPT to the cost of an optimal solution of $\mathcal{P}$. Later, we will show how to obtain an actual schedule given a solution of $\mathcal{P}$, such that the cost of the schedule is larger only by a factor of at most $1 + \delta$ than the objective function value of the solution to $\mathcal{P}$. Let (X^*, Y^*) denote an optimal solution to the mixed-integer linear program, and let Z^* be its objective value. Let Z^{OPT} denote the cost of OPT.

▶ **Theorem 10.** *If (X^*, Y^*) is an optimal solution of $\mathcal{P}$, then $Z^* \leq Z^{OPT}$.*

Using the solution (X^*, Y^*) for finding an approximate schedule. Our last rounding steps are designed in order to use (X^*, Y^*) for getting an approximate schedule for the rounded ratio problem. These steps are the constructive methods for establishing our scheme.

We start with constructing an alternative set of values of the variables and bounding the resulting cost from above. We will later convert this alternative solution (that may violate some of the constraints of $\mathcal{P}$) into a schedule. For every $C \in \mathcal{C}$, let $X'_C = \lfloor X^*_C \rfloor$. If $X^*_C = 0$, then we set $X'_C = 0$. In this case $Y^*_{r,i,C} = 0$ must hold for all r, i by (3), and we set $Y'_{r,i,C} = 0$. If $X^*_C > 0$, we consider two cases. If $C \in C_H$, then the variables of configurations are integral, and therefore $X'_C = X^*_C$. In this case we set $Y'_{r,i,C} = \lceil Y^*_{r,i,C} \rceil$ for any $0 \leq r \leq y$, $i \in \mathcal{I}'(C)$. Otherwise, that is, in the case $C \in C_L$, let $Y'_{r,i,C} = \left\lfloor \frac{X'_C}{X^*_C} \cdot Y^*_{r,i,C} \right\rfloor$.

Using these definitions, in the case $Y'_{r,i,C} > 0$, if $C \in C_H$, then we have $Y'_{r,i,C} \leq Y^*_{r,i,C} + 1$, and otherwise we have $Y'_{r,i,C} \leq \frac{X'_C}{X^*_C} \cdot Y^*_{r,i,C}$. Moreover, if $C \in C_L$ and $X^*_C > 0$, then we have $Y'_{r,i,C} > \frac{X'_C}{X^*_C} \cdot Y^*_{r,i,C} - 1 > Y^*_{r,i,c} - \frac{Y^*_{r,i,C}}{X^*_C} - 1$, by $X^*_C < X'_C + 1$ (that holds even if $X'_C = 0$).

Bounding the cost of (X', Y'). We will define a schedule and find an upper bound on its objective function value. Before that, we analyze the first increase in the cost which is due to the transformation from (X^*, Y^*) to (X', Y'), and show that it is by an additive factor of at most $\delta^{100} \cdot Z^*$. The solution (X', Y') that we consider is not a feasible solution for $\mathcal{P}$, as (for example) constraint (2) does not necessarily hold as it is possible that $\sum_{C \in \mathcal{C}} n_{r,i}(C) X'_C + \sum_{C \in \mathcal{C}} \cdot Y'_{r,i,C} \neq n_{r,i}$, but we can still bound its objective function value using the objective function value of the optimal (and feasible) solution.

Since $X'_C \leq X^*_C$ for $C \in \mathcal{C}$, and $Y'_{r,i,C} \leq Y^*_{r,i,C}$ for all $C \in C_L$, $0 \leq r \leq y$, and $i \in \mathcal{I}'(C)$, the objective function value for these variables is at most Z^* plus

$$\begin{aligned}
&\textstyle\sum_{r=0}^{y} \sum_{i \in \mathcal{I}} (1+\delta)^{r+2i} \sum_{C \in C_H : i \in \mathcal{I}'(C)} \frac{Y'_{r,i,C} - Y^*_{r,i,C}}{2(1+\delta)^{j_2(C)}} \\
&\leq \textstyle\sum_{r=0}^{y} \sum_{C \in C_H} \sum_{i \in \mathcal{I}'(C)} (1+\delta)^{r+2i} \frac{X^*_C}{2(1+\delta)^{j_2(C)}} \ ,
\end{aligned}$$

since for $C \in C_H$ it holds that $Y'_{r,i,C} = 0$ if $X^*_C = 0$, and otherwise $Y'_{r,i,C} - Y^*_{r,i,C} \leq 1 \leq X^*_C$. Next, we show that

$$\sum_{r=0}^{y} \sum_{i \in \mathcal{I}'(C)} (1+\delta)^{r+2i} \frac{1}{2(1+\delta)^{j_2(C)}} \leq \delta^{100} \cdot cost(C)$$

for any $C \in C_H$, and thus we will conclude that the objective function value of the set of variables X'_C, $Y'_{r,i,C}$ is at most $(1+\delta^{100}) \cdot Z^*$. We have $cost(C) \geq \frac{(1+\delta)^{2j_1(C)-4}}{2(1+\delta)^{j_2(C)}}$ (by Lemma 1, since $cost(C)$ is computed for jobs of total size above $(1+\delta)^{j_1(C)-2}$, and the jobs densities are no smaller than 1). Let $i' = \max \mathcal{I}'(C)$. We have

$$(1+\delta)^{2i'} \leq (\gamma(1+\delta)^{j_1(C)-1})^2$$

and

$$\sum_{i \leq i'} (1+\delta)^{2i} < (1+\delta)^{2i'+1}/\delta \ .$$

Additionally, $\sum_{r=0}^{y} (1+\delta)^r < (1+\delta)^{y+1}/\delta$. For a given $C \in C_H$, we get

$$\begin{aligned}
&\textstyle\sum_{r=0}^{y} \sum_{i \in \mathcal{I}'(C)} (1+\delta)^{r+2i} \frac{1}{2(1+\delta)^{j_2(C)}} \leq \frac{(1+\delta)^{y+2i'+2}}{2\delta^2(1+\delta)^{j_2(C)}} \leq \frac{(1+\delta)^{y+2}(\gamma(1+\delta)^{j_1(C)-1})^2}{2\delta^2(1+\delta)^{j_2(C)}} \\
&= \frac{(1+\delta)^{y+2} \frac{\delta^{24}}{(1+\delta)^{2y}} (1+\delta)^{2j_1(C)-2}}{2\delta^2(1+\delta)^{j_2(C)}} \leq cost(C) \cdot \delta^{22}(1+\delta)^{4-y} \leq \delta^{100} \cdot cost(C)
\end{aligned}$$

(since $(1+\delta)^y \geq \frac{1}{\delta^{1500}}$). Thus the claim regarding the cost of (X', Y') holds.

Defining a schedule of most jobs while leaving a small subset of unassigned jobs. We will define a schedule that is based on the rounded variables. The schedule will require additional space for jobs that are small for their configurations (in some configurations) both due to rounding of variables, and the actual assignment of jobs to machines which does not exist in $\mathcal{P}$ (for such jobs). Due to the rounding of numbers of copies of configurations, some jobs that are large for their configurations will also remain unassigned. Such jobs will obviously have

to be assigned, thus increasing the cost. For the resulting schedule, the objective function value of $\mathcal{P}$ may be no longer correct, though we show that the true objective is not much larger.

Next, we find additional upper bounds that are based on the rounded variables (only those that were rounded up). We prove upper bounds on the space used for jobs that are small for their configurations where a possible increase is due to the rounding of variables only. Additional space to be used for such jobs will be defined and analyzed later, and here we only analyze the space that was already defined.

Using condition (3), for every $C \in C_L$ such that $X^*_C > 0$ and $0 \leq r \leq y$, we have

$$\sum_{i \in \mathcal{I}'(C)} (1+\delta)^i \cdot Y'_{r,i,C} \leq \sum_{i \in \mathcal{I}'(C)} (1+\delta)^i \cdot \frac{X'_C}{X^*_C} \cdot Y^*_{r,i,C}$$

$$\leq \frac{X'_C}{X^*_C} \cdot (t_r(C)+1) \cdot \gamma \cdot (1+\delta)^{j_1(C)-1} \cdot X^*_C = (t_r(C)+1) \cdot \gamma \cdot (1+\delta)^{j_1(C)-1} \cdot X'_C \ .$$

Similarly, for every $C \in C_H$ such that $X^*_C > 0$ and $0 \leq r \leq y$, we have

$$\begin{aligned} &\textstyle\sum_{i \in \mathcal{I}'(C)} (1+\delta)^i \cdot Y'_{r,i,C} \leq \sum_{i \in \mathcal{I}'(C)} (1+\delta)^i \cdot (Y^*_{r,i,C}+1) \\ &\leq (t_r(C)+1) \cdot \gamma \cdot (1+\delta)^{j_1(C)-1} \cdot X^*_C + \textstyle\sum_{i \in \mathcal{I}'(C)} (1+\delta)^i \\ &\qquad \leq (t_r(C)+2+\tfrac{2}{\delta}) \cdot \gamma \cdot (1+\delta)^{j_1(C)-1} \cdot X'_C \end{aligned}$$

as $X'_C = X^*_C \geq 1$, and

$$\sum_{i \in \mathcal{I}'(C)} (1+\delta)^i \leq \gamma (1+\delta)^{j_1(C)-1} \sum_{i=0}^{\infty} \frac{1}{(1+\delta)^i} = \gamma (1+\delta)^{j_1(C)-1} \cdot \frac{1+\delta}{\delta} \ .$$

If $X^*_C = 0$, then the corresponding upper bound (the first one if $C \in C_L$, and the second one if $C \in C_H$) hold trivially, as all variables are equal to zero.

We let $n'_{r,i} = n_{r,i} - \sum_{C \in \mathcal{C}} n_{r,i}(C) X'_C - \sum_{C \in \mathcal{C}} Y'_{r,i,C}$. If $n'_{r,i} > 0$, then we say that $n'_{r,i}$ is the number of unassigned jobs of size $(1+\delta)^i$ and density $(1+\delta)^r$. These jobs will be called *unassigned jobs*, as they will remain unassigned if we assign jobs exactly using configurations that are based on the modified set of variables (including the assignment to configurations of jobs that are small for their configurations). We explain this assignment, which is called the first assignment step, before we proceed.

The first assignment step will be to assign jobs according to the configurations for which $X'_C > 0$, such that there will be X'_C machines whose sets of large jobs will be as required (based on the definition of C). We will then assign $Y'_{r,i,C}$ jobs of size $(1+\delta)^i$ and density $(1+\delta)^r$ to the machines whose configuration is C (these are small jobs for C). In order to ensure that all jobs can be scheduled, for each machine that is assigned the configuration C, additionally to total size of at most $t_r(C) \cdot \gamma \cdot (1+\delta)^{j_1(C)-1}$ jobs of density $(1+\delta)^r$ that are small jobs for configuration C that the machine can contain, such jobs of total size at most $\frac{3\gamma}{\delta} \cdot (1+\delta)^{j_1(C)-1}$ are allocated to this machine. These jobs are called *additional jobs*, and we will calculate the increase in the total cost as a result. The unassigned jobs will all be assigned to a machine of speed 1 whose configuration C' has $j_1(C') = 0$ or $j_1(C') = -1$. We will compute an upper bound on the total size of these unassigned jobs that will allow us to find an upper bound on the increase in the cost.

Consider the machines whose configuration is C. For every $0 \leq r \leq y$, create X'_C *bins* of size $t_r(C) \cdot \gamma \cdot (1+\delta)^{j_1(C)-1}$ each (these bins are called bins of the first kind), and X'_C bins of size $\frac{3\gamma}{\delta} \cdot (1+\delta)^{j_1(C)-1}$ (these bins are called bins of the second kind); if $t_r(C) = 0$, then

we introduce only the second kind of bins. The additional jobs are those that are packed into bins of the second kind. We define the allocation of jobs to machines by packing them as items into these bins. If the number of items of a certain type assigned using this packing process exceeds the existing number of jobs, then some of the spaces allocated in this process for items will not receive jobs. For every $i \in \mathcal{I}'(C)$, pack $Y'_{r,i,C}$ *items* of size $(1+\delta)^i$ into these bins using First Fit.

We show that all items are packed. Assume by contradiction that this is not the case. Since the size of each item is below $\gamma(1+\delta)^{j_1(C)-1}$ and there is an item that cannot be packed, each bin of the first kind (if such a bin exists) is occupied by at least $(t_r(C)-1)\cdot\gamma\cdot(1+\delta)^{j_1(C)-1}$, and each bin of the second kind is occupied by at least $(\frac{3}{\delta}-1)\cdot\gamma\cdot(1+\delta)^{j_1(C)-1}$. We find that the total size of items of density $(1+\delta)^r$ that are to be packed into these bins as small, which is equal to $\sum_{i\in\mathcal{I}'(C)}(1+\delta)^i Y'_{r,i,C}$ is above $(t_r(C)+\frac{3}{\delta}-2)\cdot\gamma\cdot(1+\delta)^{j_1(C)-1}\cdot X'_C$, contradicting the upper bound that we proved on $\sum_{i\in\mathcal{I}'(C)}(1+\delta)^i\cdot Y'_{r,i,C}$, since $\frac{3}{\delta}-2 > 2+\frac{2}{\delta}$ (since $\delta \le \frac{1}{8}$).

Bounding the increase in the cost of a machine due to the first assignment step. The increase in the cost of each machine (since the total size of small jobs of density $(1+\delta)^r$ for each $0 \le r \le y$ may become larger) can be upper bounded as follows. A possible schedule, i.e., a permutation of the job set of the machine, is obtained by assigning the items of the bins of the first kind as a block for each value of r of size at most $t_r(C)\cdot\gamma(1+\delta)^{j_1(C)-1}$, and the jobs of the bins of the second kind as a block of the last jobs assigned to the machine (this is possibly not an optimal ordering). For a job j that is small for its configuration the difference between its cost and Γ_j is already included in the objective function value (since it is assigned as a small job), and thus we compute the total Γ-values of the added blocks. Instead of considering these blocks (for different values of r) separately, we see it as one block assuming that all densities are equal to $(1+\delta)^y$ (this assumption cannot reduce the cost). We assume without loss of generality that $j_2(C)=0$. Let

$$W = \sum_{0\le r\le y, i\in\mathcal{I}(C)} (1+\delta)^i\cdot n_{r,i}(C) + \sum_{r=0}^{y} t_r(C)\cdot(\gamma\cdot(1+\delta)^{j_1(C)-1})$$

be the work that was used for calculating $cost(C)$ in $\mathcal{P}$. The total size of this united block of small jobs added to this machine is at most $\frac{3(y+1)\gamma}{\delta}(1+\delta)^{j_1(C)-1}$, and thus, the sum of Γ-values for all these blocks is at most

$$(1+\delta)^y\cdot\left(W+\frac{3\cdot(y+1)\cdot\gamma(1+\delta)^{j_1(C)-1}}{2\delta}\right)\cdot\frac{3(y+1)\gamma(1+\delta)^{j_1(C)-1}}{\delta}$$

by Lemma 3. We use $W \ge (1+\delta)^{j_1(C)-2}$, $\delta \le 1/8$, $y+1 \le \frac{1}{\delta^9}$, $\gamma \le \delta^{1512}$, and $(1+\delta)^y\cdot\gamma = \delta^{12}$, and get

$$\frac{3(y+1)\gamma(1+\delta)^{j_1(C)-1}}{2\delta} \le \delta^{1501}\cdot W\ ,$$

and $(1+\delta)^y\cdot\frac{3(y+1)\gamma(1+\delta)^{j_1(C)-1}}{\delta} \le 3(1+\delta)\delta^2\cdot W$ and get that the total Γ-values of the added blocks is at most

$$(1+\delta^{1501})W\cdot 3(1+\delta)\cdot\delta^2\cdot W \le 7\delta^2\cdot W^2/2\ .$$

By Lemma 1, we have $cost(C) \ge W^2/2$ as all densities are at least 1. Thus, the second increase in the cost is by an additive factor of at most $7\delta^2\cdot Z^*$.

Bounding the cost of scheduling the unassigned jobs. Recall that $n'_{r,i}$ is the number of unassigned jobs of size $(1+\delta)^i$ and density $(1+\delta)^r$, and for $C \in C_H$, $X'_C = X^*_C$ and $Y'_{r,i,C} \geq Y^*_{r,i,C}$. Using (2), we have

$$\begin{aligned} n'_{r,i} &= n_{r,i} - \textstyle\sum_{C\in C_H} n_{r,i}(C)\cdot X'_C - \sum_{C\in C_H : i\in\mathcal{I}'(C)} Y'_{r,i,C} \\ &\quad - \textstyle\sum_{C\in C_L} n_{r,i}(C)\cdot X'_C - \sum_{C\in C_L : i\in\mathcal{I}'(C)} Y'_{r,i,C} \\ &\leq n_{r,i} - \textstyle\sum_{C\in C_H} n_{r,i}(C)\cdot X^*_C - \sum_{C\in C_H : i\in\mathcal{I}'(C)} Y^*_{r,i,C} \\ &\quad - \textstyle\sum_{C\in C_L}(n_{r,i}(C)\cdot(X^*_C - 1)) - \sum_{C\in C_L : i\in\mathcal{I}'(C), X^*_C>0}(Y^*_{r,i,C} - \frac{Y^*_{r,i,C}}{X^*_C} - 1) \\ &= \textstyle\sum_{C\in C_L} n_{r,i}(C) + \sum_{C\in C_L : i\in\mathcal{I}'(C), X^*_C>0}(\frac{Y^*_{r,i,C}}{X^*_C} + 1)\ . \end{aligned}$$

The total size of unassigned jobs is at most

$$\begin{aligned} &\textstyle\sum_{r=0}^{y}\sum_{C\in C_L}\sum_{i\in\mathcal{I}(C)}(1+\delta)^i\cdot n_{r,i}(C) \\ &+\textstyle\sum_{r=0}^{y}\sum_{C\in C_L : X^*_C>0}\sum_{i\in\mathcal{I}'(C)}(1+\delta)^i\cdot(\frac{Y^*_{r,i,C}}{X^*_C}+1)\ . \end{aligned}$$

Using

$$\sum_{i\in\mathcal{I}'(C)}(1+\delta)^i\cdot\frac{Y^*_{r,i,C}}{X^*_C} \leq (t_r(C)+1)\cdot\gamma\cdot(1+\delta)^{j_1(C)-1}$$

for $C \in C_L$ such that $X^*_C > 0$ (which holds by constraint (3) divided by X^*_C), the total size of unassigned jobs is at most

$$\begin{aligned} &\textstyle\sum_{r=0}^{y}\sum_{C\in C_L}\Big((t_r(C)+1)\gamma(1+\delta)^{j_1(C)-1} + \sum_{i\in\mathcal{I}(C)}(1+\delta)^i n_{r,i}(C)\Big) \\ &\qquad + \textstyle\sum_{r=0}^{y}\sum_{C\in C_L}\sum_{i\in\mathcal{I}'(C)}(1+\delta)^i\ . \end{aligned}$$

Using

$$\sum_{0\leq r\leq y, i\in\mathcal{I}(C)}(1+\delta)^i n_{r,i}(C) + \sum_{r=0}^{y} t_r(C)(\gamma(1+\delta)^{j_1(C)}) \leq (1+\delta)^{j_1(C)+1}$$

(by the definition of a configuration) and $\sum_{i\in\mathcal{I}'(C)}(1+\delta)^i \leq \gamma(1+\delta)^{j_1(C)}/\delta$ for any $C \in \mathcal{C}$, the total size of unassigned jobs is at most

$$\begin{aligned} &\textstyle\sum_{C\in C_L}(1+\delta)^{j_1(C)+1} + \sum_{r=0}^{y}\sum_{C\in C_L}\gamma(1+\delta)^{j_1(C)-1} \\ &+\textstyle\sum_{r=0}^{y}\sum_{C\in C_L}\gamma(1+\delta)^{j_1(C)}/\delta < \sum_{C\in C_L} 2(1+\delta)^{j_1(C)}\ . \end{aligned}$$

since $(1+\delta+(y+1)\gamma(1/\delta+1)) \leq 1+\delta+\frac{1}{\delta^9}\cdot\delta^{1500}\cdot\frac{2}{\delta} < 2$.

The proof of the following lemma is derived using the definition of $g(\delta)$.

▶ **Lemma 11.** *The total size of unassigned jobs is at most* 2γ.

Now we show that this leads to the required result. The jobs are assigned to a machine with a configuration C' such that $j_1(C') \in \{1, 0, -1\}$. Let x be the completion time of this machine before modifications, where $x \in (\frac{1}{(1+\delta)^3}, (1+\delta)^2]$. The modifications described so far during the first assignment step may increase the completion time of the machine by at most

$$\frac{3}{\delta}\cdot\gamma(y+1) < \delta^{1400}$$

(since $3\gamma(y+1)/\delta \leq \frac{3}{\delta} \cdot \delta^{1500}\frac{1}{\delta^9} < \delta^{1400}$). Even if the density of each such job is $(1+\delta)^y$, and the unassigned jobs are now assigned as a block on the most loaded machine of speed 1 starting at time $x+\delta^{1400}$, the sum of their weighted completion times will be at most

$$2\gamma(1+\delta)^y \cdot (x+\delta^{1400}+2\gamma) < 2\delta^{12}(x+\delta^{1399}) \ ,$$

while the value of $cost(C')$ is at least $x^2/2$. It holds that

$$\frac{2\delta^{12}(x+\delta^{1399})}{x^2/2} = \frac{4\delta^{12}}{x} + \frac{4\delta^{1411}}{x^2} \leq 4\delta^{12}(1+\delta)^3 + 4\delta^{1411}(1+\delta)^6 < 6\delta^{12} \ ,$$

so the increase in the cost is at most by an additive factor of $15\delta^{12} \cdot Z^*$. The total cost will be at most $Z^*(1+\delta^{100}+7\delta^2+6\delta^{12}) < (1+\delta) \cdot Z^*$, and thus the following theorem holds.

▶ **Theorem 12.** *There exists a schedule whose cost is at most $(1+\delta) \cdot Z^*$, and such a schedule can be constructed in polynomial time from (X^*, Y^*).*

Summary of the algorithm for the bounded ratio problem. In summary, in this section we provided an $(1+\delta)$-approximation algorithm for the bounded ratio scheduling problem. The algorithm tests all possible intervals $D_{j,b}$, and for each one it constructs the MILP $\mathcal{P}$, and finds an optimal solution. The best solution is converted into a schedule. We conclude the paper with the statement of our main result.

▶ **Theorem 13.** *There is an EPTAS for the problem of minimizing the total weighted completion time on uniformly related machines.*

References

1. Foto N. Afrati, Evripidis Bampis, Chandra Chekuri, David R. Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Clifford Stein, and Maxim Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proc. 40th Symp. Foundations of Computer Science (FOCS)*, pages 32–44, 1999. doi:10.1109/SFFCS.1999.814574.
2. Foto N. Afrati and Ioannis Milis. Designing PTASs for MIN-SUM scheduling problems. *Discrete Applied Mathematics*, 154(4):622–639, 2006. doi:10.1016/J.DAM.2005.05.014.
3. N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling. In *Proc. 8th Symp. on Discrete Algorithms (SODA)*, pages 493–500, 1997.
4. N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.
5. Alok Baveja, Xiaoran Qu, and Aravind Srinivasan. Approximating weighted completion time via stronger negative correlation. *Journal of Scheduling*, 27(4):319–328, 2024. doi:10.1007/S10951-023-00780-Y.
6. Sebastian Berndt, Hauke Brinkop, Klaus Jansen, Matthias Mnich, and Tobias Stamm. New support size bounds for integer programming, applied to makespan minimization on uniformly related machines. In *Proc. 34th International Symp. on Algorithms and Computation (ISAAC)*, 2023. doi:10.4230/LIPIcs.ISAAC.2023.13.
7. M. Cesati and L. Trevisan. On the efficiency of polynomial time approximation schemes. *Information Processing Letters*, 64(4):165–171, 1997. doi:10.1016/S0020-0190(97)00164-6.
8. Chandra Chekuri and Sanjeev Khanna. A PTAS for minimizing weighted completion time on uniformly related machines. In *Proc. 28th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 848–861, 2001. doi:10.1007/3-540-48224-5_69.
9. R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer-Verlag, Berlin, 1999.

10 L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. *Algorithmica*, 39(1):43–57, 2004. doi:10.1007/S00453-003-1077-7.

11 Leah Epstein and Asaf Levin. An efficient polynomial time approximation scheme for load balancing on uniformly related machines. *Mathematical Programming*, 147(1-2):1–23, 2014. doi:10.1007/S10107-013-0706-4.

12 Leah Epstein and Asaf Levin. Minimum total weighted completion time: Faster approximation schemes. *CoRR*, abs/1404.1059, 2014. arXiv:1404.1059.

13 Leah Epstein, Asaf Levin, Alan J. Soper, and Vitaly A. Strusevich. Power of preemption for minimizing total completion time on uniform parallel machines. *SIAM Journal on Discrete Mathematics*, 31(1):101–123, 2017. doi:10.1137/16M1066610.

14 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, Berlin, 2006.

15 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

16 Stanisław Gawiejnowicz. A review of four decades of time-dependent scheduling: Main results, new topics, and open problems. *Journal of Scheduling*, 23(1):3–47, 2020. doi:10.1007/S10951-019-00630-W.

17 M. X. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proc. 8th Symp. on Discrete Algorithms (SODA)*, pages 591–598, 1997.

18 D. S. Hochbaum. Various notions of approximations: Good, better, best, and more. In D. S. Hochbaum, editor, *Approximation algorithms*. PWS Publishing Company, 1997.

19 D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, 1985. doi:10.1145/2455.214106.

20 D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987. doi:10.1145/7531.7535.

21 D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988. doi:10.1137/0217033.

22 W. A. Horn. Minimizing average flow time with parallel machines. *Operations Research*, 21(3):846–847, 1973.

23 K. Jansen. An EPTAS for scheduling jobs on uniform processors: Using an MILP relaxation with a constant number of integral variables. *SIAM Journal on Discrete Mathematics*, 24(2):457–485, 2010. doi:10.1137/090749451.

24 K. Jansen and C. Robenek. Scheduling jobs on identical and uniform processors revisited. In *Proc. of the 9th International Workshop on Approximation and Online Algorithms (WAOA)*, pages 109–122, 2011.

25 Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. *Mathematics of Operations Research*, 45(4):1371–1392, 2020. doi:10.1287/MOOR.2019.1036.

26 Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proc. 15th Symp. Theory of Computing (STOC)*, pages 193–206, 1983. doi:10.1145/800061.808749.

27 Dušan Knop and Martin Kouteckỳ. Scheduling meets n-fold integer programming. *Journal of Scheduling*, 21:493–503, 2018. doi:10.1007/S10951-017-0550-0.

28 Ishai Kones and Asaf Levin. A unified framework for designing eptas for load balancing on parallel machines. *Algorithmica*, 81(7):3025–3046, 2019. doi:10.1007/S00453-019-00566-9.

29 Hanane Krim, Nicolas Zufferey, Jean-Yves Potvin, Rachid Benmansour, and David Duvivier. Tabu search for a parallel-machine scheduling problem with periodic maintenance, job rejection and weighted sum of completion times. *Journal of Scheduling*, pages 1–17, 2022.

30 H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983. doi:10.1287/MOOR.8.4.538.

31 D. Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008. doi:10.1093/COMJNL/BXM048.

32 Sartaj K. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23:116–127, 1976. doi:10.1145/321921.321934.

33 Martin Skutella and Gerhard J. Woeginger. A PTAS for minimizing the total weighted completion time on identical parallel machines. *Mathematics of Operations Research*, 25(1):63–75, 2000. doi:10.1287/MOOR.25.1.63.15212.

34 W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

35 Vipin Ravindran Vijayalakshmi, Marc Schröder, and Tami Tamir. Minimizing total completion time with machine-dependent priority lists. *European Journal of Operational Research*, 315(3):844–854, 2024. doi:10.1016/J.EJOR.2023.12.030.

Lower Bounds for Several Standard Bin Packing Algorithms in the Random Order Model

Leah Epstein
Department of Mathematics, Faculty of Natural Sciences, University of Haifa, Israel

Asaf Levin
Faculty of Data and Decisions Science, The Technion, Haifa, Israel

Abstract

We consider the performance of standard bin packing algorithms in the random order model. We provide an improved lower bound of 1.15582656 on the asymptotic approximation ratio of Best Fit (BF) for randomly ordered inputs. We also show lower bounds on the asymptotic approximation ratio for two bounded space bin packing algorithms in this model, namely for 2-BF and 2-FF. These are well-studied bounded space algorithms and the first one has the same asymptotic worst-case performance as BF. However, the resulting lower bounds on their performances in the random order model are much higher than that of BF.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Bin packing, Best Fit, Random order, Bounded space algorithms

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.26

Funding *Asaf Levin*: Partially supported by ISF – Israel Science Foundation grant number 1467/22.

1 Introduction

Bin packing (BP) or standard bin packing [31, 30, 36, 6, 7] is a combinatorial optimization problem introduced in the early 1970's. In this problem, a set of items of positive rational sizes of at most 1 is to be partitioned into subsets of total sizes not exceeding 1. The subsets are called bins, the process of assigning an item to a bin is called packing, and the goal is to minimize the number of (non-empty) bins. The variant where the input is a set of items is called (standard or classic) offline BP. In the online variant of standard BP, items are presented one by one, such that each item has to be packed into a bin before the next item is presented. Many natural algorithms for offline BP process items as a list, and can be seen as online algorithms.

Since standard BP is NP-hard (and so are many other versions of BP), it is frequently studied with respect to approximation. The approximation ratio of an algorithm for a minimization problem and an input is the ratio between the costs of the algorithm and of an optimal solution for this input. The absolute approximate ratio of an algorithm is the worst-case approximation ratio over valid inputs. The asymptotic approximation ratio, which is the standard measure for bin packing type problems, is the superior limit of the sequence of approximation ratios with fixed optimal costs (when these fixed costs grow without bound). For online algorithms, the approximation ratio is also called *competitive ratio*, but the definitions are identical. Since the studied algorithms are of interest both as offline algorithms and as online algorithms, we use the terms *asymptotic approximation ratio* and *absolute approximation ratio*.

Here, we consider online bin packing algorithms with respect to randomly ordered inputs and the asymptotic approximation ratio, and we briefly comment on the absolute approximation ratio of one algorithm. For worst-case analysis, the best possible asymptotic approximation ratio of an online algorithm is in the interval $[1.54278, 1.57829]$ [6, 7] and the

© Leah Epstein and Asaf Levin;
licensed under Creative Commons License CC-BY 4.0

best absolute approximation ratio is $\frac{5}{3}$ [8]. Bin packing was also analyzed with respect to average case analysis for various input distributions including $U[0,1]$ [17, 9, 37, 32, 5], where these models are different from our model, which is defined below.

The algorithms that we analyze. The Best Fit (BF) algorithm [31, 20, 38] is defined as follows. It starts with an empty set of active bins. Whenever a new item is revealed, it computes the subset of active bins in which the new item can be added without exceeding the bound of 1 on the total size of items including the new item. If this subset is non-empty, the algorithm picks a bin of maximum total size of items from the subset and packs the item there. If the subset is empty, it opens a new (empty) bin that is defined as an active bin and it packs the item there. The asymptotic (and absolute) asymptotic approximation ratio of BF is 1.7 [31, 20].

In addition to BF, we analyze two well-studied bounded space algorithms, namely 2-BF and 2-FF. Bounded space algorithms also have an active set of bins, but their number is bounded from above by a constant integer parameter. Thus, a bounded space algorithm may close a bin in order to open a new active bin, in which case the closed bin cannot be used again. Next Fit (NF) [30] is a greedy bounded space algorithm that has at most one active bin. If it has no active bin, NF opens a new bin which becomes active, and the item is packed into that bin. If NF has an active bin and the item can be packed into that bin with respect to its size, this is done. Otherwise, if there is an active bin, it is closed, and a new bin is opened, such that it receives the new item and becomes active.

Bounded space algorithms with larger numbers of active bins were studied as well [36, 43, 41, 40, 18, 44, 42]. The algorithm 2-BF has at most two active bins at all times, and initially it has no active bins. When a new item is presented, it is packed into an active bin with the maximum already packed total size (where it fits). If this is not possible and the number of active bins is 2, the most loaded active bin is closed (and it will not be able to receive new items), and (no matter whether a bin was closed or not) a new active bin is opened for the new item. A related algorithm is called 2-FF, and the difference is that in the case where there are two active bins such that each one of them can receive the new item, the bin that was opened first is used (and not the fullest bin). The latter algorithm can be seen as the bounded space version of First Fit (FF) [31], which packs every item into the bin of minimum index where it can be added. The two algorithms were analyzed with respect to the worst-case asymptotic approximation ratio. The asymptotic approximation ratio of 2-BF is 1.7 [19], as for BF (and FF) [31, 20], and the asymptotic approximation ratio of 2-FF is 2 (as for NF) [40, 44].

Randomly ordered inputs. The random order model was introduced for comparison and for contrast with worst-case analysis. In this model, the expected cost or profit over all permutations for an input is compared to the optimal offline cost or profit. For bin packing, this means that all $n!$ permutations of a set of n items are considered with a uniform distribution over the permutations (which are input sequences). For an offline optimal solution, there is no meaning to the permutation, since the input of an offline bin packing algorithm is a set of items rather than a sequence. Such inputs were studied for various problems such as graph coloring [10, 25] and scheduling [2, 1], and other problems [33, 39, 14, 35, 27, 4, 28, 26]. Here, we study it with respect to bin packing [34, 16, 3, 5, 29]. We follow previous work and use a stochastic process for presenting lower bounds on the performance guarantee in the random order model. This stochastic process is a Markov chain with a finite number of states whose transition probabilities are provided by the family of permutations of a common

set of items. This Markov chain allows us to compute the equilibrium probabilities of the states using standard approaches, and these equilibrium probabilities allow us to find the resulting lower bound in a straightforward computation. Thus, the crux is to define the input set of items and the corresponding set of states and transition probabilities of the Markov chain. Note that a related model where algorithms are compared directly based on inputs with their random permutations and not with comparison to an optimal offline solution was studied for several problems including bin packing [11, 21, 15, 13, 12].

Kenyon [34] started the study of BF with respect to randomly ordered inputs. She proved a lower bound of approximately 1.08 on the asymptotic approximation ratio of BF with randomly ordered inputs, and wrote that 1.15 may be a lower bound. Many years later, Albers, Khan, and Ladewig [3] improved the lower bound to 1.1037, and finally Hebbar, Khan, and Sreenivas [29] improved the lower bound to 1.144. Kenyon [34] also proved the surprising result that the asymptotic approximation ratio of BF in this model is at most 1.5. By using various ingredients, this result was improved to slightly below 1.5 [29]. For the special case of items of sizes in $(\frac{1}{4}, \frac{1}{2}]$, an upper bound of approximately 1.4941 on the asymptotic approximation ratio of BF for randomly ordered inputs was shown [5].

Randomly ordered inputs with items larger than 1/3 (inputs with large items) were studied by Albers, Khan, and Ladewig [3], who showed that the asymptotic approximation for this case is at most 1.25. It was proved afterwards [5] that the asymptotic approximation ratio for this case is in fact 1. Interestingly, the absolute ratio for general inputs and even for inputs with large items is not 1. Lower bounds of 1.3 and 1.2, respectively, were shown [3] for the absolute approximation ratio for general inputs and for items of sizes above $\frac{1}{3}$, respectively (where for the last case an upper bound of $\frac{21}{16} = 1.3125$ on the absolute approximation ratio of BF was proved in the same work).

Coffman et al. analyzed NF for BP with random order [16], and showed that the asymptotic competitive ratio is unchanged (as opposed to the situation with BF [34]). The main idea for the lower bound proof is to use the worst-case input of NF that alternates between items of size $\frac{1}{2}$ and very small items of size $\varepsilon > 0$. It is possible to add multiple items of size ε instead of just one between every two items of size $\frac{1}{2}$, and applying NF on almost every permutation of this input will result in bins with one item of size $\frac{1}{2}$ and many items of size ε, and a very small number of bins with two items of size $\frac{1}{2}$.

Bin packing has different variants. A dual problem to BP is bin covering (BC) (see for example [13, 22, 23]), which is a maximization problem. BC can also be offline or online. The input is the same as for bin packing, but subsets should be covered in the sense that total sizes of bins that count towards the objective should be at least 1. Bin covering and other related problems were studied with respect to inputs permuted in a random order [13, 22, 23, 24]. Fischer [24] presented algorithms of asymptotic approximation ratios that are 1 or close to 1 using definitions of the analysis of the type that we perform here, though those are not natural algorithms, and they are randomized.

Our results. We provide lower bounds on the asymptotic approximation ratio for three online algorithms for standard bin packing. Two of these algorithms are the bounded space algorithms 2-BF and 2-FF, and the third algorithm is BF, which was already studied with respect to random orders. We provide an improved lower bound for BF, and much higher lower bounds for algorithms whose performance is the same as BF (2-BF) and not much worse than BF (2-FF) in terms of worst-case analysis. Our lower bound for Best Fit is 1.15582656, while the previous bound was 1.1440 [29]. Our lower bound for 2-BF is $\frac{4}{3}$, and our lower bound for 2-FF is above 1.41. The last two bounds for the bounded space

algorithms are based on common input sets of items. We start our study by considering this type of input and the resulting bounds for 2-BF and 2-FF in Section 2. Then, we turn our attention to BF, and present our lower bound in Section 3. Our lower bound for the asymptotic approximation ratio of BF is based on Markov chains on finite sets of item sizes (and so are the other lower bounds). The cardinalities of the sets of item sizes are (positive) integer parameters of our construction. Previous lower bounds for BF were also based on the approach of Markov chains [34, 3, 29] (as described above), but there was a single (constant cardinality) set of items for each proof (with at least two and at most seven item sizes) rather than a class of sets of sizes. Our lower bound for BF holds even for items of sizes in $(\frac{1}{4}, \frac{1}{2}]$, which is the special case that was studied by Ayyadevara et al. [5] with respect to an upper bound for BF, while the current best lower bound for this special case is the one of Kenyon [34]. In Section 4, we prove a new lower bound on the absolute approximation ratio of BF for randomly ordered inputs, improving the lower bound of 1.3 [3] to 1.5. A forthcoming full version will contain additional details and omitted proofs.

2 Lower Bounds for Two Bounded Space Algorithms

We start with defining the common new input for the lower bounds of 2-BF and 2-FF and common characteristics of the two Markov chains. Then we will show how to construct a chain for proving a lower bound on the asymptotic approximation ratio for this input with random order for 2-BF. The resulting proof for 2-BF is a complete analytical analysis of the state probabilities at equilibrium. Then, we turn our attention to the analysis of a corresponding lower bound for 2-FF. This last bound is computed using a computer assisted program to solve a (large) system of linear equations. We augment the last computer assisted proof with a very close bound derived using an analytic (and complete) proof.

2.1 The Common Input and Common Aspects of the Two Markov Chains

We define an input for these algorithms. There will be large, medium, and small items. Let $n > 0$ be a large integer, and let $N = 2kn$, where $k \geq 1$ is an integral parameter. The total number of items will be N^N. Let $\varepsilon < \frac{1}{N^{N+3}}$. A small item has size $N^2 \cdot \varepsilon$ and the probability for such an item is $\frac{N^{N-1}-1}{N^{N-1}}$. The expected number of items that are not small is N. The remaining probabilities (for medium and large items) will be equal, and there are $2k$ item sizes, where there are k sizes of large items and k sizes of medium items. For $1 \leq i \leq k$, an i-large item has size $\frac{1}{2} + i \cdot \varepsilon$, and these items (for all values of i) are called large. For $1 \leq i \leq k$, an i-medium item has size $\frac{1}{2} - i \cdot \varepsilon$, and these items (for all values of i) are called medium. A feasible solution (that provides an upper bound on the optimal cost) assigns all small items to a single bin. It also assigns every i-medium item with a i-large item, whenever this is possible. The remaining items are packed into bins alone, and the resulting number of bins is approximately $k \cdot n + o(n)$. See [29] for details regarding the property that this is indeed the optimal cost for inputs with a constant number of input sizes. The articles [3, 29] also show precisely that using an input drawn from a distribution on a discrete set of item sizes also gives a lower bound on the asymptotic approximation ratio of randomly ordered items.

Given an integer parameter $k \geq 1$, the Markov chains we define in what follows consist of $2k+1$ states. There is an initial state denoted by A, k states denoted by $B_1, B_2, \ldots, B_k$ and k states denoted by $C_1, C_2, \ldots, C_k$. For each state, the transition probabilities are integer multiples of $\frac{1}{2k}$ such that the sum over all transitions out of each state is 1 for the state.

We define the transitions of each state where we write these values multiplied by $2k$, then we write the probability of each state explicitly and show that they are consistent with the transitions. Since every transition corresponds to the arrival of one item out of specific types in the constructions, we discuss even transitions for which the state is unchanged.

Our Markov chains will have a well-defined equilibrium. We use the notation $p(A)$, $p(B_i)$ (for $1 \leq i \leq k$), and $p(C_i)$ (for $1 \leq i \leq k$) for the equilibrium probabilities of the states. We also use $p(B) = \sum_{i=1}^{k} p(B_i)$ and $p(C) = \sum_{i=1}^{k} p(C_i)$.

2.2 Analysis for 2-BF

We first define the transition probabilities of the Markov chain that we use. Next, we will use those transition probabilities to obtain the corresponding equilibrium probabilities. Last, we show that this Markov chain allows us to prove our lower bound of the performance guarantee of 2-BF in the random order model.

The transition probabilities. These probabilities are given as follows.

- For A, there is one transition to each other state with value 1 (that is, with probability $\frac{1}{2k}$). The value of the transition to A is zero.
- For B_i (where $1 \leq i \leq k$) the value for the transition to A is $i + k$. The value for the transition back to B_i is $k - i$ (so for $i = k$ it is zero), and for B_j such that $j \neq i$ the value of the transition to B_j is zero. The values of all remaining transitions (to states of the form C_ℓ) are also equal to zero.
- For C_i (where $1 \leq i \leq k$) the value for the transition to A is $k - i + 1$. The corresponding value for B_j is 1 if $j < i$, and otherwise it is zero. The corresponding value for C_j is 1 if $j < i$, it is $k - i + 1$ if $j = i$, and it is zero if $j > i$.

Observe that this chain has a unique equilibrium (i.e., the equilibrium the chain tends to when the number of transitions grows without bound does not depend on the initial state).

The equilibrium probabilities. We show that the following probabilities satisfy the requirements, and since there is a unique solution, they give this solution. We define the probabilities based on $p(A)$ and ensure that the sum of probabilities is 1.

▶ **Lemma 1.** *Let* $\lambda_k = \sum_{j=1}^{k} \frac{1}{(k+j)^2}$. *We have the three following claims.*

1. *It holds that*

$$p(A) = p(C) = \frac{1}{2 + 2k\lambda_k} \ .$$

2. *For* $1 \leq i \leq k$, *it holds that*

$$p(C_i) = \frac{2k \cdot p(A)}{k + i - 1} - \frac{2k \cdot p(A)}{k + i} = \frac{2k \cdot p(A)}{(k + i - 1) \cdot (k + i)} \ .$$

3. *For* $1 \leq i \leq k$, *it holds that*

$$p(B_i) = \frac{2k \cdot p(A)}{(k + i)^2} > 0 \ .$$

Observe that for large values of k it holds that $\lambda_k \approx \int_k^{2k} \frac{1}{x^2} dx$, and the value of this definite integral is $\frac{1}{2k}$. Thus, $p(A) = \frac{1}{2+2k\lambda_k}$ is approximately equal to $\frac{1}{3}$ when k grows without bound.

The usage of the Markov chain for the analysis of 2-BF. Our goal is to prove the following theorem.

▶ **Theorem 2.** *The asymptotic approximation ratio of 2-BF on randomly permuted inputs is at least* $\frac{4}{3} \approx 1.33333$.

Proof. The small items are sufficiently small such that all of them would fit into a bin together with one medium item or one large item. However, once a small item is packed with an item that is not small, since $N^2\varepsilon > 2k\varepsilon$, the bin cannot contain another item that is not small. We use the property of the packing that after a bin receives one medium or large item and at least one small item, it will only receive small items. On the other hand, by the same argument, a bin that contains two items that are not small (two medium items or a medium item and a large item) cannot contain any small items. Thus, such a bin cannot contain any additional items, and with a slight abuse of notation we assume that it is closed when it receives the second item.

With high probability it holds that there is at least one small item between every two items that are not small, and there is at least one small item in the beginning of the input. We assume the process of arrival of items and packing them by 2-BF has the following form. Consider the prefix of the input before and including the first item that is not small. This bin has a total size above $\frac{1}{2} + k\varepsilon$ after the prefix arrives. At this time, there is one active bin, and this bin can only receive small items. From this time on, there will be either one active bin or two active bins. There will always be an active bin that can only receive small items, and if there is a second active bin, it will have one item, where this item is not small. State A will correspond to having a single active bin. Transitions will be defined for times that an item that is not small arrives (after the prefix).

The other $2k$ states are defined as follows. For each such state there is one bin that already has one item that is not small and at least one small item, so it can only receive small items, and one bin that contains a single item, where this item is medium or large. State B_i (for $1 \leq i \leq k$) is the state where the second bin has an i-medium item, and state C_i (for $1 \leq i \leq k$) is the state where the second bin has an i-large item.

If the stochastic process is currently in state B_i and a j-medium item arrives, the new item is packed into the bin that has no small items, and there is a transition to state A. If the process is currently in state B_i and a j-large item arrives, in the case $j \leq i$, the new item is packed into the bin that has no small items, and there is a transition to state A. In the case $j > i$, the bin with small items (which has the largest total size) is closed and the new item is packed into a new bin. One of the two bins will receive at least one small item (with high probability) before the next medium or large item arrives and this is the bin with the large item. Thus, in this case the transition is to the same state (B_i).

If the process is currently in state C_i and a j-large item arrives, the bin with small items (which has the largest total size) is closed and the new item is packed into a new bin. One of the two bins will receive at least one small item (with high probability) before the next medium or large item arrives and this is the bin with the larger large item. This is the new bin if $j \geq i$, and otherwise it is the bin with the i-large item. The state remains C_i if $j \geq i$, and otherwise the transition is to the state C_j. If the process is currently in state C_i and a j-medium item arrives, the transition is to state A if $j \geq i$. Otherwise, the bin with at least one small item is closed, and the new item is packed into a new bin. The larger item packed alone in a bin is the large item, and therefore in the case $j < i$ the new state is B_j.

The resulting chain is thus the one analyzed earlier in Lemma 1. The expected cost of 2-BF is $1 - p(A)$ times the number of items that are not small. The expected cost of an optimal solution is $\frac{1}{2}$ times the number of items (that are not small). For k growing without bound we find a lower bound of $\frac{4}{3}$. ◀

2.3 Analysis for 2-FF

Next, we discuss the difference with 2-FF. Since the closing rule is unchanged, the algorithm will close the bin that has a small item for this algorithm as well. The states are therefore unchanged, but the bin which will start receiving small items out of the two bins that have one item each (where these items are not small) is the older one. There is no change in the transitions to state A, but in cases where the transition is to another state, it is always to C_j when the new item is j-large, and to B_j when the new item is j-medium. These changes are reflected in the proof of the following theorem.

▶ **Theorem 3.** *The asymptotic approximation ratio of 2-FF on randomly permuted inputs is at least* 1.41318399*.*

The bound in the last theorem is proved using a computer assisted solution of a system of linear equations. In the full version of this work, we provide an analytic proof showing a slightly weaker bound of $\frac{8355970560}{5918778241} \approx 1.41177$.

Proof. We again use a Markov chain with $2k+1$ states. The transition probabilities of this chain are given as follows.

- For A, there is one transition to each other state with probability $\frac{1}{2k}$. The probability of the transition to A is zero.
- For B_i (where $1 \leq i \leq k$) the probability for the transition to A is $\frac{i+k}{2k}$. The probability for the transition to C_j for $j > i$ is $\frac{1}{2k}$. All other transition probabilities are equal to zero.
- For C_i (where $1 \leq i \leq k$) the probability for the transition to A is $\frac{k-i+1}{2k}$. The probability for the transition to C_j (for any j) is $\frac{1}{2k}$. The probability for the transition to B_j is $\frac{1}{2k}$ if $j < i$ and otherwise it is zero (thus the probability for the transition to B_k is always zero).

We solved this chain numerically for $k = 10000000$ and found that $p(A) \approx 0.29340800445$. This shows a lower bound of 1.41318399 for 2-FF with random orders. ◀

3 An Improved Lower Bound for Best Fit

In this section, we turn our attention to Best Fit in the random order model. We consider a class of inputs parameterized by an integer parameter $k \geq 1$. Item sizes are slightly smaller than $\frac{1}{3}$ (k such sizes) and slightly larger than $\frac{1}{3}$ (k such sizes), and there is also a size of exactly $\frac{1}{2}$ (so there are $2k+1$ different sizes). Let $\varepsilon > 0$ be a small constant such that $\varepsilon < \frac{1}{100k}$. Let $Z = \frac{1}{2}$. For $i = 1, 2, \ldots, k$, let

$$U_i = \frac{1}{3} - \frac{\sum_{j=i}^{k} \varepsilon^{k-j+1}}{2} \text{ and } V_i = \frac{1}{3} + \sum_{j=i}^{k} \varepsilon^{k-j+1}.$$

It holds that the sequence U_i is monotonically increasing as a function of i and the sequence V_i is monotonically decreasing as a function of i.

We say that a bin is *ready* if it cannot receive an additional item. Since the smallest item size is U_1, this means that its total size of items is above $1 - U_1$. Note that $U_1 > \frac{1}{3} - k\varepsilon > \frac{1}{4}$, and therefore no bin can contain more than three items. On the other hand, no item is larger than $\frac{1}{2}$ since $V_1 \leq \frac{1}{3} + k\varepsilon < \frac{1}{2}$. A *packing pattern* is a non-empty set of three item sizes out of the $2k+1$ item sizes which can be packed into a bin. A multiset of items that cannot be packed into a bin is called an *invalid pattern* (and a pattern is sometimes called a valid pattern). For example, it is not possible to pack two items of size U_k and an item

of size V_{k-1} into a bin since $2 \cdot U_k + V_{k-1} = 1 - \varepsilon + (\varepsilon + \varepsilon^2) > 1$ (so this is an invalid pattern), but it is possible to pack two items of size U_k and an item of size V_k into a bin since $2 \cdot U_k + V_k = 1 - \varepsilon + \varepsilon = 1$ (so this is a pattern or a valid pattern).

▶ **Lemma 4.** *The following packing patterns are a complete list of packing patterns.*
- *Any pattern consisting of a single item ($2k + 1$ patterns in total).*
- *Any pattern consisting of two items (less than $(2k + 1)^2$ patterns in total).*
- *Any pattern consisting of three items of sizes below $\frac{1}{3}$.*
- *Any pattern consisting of three items which are U_{j_1}, U_{j_2}, and V_{j_3} such that $j_3 \geq \max\{j_1, j_2\}$.*

▶ **Corollary 5.** *Any bin with one of the following packing patterns is ready.*
Any valid pattern consisting of three items.
Any pattern consisting of two items of sizes above $\frac{1}{3}$.
Any pattern with an item of size U_{j_1} and an item of size V_{j_2} such that $j_2 < j_1$.

Proof. For patterns with three items the claim holds since there are no patterns with four items. For patterns with two items the claim holds by Lemma 4 since there is no valid pattern containing such a pair of items and an additional item. ◀

Thus, bins that are not ready are those with a single item, with two items smaller than $\frac{1}{3}$, and with a pair of items of sizes U_{j_1}, V_{j_2} such that $j_2 \geq j_1$. The following lemma establishes that although BF is not a bounded space algorithm, when we restrict our attention to this class of inputs, BF is in fact bounded space.

▶ **Lemma 6.** *BF never encounters a situation where it has two bins with the same pattern that are not ready. Moreover, it never has two bins with a single item each, it has at most one bin with two items smaller than $\frac{1}{3}$, and if it has such a bin, then there is no bin with a single item of size below $\frac{1}{3}$.*

Proof. The claim holds for bins with a single item since any two items can be packed into a bin together and BF does not open a new bin when it can pack an item into an existing bin.

Assume that there are two bins with one or two items smaller than $\frac{1}{3}$ packed into each of these bins. When the second one of these bins was opened, its first item could have been packed into the first one of these two bins, while BF does not open a new bin when it can pack an item into an existing bin.

Finally, consider two bins, each with a pair of items of sizes U_{j_1}, V_{j_2} such that $j_2 \geq j_1$. When the second such bin is opened, the first such bin already has two items, since any two items can be packed together into a bin. When the item of size U_{j_1} of the second bin arrives, its bin has at most one item. Thus, the first bin has a total size $U_{j_1} + V_{j_2}$ while the second bin has at most V_{j_2}. Since a pattern with one item of size V_{j_2} and two items of size U_{j_1} is valid, the item of size U_{j_1} should have preferred to be packed into the first bin by BF. ◀

We use the following names for bins that are not ready.
- A bin with a single item is called OH if it has an item of size Z, it is called OS_i if it has an item of size U_i and it is called OL_i if it has an item of size V_i. Recall that there is at most one such bin (out of all $2k + 1$ types for such a bin) at every time.
- A bin with two items of sizes U_{j_1} and U_{j_2} where $j_2 \geq j_1$ is called TS_{j_2}. Note that we only remember the size of the larger item of the two, and recall that there is at most one such bin (out of all k types for such a bin) at every time.
- A bin with items of sizes U_{j_1} and V_{j_2} where $j_2 \geq j_1$ is called SL_{j_2}. Note that we only remember the size of the larger item of the two.

To summarize we recall the number of items in a bin as well as the size of its largest item.

▶ **Lemma 7.** *BF never encounters a situation where it has the following pairs of bins that are not ready.*

- *An SL_{j_1} bin and a TS_{j_2} bin such that $j_2 \leq j_1$.*
- *An SL_{j_1} bin and an OS_{j_2} bin such that $j_2 \leq j_1$.*
- *A TS_{j_1} bin and an OL_{j_2} bin such that $j_2 \geq j_1$.*

Proof. We assume by contradiction that one of the stated pairs of bins exists, and show that none of the two orders of creation (by BF) for the two bins is possible. Since every pair of items can be packed into a bin together, the first bin of the two already has two items when the second one receives its first item. This excludes the option that an SL_{j_1} bin receives its first item though an OS_{j_2} bin exists for $j_2 \leq j_1$, and that a TS_{j_1} bin receives its first item while an OL_{j_2} bin exists for $j_2 \geq j_1$. We are left with four cases where a new bin is opened when the other bin already has two items. Moreover, we use the property that no item is larger than $\frac{1}{2}$ while every item is larger than $\frac{1}{4}$, so the total size of two items is always larger than that of one item.

Consider an SL_{j_1} bin and a TS_{j_2} bin such that $j_2 \leq j_1$ that were created in this order. Neither the first item nor the second item of the TS_{j_2} could be packed into the SL_{j_1} bin (since two items are larger than one item, BF would have preferred the SL_{j_1} bin). Since the SL_{j_1} bin is not ready, it has an item of size V_{j_1}, and an item of size at most U_{j_1}. The TS_{j_2} bin has an item at most $U_{j_2} \leq U_{j_1}$ (in fact it has two such items). This contradicts the fact that no item of the TS_{j_2} could be packed into the SL_{j_1} bin. Similarly, consider the case of an SL_{j_1} bin and an OS_{j_2} bin such that $j_2 \leq j_1$, that were created in this order. A contradiction is reached in the same way (the second bin has just one item of size U_{j_2} but this is sufficient for the contradiction.

Consider a TS_{j_2} bin and an SL_{j_1} bin that were created in this order. The smaller item of the SL_{j_1} bin could have been packed into the TS_{j_2} bin (no matter what the values j_1, j_2 are). Consider a TS_{j_1} bin and an OL_{j_2} bin such that $j_2 \geq j_1$ that were created in this order. The total size of two items of size at most U_{j_1} each and one item of size $V_{j_2} \leq V_{j_1}$ can be packed together. ◀

We have seen that there can be at most one TS_i bin that is not ready (for some i). We extend this claim for SL_{j_1} bins that are not ready (for some j_1).

▶ **Lemma 8.** *BF never encounters a situation where there are two SL_{j_1} bins that are not ready.*

Proof. Let U_{j_2}, U_{j_3} be the sizes of items packed with items of sizes V_{j_1} into these bins. Since the bins are not ready, it holds that $j_1 \geq j_2, j_3$. Assume that the item of size U_{j_3} is packed into the bin that was created later by BF. Since $U_{j_2}, U_{j_3} \leq U_{j_1}$, the item of size U_{j_3} could be packed into the first SL_{j_1} bin, and BF would have packed it there. ◀

Note that we did not exclude the option of a pair of bins that are an SL_{j_1} bin and a SL_{j_2} bin (for $j_1 \neq j_2$) that are not ready, and this is in fact possible. For example, if an item of size V_1 was packed with an item of size U_1 and then an item of size V_2 arrives followed by an item of size U_2, BF will have one SL_1 bin and one SL_2 bin, none of which is ready.

To define a suitable Markov chain for the analysis of BF on our family of instances, we discuss its states and transition probabilities. Every state has a subset of bins that is not ready. The initial state has an empty set of such bins. Since there are k options for SL_i bins where each state may have at most one bin for every index, every state has a subset of

indexes of $\{1, 2, \ldots, k\}$ of such bins. We discuss the set of states as a function of the larger index i such that the state has an SL_i bin (and also states without such bins for which we let $i = 0$).

For a fixed value of i and any set of values for $j < i$ (2^{i-1} possible subsets of SL_j bins for $j < i$) the following states exist (where the selected subset is called *the specific subset*), such that they are partitioned into seven types:

1. In addition to the specific subset there are no other bins that are not ready.
2. In addition to the specific subset there is an OH bin and no other bins.
3. In addition to the specific subset there is an OL_ℓ bin and no other bins. Such a state exists for any $1 \leq \ell \leq k$.
4. In addition to the specific subset there is an OS_ℓ bin and no other bins. Such a state exists for any $i + 1 \leq \ell \leq k$.
5. In addition to the specific subset there is a TS_ℓ bin and no other bins. Such a state exists for any $i + 1 \leq \ell \leq k$.
6. In addition to the specific subset and a TS_ℓ for a fixed ℓ, there is also an OH bin. Such a state exists for any $i + 1 \leq \ell \leq k$.
7. In addition to the specific subset and a TS_ℓ for a fixed ℓ, there is also an OL_q bin. Such a state exists for any $i + 1 \leq \ell \leq k$ and $1 \leq q \leq \ell - 1$.

This set of states includes all possible states based on the properties proved above. The next lemma enumerates the possible states.

▶ **Lemma 9.** *The set of states has exactly* $2^k \cdot (2k + 3) - 1$ *states.*

Next, we explain the transitions for all states with a new item of each one of the sizes. The probability of the transition is equal to the probability of the item, as explained later. For an item of size Z, given a state which is a collection of bins that are not ready, there are two cases. If there is a bin with a single item, the new item is packed there, and the transition it to the state which has the same set of bins excluding the bin with the single item that became ready. Since there is at most one bin with a single item for each state, the transition is well-defined. If the state has no bin with a single item, the transition is to a state that contains the same collection of bins and also OH.

For an item of size V_i, the transition is as follows. If the collection of bins of the current state has a TS_j bin such that $j \leq i$, the transition is to a state with the same collection of bins excluding the TS_j bin. Otherwise, if the collection of bins of the state has a bin with a single item, the new item is packed there and we find whether the bin becomes ready. If it is a OH bin or a OL_ℓ bin for some value of ℓ, it becomes ready. If the bin with a single item is OS_ℓ it becomes ready if $\ell > i$, and otherwise it becomes a SL_i bin. Thus the transition is to a state that contains the same collection of bins excluding the bin that was used to pack the item. If this bin did not become ready, an SL_i bin is added to the collection of the state. Note that in the case that the state has an additional bin there was no SL_i bin in the state because in the case $\ell \leq i$ there cannot be an OS_ℓ bin and an SL_i bin with $i \geq \ell$ at the same time by Lemma 7. If there is also no bin with a single item, an OL_i bin is created, and the transition is to a state that contains the same collection of bins as the previous state, together with an OL_i bin.

For an item of size U_i, the transition is as follows. If there is an SL_j bin such that $i \leq j$, the item is packed there and the bin becomes ready. Since there may be multiple such bins of the form SL_j (with $i \leq j$ and different values of j), the one with the smallest value of j is chosen. In this case we show that this is indeed the selection of BF. For two bins that

are SL_{j_1} and SL_{j_2} bins for $j_1 > j_2$, it holds that the total size for the first bin is at most $V_{j_1} + U_{j_1}$ and the total size of the second bin is at least $V_{j_2} + U_1$. It holds that the difference is at least

$$V_{j_2} + U_1 - V_{j_1} - U_{j_1} = \sum_{\ell=j_2}^{j_1-1} \varepsilon^{k-\ell+1} - (\sum_{\ell=1}^{j_1-1} \varepsilon^{k-\ell+1})/2 = (\sum_{\ell=j_2}^{j_1-1} \varepsilon^{k-\ell+1} - \sum_{\ell=1}^{j_2-1} \varepsilon^{k-\ell+1})/2 > 0,$$

since $j_1 \geq j_2 + 1$ and $j_2 \geq 1$, and similarly to previous calculations. If there is no SL_j bin with $i \leq j$, but there is an TS_j bin, that bin is used and becomes ready. Otherwise, if there is a bin with a single item, the new item is packed there. The bin becomes ready if it is an OH bin. If it is an OS_j bin, it does not become ready and it turns into an TS_p bin where $p = \max\{i, j\}$ (and there was no bin with two items of size below $\frac{1}{3}$ before this). If it is an OL_j bin, it becomes ready if $j < i$ and otherwise it becomes an SL_j bin (which did not exist because there was no SL_j bin for $j \geq i$ if we reach this stage). If there was also no bin with a single item, a new OS_i bin is created. The collection of bins of the state is modified as follows. If an existing bin was used for the item and the bin became ready, it is removed from the collection. If it did not become ready, the previous bin is removed from the collection and the new one is added instead. If the item is packed into a new bin, the new bin is added to the collection.

Using the set of states and the corresponding transitions, we construct the Markov chain, and define the system of equations corresponding to the equilibrium as a system of linear equations. We solved it for different sets of probabilities for input items using online tools. We used the property that a lower bound for a randomly created input can be used as a lower bound for randomly permuted inputs.

For inputs with N items, we use probabilities p_0 for an item of size Z and $p_1, p_2, \ldots, p_k$ such that an item of size U_i has probability of p_i while an item of size V_i has probability $p_{i+k} \leq p_i/2$. Obviously we ensure that $p_0 + \sum_{i=1}^{2k} p_i = 1$. An optimal solution has cost that is very close to

$$N \cdot (\frac{p_0}{2} + \frac{\sum_{i=1}^{2k} p_i}{3}) = \frac{N}{3} + \frac{p_0 \cdot N}{6} \ .$$

This holds since for every $1 \leq i \leq k$, if there are exactly $p_i \cdot N$ items of size U_i and $p_{i+k} \cdot N$ items of size V_i, it is possible to pack the items in triples with at most one item of size V_i in each triple (if $p_i > 2p_{i+k}$ some triples have three items of size U_i).

The (expected) cost of the algorithm is the sum of probabilities of transitions from a state without a bin consisting of a single item to a state with such a bin multiplied by N. The calculation requires the equilibrium probabilities of states, and transition probabilities are calculated using the probabilities for different items.

We have compared the results of our program on multiple inputs to the outputs of the program of [29] and always obtain the same result (at least eight first digits after the decimal point). However, for our largest inputs (for $k = 9$) we could only run our program.

In this way we get the lower bound 1.15539228 using both programs (with a cost of approximately $0.40092127 \cdot N$ for BF) using the probabilities

$$p_0 = 0.0820008,\ p_1 = 0.0848666,\ p_2 = 0.0802666,\ p_3 = 0.0762666,\ p_4 = 0.0724666,$$
$$p_5 = 0.0688666,\ p_6 = 0.0650666,\ p_7 = 0.0610666,\ p_8 = 0.0566666,\ p_9 = 0.0697,$$
$$p_{10} = 0.0424333,\ p_{11} = 0.0401333,\ p_{12} = 0.0381333,\ p_{13} = 0.0362333,$$
$$p_{14} = 0.0344333,\ p_{15} = 0.0325333,\ p_{16} = 0.0305333,\ p_{17} = 0.0283333,\ \text{and}\ \ p_{18} = 0\ .$$

The improved bound for $k = 9$ is a lower bound of 1.15582656 (with a cost of approximately $0.398760164 \cdot N$ for BF) using the probabilities (where for $1 \leq i \leq 9$, we let $p_i = 2 \cdot p_{i+9}$):

$$p_0 = 0.07,\ p_{10} = 0.0411,\ p_{11} = 0.0386,\ p_{12} = 0.0376,\ p_{13} = 0.0361,\ p_{14} = 0.0346,$$
$$p_{15} = 0.0329,\ p_{16} = 0.0311,\ p_{17} = 0.0295,\ p_{18} = 0.0285\ .$$

We have thus proved the next theorem.

▶ **Theorem 10.** *The asymptotic approximation ratio of BF on randomly permuted inputs is at least* 1.15582656*.*

4 The Absolute Approximation Ratio for Best Fit

In this section we improve the lower bound on the absolute approximation ratio of BF with random order from 1.3 to 1.5.

▶ **Proposition 11.** *The absolute approximation ratio of BF with random order is at least* 1.5*.*

Proof. We use a direct proof, considering all permutations of a fixed set of items. Let $N > 0$ be a large integer. The input consists of N items of size $\frac{1}{N}$ each (larger items) and $N + 1$ items of size $\frac{1}{N+1}$ each (smaller items). An optimal offline solution packs two completely full bins, such that one bin has all larger items and the other bin has all smaller items.

In order for a solution to use two bins rather than three bins, each bin has to contain items of total size exactly 1. We claim that a bin that has at least one item of each size cannot have a total size of exactly 1. Assume that a bin has $x \geq 1$ larger items and $y \geq 1$ smaller items, and it has a total size of 1. We have $\frac{x}{N} + \frac{y}{N+1} = 1$, and equivalently $(N+1) \cdot x + N \cdot y = N(N+1)$. Since the two variables x, y are integral, we use simple number theory arguments. The numbers $N+1$ and N are coprime. Since $(N+1) \cdot x = N(N+1-y)$, we find that x is divisible by N. Since $N \cdot y = (N+1)(N-x)$, we conclude that y is divisible by $N+1$. Thus, $x \geq N$ and $y \geq N+1$, and we get a total size of at least 2, which is a contradiction since these items fit into one bin.

There are $(2N+1)!$ possible input permutations. We find those permutations that BF packs two bins for. If the first item of the permutation is larger, all larger items have to appear consecutively in the permutation such that the first bin will not contain a smaller item. If the first item of the permutation is smaller, the first bin has to receive at least N smaller items that have to appear consecutively. This property guarantees that there is no space for a larger item (and one additional smaller item may appear later). The number of permutations where all larger items appear as the first N items is $N! \cdot (N+1)!$, due to permuting the prefix of larger items and permuting the suffix of smaller items. The permutations where there are N smaller items at the beginning differ not only by the order but also by the smaller item that appears in the suffix of length $N+1$. Thus, this number is $(N+1) \cdot N! \cdot (N+1)!$. In total, for $(N+1)(N+2) \cdot (N!)^2 = (N+2)! \cdot N!$ permutations, the number of bins is 2, while any other permutation results in (at least) three bins for BF.

We would like to show that the expected number of bins tends to 3 as N grows. Thus, we find an upper bound on $\frac{(N+2)! \cdot N!}{(2N+1)!}$:

$$\frac{(N+2)! \cdot N!}{(2N+1)!} = \frac{\prod_{i=1}^{N-1}(i+1)}{\prod_{i=1}^{N-1}(N+i+2)} \leq \frac{1}{2^{N-1}}\ ,$$

since $\frac{i+1}{N+i+2} \leq \frac{1}{2}$ for $i < N$. The expected number of bins of BF for the given input is therefore at least $3 - \frac{1}{2^{N-1}}$. ◀

References

1 S. Albers, W. Gálvez, and M. Janke. Machine covering in the random-order model. *Algorithmica*, 85(6):1560–1585, 2023. doi:10.1007/S00453-022-01011-0.

2 S. Albers and M. Janke. Scheduling in the random-order model. *Algorithmica*, 83(9):2803–2832, 2021. doi:10.1007/S00453-021-00841-8.

3 S. Albers, A. Khan, and L. Ladewig. Best Fit bin packing with random order revisited. *Algorithmica*, 83(9):2833–2858, 2021. doi:10.1007/S00453-021-00844-5.

4 S. Albers, A. Khan, and L. Ladewig. Improved online algorithms for knapsack and GAP in the random order model. *Algorithmica*, 83(6):1750–1785, 2021. doi:10.1007/S00453-021-00801-2.

5 N. Ayyadevara, R. Dabas, A. Khan, and K. V. N. Sreenivas. Near-optimal algorithms for stochastic online bin packing. In *Proc. of the 49th International Colloquium on Automata, Languages, and Programming, (ICALP'22)*, volume 229, pages 12:1–12:20, 2022. doi:10.4230/LIPICS.ICALP.2022.12.

6 J. Balogh, J. Békési, Gy. Dósa, L. Epstein, and A. Levin. A new and improved algorithm for online bin packing. In *Proc. of the 26th European Symposium on Algorithms (ESA2018)*, pages 5:1–5:14, 2018. doi:10.4230/LIPIcs.ESA.2018.5.

7 J. Balogh, J. Békési, Gy. Dósa, L. Epstein, and A. Levin. A new lower bound for classic online bin packing. *Algorithmica*, 83(7):2047–2062, 2021. doi:10.1007/S00453-021-00818-7.

8 J. Balogh, J. Békési, Gy. Dósa, J. Sgall, and R. van Stee. The optimal absolute ratio for online bin packing. *Journal of Computer and System Sciences*, 102:1–17, 2019. doi:10.1016/J.JCSS.2018.11.005.

9 J. L. Bentley, D. S. Johnson, F. T. Leighton, C. C. McGeoch, and L. A. McGeoch. Some unexpected expected behavior results for bin packing. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing (STOC'84)*, pages 279–288, 1984.

10 B. Bosek, G. Gutowski, M. Lasoń, and J. Przybyło. First-Fit coloring of forests in random arrival model. In *Proc. of the 49th International Symposium on Mathematical Foundations of Computer Science (MFCS'24)*, pages 33:1–33:10, 2024. doi:10.4230/LIPIcs.MFCS.2024.33.

11 J. Boyar and L. M. Favrholdt. The relative worst order ratio for online algorithms. *ACM Transactions on Algorithms*, 3(2):22, 2007. doi:10.1145/1240233.1240245.

12 J. Boyar, L. M. Favrholdt, and K. S. Larsen. Relative worst-order analysis: A survey. *ACM Computing Surveys*, 54(1):8:1–8:21, 2021. doi:10.1145/3425910.

13 J. Boyar, S. Irani, and K. S. Larsen. A comparison of performance measures for online algorithms. *Algorithmica*, 72(4):969–994, 2015. doi:10.1007/S00453-014-9884-6.

14 J. Boyar, K. S. Larsen, and A. Maiti. A comparison of performance measures via online search. *Theoretical Computer Science*, 532:2–13, 2014. doi:10.1016/J.TCS.2013.07.022.

15 M. G. Christ, L. M. Favrholdt, and K. S. Larsen. Online bin covering: Expectations vs. guarantees. *Theoretical Computer Science*, 556:71–84, 2014. doi:10.1016/J.TCS.2014.06.029.

16 E. Coffman Jr., J. Csirik, L. Rónyai, and A. Zsbán. Random-order bin packing. *Discrete Applied Mathematics*, 156(14):2810–2816, 2008. doi:10.1016/J.DAM.2007.11.004.

17 E. G. Coffman Jr., K. So, M. Hofri, and A. C.-C. Yao. A stochastic model of bin-packing. *Information and control*, 44(2):105–115, 1980. doi:10.1016/S0019-9958(80)90050-9.

18 J. Csirik and B. Imreh. On the worst-case performance of the *NkF* bin-packing heuristic. *Acta Cybernetica*, 9(2):89–105, 1989. URL: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3358.

19 J. Csirik and D. S. Johnson. Bounded space on-line bin packing: Best is better than first. *Algorithmica*, 31(2):115–138, 2001. doi:10.1007/S00453-001-0041-7.

20 Gy. Dósa and J. Sgall. Optimal analysis of Best Fit bin packing. In *Proc. of the 41st International Colloquium on Automata, Languages, and Programming (ICALP'14) Part I*, pages 429–441, 2014.

21 L. Epstein, L. M. Favrholdt, and J. S. Kohrt. Comparing online algorithms for bin packing problems. *Journal of Scheduling*, 15(1):13–21, 2012. doi:10.1007/S10951-009-0129-5.

22 C. Fischer and H. Röglin. Probabilistic analysis of the dual next-fit algorithm for bin covering. In *Proc. of the 12th Latin American Symposium on Theoretical Informatics (LATIN'16)*, pages 469–482, 2016.

23 C. Fischer and H. Röglin. Probabilistic analysis of online (class-constrained) bin packing and bin covering. In *Proc. of the 13th Latin American Symposium on Theoretical Informatics (LATIN'18)*, pages 461–474, 2018.

24 C. O. Fischer. *New results on the probabilistic analysis of online bin packing and its variants*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2019.

25 F. Frei, M. Gehnen, D. Komm, R. Královic, R. Královic, P. Rossmanith, and M. Stocker. Tree coloring: Random order and predictions. *CoRR*, abs/2405.18151, 2024. doi:10.48550/arXiv.2405.18151.

26 M. Garg, D. Kar, and A. Khan. Random-order online independent set of intervals and hyperrectangles. In *Proc. of the 32nd Annual European Symposium on Algorithms (ESA'24)*, volume 308, pages 58:1–58:18, 2024. doi:10.4230/LIPICS.ESA.2024.58.

27 A. Gupta, G. Kehne, and R. Levin. Random order online set cover is as easy as offline. In *Proc. of the 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS'22)*, pages 1253–1264, 2021.

28 A. Gupta and S. Singla. Random-order models. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 234–258. Cambridge University Press, 2021. doi:10.1017/9781108637435.015.

29 A. Hebbar, A. Khan, and K. V. N. Sreenivas. Bin packing under random-order: Breaking the barrier of 3/2. In *Proc. of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA'24)*, pages 4177–4219, 2024. Also in CoRR:abs/2401.04714.

30 D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974. doi:10.1016/S0022-0000(74)80026-7.

31 D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974. doi:10.1137/0203025.

32 S. Kamali and A. López-Ortiz. All-around near-optimal solutions for the online bin packing problem. In *Proc. of the 26th International Symposium on Algorithms and Computation (ISAAC'15)*, pages 727–739, 2015.

33 A. R. Karlin, S. J. Phillips, and P. Raghavan. Markov paging. *SIAM Journal on Computing*, 30(3):906–922, 2000. doi:10.1137/S0097539794268042.

34 C. Kenyon. Best-Fit bin-packing with random order. In *Proc. of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'96)*, pages 359–364, 1996. Also in research report no 95-19, Ecole Normale Supérieure de Lyon, 1996.

35 T. Kesselheim, K. Radke, A. Tönnis, and B. Vöcking. Primal beats dual on online packing LPs in the random-order model. *SIAM Journal on Computing*, 47(5):1939–1964, 2018. doi:10.1137/15M1033708.

36 C. C. Lee and D. T. Lee. A simple online bin packing algorithm. *Journal of the ACM*, 32(3):562–572, 1985.

37 T. Leighton and P. W. Shor. Tight bounds for minimax grid matching with applications to the average case analysis of algorithms. *Combinatorica*, 9(2):161–187, 1989. doi:10.1007/BF02124678.

38 A. Levin. Comparing the costs of any fit algorithms for bin packing. *Operations Research Letters*, 50(6):646–649, 2022. doi:10.1016/J.ORL.2022.09.006.

39 M. Mahdian and Q. Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing lps. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC'11)*, pages 597–606, 2011.

40 W. Mao. Best-k-fit bin packing. *Computing*, 50(3):265–270, 1993.

41 W. Mao. Tight worst-case performance bounds for Next-k-Fit bin packing. *SIAM Journal on Computing*, 22(1):46–56, 1993. doi:10.1137/0222004.

42 J. Sgall. A new analysis of best fit bin packing. In *Proc. of the 6th International Conference on Fun with Algorithms (FUN'12)*, pages 315–321, 2012.

43 G. J. Woeginger. Improved space for bounded-space online bin packing. *SIAM Journal on Discrete Mathematics*, 6:575–581, 1993. doi:10.1137/0406045.

44 G. Zhang and M. Yue. Tight performance bound of AFB_k bin packing. *Acta Mathematicae Applicatae Sinica*, 13:443–446, 1997.

A QPTAS for Facility Location on Unit Disk Graphs

Zachary Friggstad
Department of Computing Science, University of Alberta, Edmonton, Canada

Mohsen Rezapour
Department of Computing Science, University of Alberta, Edmonton, Canada

Mohammad R. Salavatipour
Department of Computing Science, University of Alberta, Edmonton, Canada

Hao Sun
Department of Computer Science, University of Houston, TX, USA

Abstract

We study the classic (Uncapacitated) Facility Location problem on Unit Disk Graphs (UDGs). For a given point set P in the plane, the unit disk graph UDG(P) on P has vertex set P and an edge between two distinct points $p, q \in P$ if and only if their Euclidean distance $|pq|$ is at most 1. The weight of the edge pq is equal to their distance $|pq|$. An instance of Facility Location on UDG(P) consists of a set $C \subseteq P$ of clients and a set $F \subseteq P$ of facilities, each having an opening cost f_i. The goal is to pick a subset $F' \subseteq F$ to open while minimizing $\sum_{i \in F'} f_i + \sum_{v \in C} d(v, F')$, where $d(v, F')$ is the distance of v to nearest facility in F' through UDG(P).

In this paper, we present the first Quasi-Polynomial Time Approximation Schemes (QPTAS) for the problem. While approximation schemes are well-established for facility location problems on sparse geometric graphs (such as planar graphs), there is a lack of such results for dense graphs. Specifically, prior to this study, to the best of our knowledge, there was no approximation scheme for any facility location problem on UDGs in the general setting.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Approximation algorithms analysis

Keywords and phrases Facility Location, Unit Disk Graphs, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.27

Funding *Zachary Friggstad*: Supported by NSERC.
Mohammad R. Salavatipour: Supported by NSERC.

1 Introduction

Unit-disk graphs (UDGs) are a well-studied class of graphs due to their extensive applications in modeling ad-hoc communication and wireless sensor networks; see for example [18, 4, 22, 10, 15, 26, 25]. UDGs are defined as intersection graphs of a collection of unit-diamater disks in the two-dimensional plane. Specifically, each UDG represents a set of n unit disks as vertices, with each vertex corresponding to one unit disk. An edge exists between two vertices/points p, q if and only if their Euclidean distance is at most 1 (equivalently, the unit-diameter balls around p and q intersect) and the weight or length of each such edge is given by their by the Euclidean distance between the corresponding vertices.

Formally, for a given point set P in the plane, the unit disk graph representation of these points, denoted as UDG(P), is a graph $G = (V, E)$ with the vertex set V, where each vertex corresponds to a point in P. The edge set E consists of edges between points p and q if and only if their Euclidean distance, denoted as $|pq|$, is at most 1. The weight of the edge $pq \in E$ is equal to their distance $|pq|$. For a given subset $S \subseteq V$, we define the (weak) diameter of S as $\mathbf{diam}(S) = \max_{x,y} d_G(x, y)$, where $d_G(x, y)$ represents the minimum weight of a path between vertices x and y in G (we assume G is connected as we may solve facility location for each connected component).

© Zachary Friggstad, Mohsen Rezapour, Mohammad R. Salavatipour, and Hao Sun;
licensed under Creative Commons License CC-BY 4.0

For many optimization problems, approximation schemes are known when the input graph is a UDG (e.g. maximum independent set, minimum dominating set, minimum clique-partition [23, 14, 5, 24]). Some of the techniques for designing PTAS's for optimization problems on UDGs (e.g. maximum independent set) involves partitioning the input into regions of bounded size (at a small loss, e.g. ignoring points that touch the boundary of the partitions) and then solving the problem on such instances using exhaustive search and/or dynamic programming which leverage Euclidean distance properties. This shifting strategy was introduced by [13].

We study the Facility Location problem on UDGs. An instance $I = (G, C, F)$ of Facility Location consists of an edge-weighted graph G, where the edges satisfy the metric property, a set $C \subseteq V$ of clients, and a set $F \subseteq V$ of facilities, each having an opening cost $f_i \in \mathbb{R}^+$. The goal is to pick a subset $F' \subseteq F$ to open to minimize $\sum_{i \in F'} f_i + \sum_{v \in C} d(v, F')$, where $d(v, F')$ is the distance of v to nearest facility in F'. Facility Location has been studied extensively and the best known upper and lower bounds for it are 1.488 [21] and 1.46 [11], respectively. Approximation schemes are known for Facility Location when the metric is Euclidean [2] or when G is a planar graph [8]. Additionally, Cohen et al. [7] developed approximation schemes for the "uniform" (that is all facilities cost 1 to open) facility location problem in minor-free graphs. To the best of our knowledge, no approximation scheme was known for any facility location type problem on UDGs. Our main result of this paper is the following:

▶ **Theorem 1.** *There is an algorithm that, given an instance of Facility Location in UDG and $\epsilon > 0$, finds a $(1+\epsilon)$-approximate solution in time $n^{O_\epsilon(\log n)}$, where the constant in $O_\epsilon(.)$ is $\epsilon^{-O(\epsilon^{-2})}$.*

In order to prove Theorem 1 we combine ideas from [8] with a low-diameter decomposition for UDGs that follows from [19, 20]. We also introduce a new dissection procedure obtained by finding a proper balanced separator for UDGs. This allows us (at a small loss) to break the problem into independent instances and use dynamic programming to combine the solutions to obtain the solution for the original instance. There are many details on how to put these pieces together carefully so as to bound the overall error.

2 Preliminaries

For planar graphs and, more generally, graphs that exclude $K_{r,r}$ as a minor for some fixed r, Klein-Plotkin-Rao [16] showed a decomposition of the input graph into low diameter parts by removing a small fraction of edges. More specifically, given a graph G with n nodes and m edges that excludes $K_{r,r}$ as a minor, one can remove $O(mr/\delta)$ edges so that the (weak) diameter of each remaining component is at most $O(r^2\delta)$. The general idea was based on chopping breadth-first search (BFS) trees (i.e. shortest-path trees in the unweighted version of the graph): suppose one constructs a BFS tree from some root node and then cut the edges at level $i \cdot \delta + r$ for $i \geq 1$ where $r \leq \delta$ is a random offset. Then repeat this procedure on each of the connected components, for $O(r)$ iterations. Then the resulting components have $O(r^2\delta)$ weak diameter. This result was further improved in [9, 1], to show for each graph without K_r as a minor there is a probabilistic decomposition into $O(r\delta)$ (weak) diameter components by removing $O(mr/\delta)$ edges. Lee [19, 20] generalized this by introducing region intersection graphs, which includes UDGs as a special case, and showed that one can obtain similar decomposition results for such graphs. Theorem 4.2 in [19] implies that a similar BFS chopping procedure applied to UDGs for a constant number of iterations results in graphs of bounded (weak) diameter. We describe this chopping procedure a bit more formally.

▶ **Definition 2** (δ-chopping operation). *For any connected graph G and any number $\delta \geq 1$, we define the δ-chopping operation on G as follows. Choose any node x_0 from $V(G)$, select a random integer $0 \leq r_0 \leq \delta$, and then compute a BFS tree from x_0. Partition $V(G)$ into annuli $A_0, A_1, A_2, \ldots$, where $A_0 = \{v \in V(G) : d'(x_0, v) < r_0\}$ and annulus A_j for $j \geq 1$ is defined as: $A_j = \{v \in V(G) : r_0 + (j-1)\delta \leq d'(x_0, v) < r_0 + \delta j\}$, where $d'(x_0, v)$ is the number of edges on the BFS tree path from x_0 to v.*

So there is an offset r_0 that cuts only a $1/\delta$-fraction of edges. Earlier works on minor free graphs [17] imply that if G is K_r-minor free if we perform this chopping procedure on each component of G recursively up to depth $O(r)$ yields components with (weak) diameter at most $O(\text{poly}(r)\delta)$. Corollary 4.3 in [19] immediately implies the following, Appendix A contains the brief argument.

▶ **Theorem 3** ([19]). *$O(1)$ iterations of the δ-chopping iteration applied to a UDG results in a graph of weak diameter $O(\delta)$.*

To prove Theorem 1 we also rely on the following result (which we prove) for the special case when the instance is in a bounded size region.

▶ **Theorem 4.** *There is a PTAS for Facility Location in UDGs when the point set P is contained within a bounding box of constant size $L = O(1)$ in the plane.*

Theorem 4 uses different techniques than discussed above. It follows from a simple reduction to prize-collecting Uncapacitated Facility Location in $\mathbb{R}^2$ for which a PTAS is known [6] after guessing an "ϵ-net" of centers in the optimum solution which is possible in polynomial time since L is bounded. The proof is deferred to Appendix B.

Our algorithm for Theorem 1 starts with some preliminary steps of algorithm of [8] that presented a PTAS for Facility Location on planar metrics. Those preliminary steps in fact are valid for general metrics (they do not use planarity in those initial steps) and reduce the problem to instances with certain structures that serve as our starting point. For this reason, we briefly outline the main steps of their algorithm. These initial steps reduce the problem to instances with certain structural properties and their proof works for general metrics (not just planar ones). Hence, the same initial reductions work in our setting as well.

2.1 Starting point: the PTAS for Facility Location on planar graphs [8]

Here, we summarize relevant results from [8] needed to prove Theorem 1.

A Well-Structured Instance

The goal of this section is to reduce Uncapacitated Facility Location in UDGs to more well-structured instances that, intuitively speaking, has the clients partitioned into annuli about facilities from some $\tilde{D} \subseteq F$ with bounded aspect ratio and that these facilities are near some facilities opened in an optimal solution. Definition 7 below contains the precise notion of what it means for an instance to be well structured. Corollary 9 is the main result from this section.

Given an instance $I = (G, C, F, f_i)$ of Facility Location, which consists of an edge-weighted graph G, a set of clients C, and a set of facilities F with opening costs f_i (for each $i \in F$), the first step of the algorithm in [8] involves partitioning the instance into separate (independent) sub-instances with specific structural properties. For any solution $D \subseteq F$, we denote by $\text{conn}(D)$ the connection cost of D ($\sum_{c \in C} dist(c, D)$) and by $\text{open}(D)$ the opening cost of facilities in D ($\sum_{i \in D} f_i$), and $\text{cost}(D) = \text{conn}(D) + \text{open}(D)$. We sometimes use $\text{cost}_I(D)$ to denote we refer

to the cost of D for instance I. To achieve this, they compute an α-approximation solution $\tilde{D}$ (where $\alpha = O(1)$) to a modified instance $\tilde{I} = (G, C, F, \epsilon f_i)$ where each opening cost is scaled down by a factor of ϵ. In other words, $\tilde{D}$ is an $O(1)$-approximation for $\tilde{I}$. It is not hard to see that $\tilde{D}$ is also an $O(1/\epsilon)$-approximation for I. For any $i \in \tilde{D}$, let $cluster(i)$ denote the set of clients connected to i in this solution and define $avgcost(i) = \frac{f_i + \sum_{j \in cluster(i)} d_G(j,i)}{|cluster(i)|}$ be the average cost of the cluster by facility i. Suppose D^* is the set of facilities in an optimum solution to I. So the cost of $(\tilde{D})$ is at most $\frac{1}{\epsilon}$ times cost of D^*. They show:

▶ **Lemma 5** (Corollary 5 [8]). $\forall f \in \tilde{D}, \exists g \in D^* : dist(f, g) \leq 2 \cdot avgcost(f)$.

Then they build a modified instance $I' = (G, C', F, f_i)$ where clients of each $cluster(f)$ are not too close or too far away from f compared to $avgcost(f)$. Let opt' be the cost of an optimum solution to I' and opt be the cost of an optimum solution to I. They show that:

▶ **Lemma 6** (Corollary 7 [8]). *For any $R \subseteq F$ if $\text{cost}_{I'}(R) \leq (1+\gamma)\text{opt}' + \delta$ (for some $\gamma, \delta > 0$) then $\text{cost}_I(R) \leq (1 + 2\gamma + 8\alpha\epsilon)\text{opt} + \delta$*

Therefore, a near optimum solution to I' yields a near optimum solution to I. In order to be able to obtain a PTAS they further partition the instance (starting from I) into several instances I_j each of which has certain structural properties.

▶ **Definition 7** (Structured Instance with Bounded Aspect Ratio). *Consider an instance of Facility Location consisting of an edge-weighted graph $G = (V, E)$, a set of clients $C \subseteq V$, and a set of facilities $F \subseteq V$ with opening costs f_i (for each $i \in F$). Suppose we are provided a set $\tilde{D} \subseteq F$ that partitions C into nonempty clusters $\{cluster(i)\}_{i \in \tilde{D}}$. We say that the instance has bounded aspect ratio of the average costs and being structured if the following properties hold:*

- **i)** *$\epsilon^2 \cdot avgcost(i) \leq d_G(j, i) \leq \epsilon^{-2} \cdot avgcost(i)$, for each $i \in \tilde{D}$ and each $j \in cluster(i)$,*
- **ii)** *for each $i \in \tilde{D}$, there exists $i^* \in D^*$ such that $d_G(i, i^*) \leq 2 \cdot avgcost(i)$,*
- **iii)** *the aspect ratio of the average costs (i.e. $\max_{i,j \in \tilde{D}} \frac{avgcost(i)}{avgcost(j)}$) is bounded by $r = \epsilon^{-O(\epsilon^{-2})}$.*

They show that one can partition I into instances $I_j = (G, C_j, F_j, f_i^j)$ such that each instance satisfies the structural properties of Definition 7 and also how to combine solutions for the various I_j to get a good solution for I':

▶ **Lemma 8** (Lemmas 10 and 11 [8]). *Given $D_j \subseteq F$ for I_j, we can build $D \subseteq F$ in polynomial time s.t. $\text{cost}_{I'}(D) \leq \sum_j \text{cost}_{I_j}(D_j) + 10\alpha\epsilon\text{opt}$. Furthermore $\sum_j \text{opt}(\text{I}_\text{j}) \leq (1 + 9\alpha\epsilon)\text{opt}$.*

Recall that all these results only use the metric property of instance I. The preceding lemmas show that to prove Theorem 1 it is sufficient to present a QPTAS for instances satisfying conditions of Definition 7 and this is what we will do. We state this formally.

▶ **Corollary 9.** *Suppose for any constant $\epsilon > 0$ there is a PTAS (resp. QPTAS) for Uncapacitated Facility Location instances in UDGs when we are additionally given $\tilde{D} \subseteq F$ and clusters $\{cluster(i)\}_{i \in \tilde{D}}$ satisfying the properties in Definition 7. Then there is a PTAS (resp. QPTAS) for any Uncapacitated Facility Location instance in UDGs.*

2.2 Overview: A recursive decomposition of UDGs

Adapting the approach from [8]

In order to get a PTAS for well-structured instances in the planar case, [8] uses a Baker-type type layering technique [3] in conjunction with the properties of the instance, and further decompose the instance into instances of constant radius at a small loss. By utilizing balanced separators for

planar graphs, they obtain a hierarchical decomposition of the plane embedding of the graph into separate regions (similar to the decomposition of Euclidean instances by Arora [2]). By placing $O(\log n)$ "portals" along each separator they use dynamic programming over this decomposition, while the portals control the interface of different regions.

In our setting, instead of using Baker layering to obtain low diameter instances, we use Theorem 3 to break the instance that satisfies conditions of Definition 7 (at a small loss) into low diameter instances. We will use a variant of a balanced separator theorem developed by [12] (which improved over a more complex separator by [27]) for UDGs. Roughly speaking, they show that given a UDG, one can find two paths originating from a vertex s to two other vertices x, y that are shortest paths, $P_{s\sim x}$ and $P_{s\sim y}$, such that the removal of these two paths and all the vertices that are within distance 1 of them leaves connected components of size at most $\frac{2}{3}|V(G)|$ (see Theorem 10). We adapt this theorem to get an (almost) balanced separator to obtain a hierarchical decomposition of a low diameter instance into smaller instances. We also place $O(\log n)$ portals at these separators and use Dynamic Programming to combine the solutions. After a logarithmic depth of hierarchical decomposition, we arrive at instances that are easy to solve using other methods (e.g. another PTAS that is described in Theorem 4).

Balanced (partly) Separators for UDGs

To obtain our dynamic program scheme, we would like to be able to break a UDG into (almost) balanced parts by picking a separator. This will act as a "cut" in the disection schema developed by Arora [2] that has been used in designing PTAS's for various optimization problems on Euclidean plane. For this, we utilize the balanced separator theorem for UDGs as presented by Yan et al. [12]. Let $N^i_G[v] = \{u \in V(G) : d_G(v, u) \leq i\}$ and $N^i_G[S] = \cup_{v\in S} N^i_G[v]$. A hop-shortest path between two nodes x, y is a path with the minimum number of edges.

▶ **Theorem 10** (Theorem 13 in [12]). *For a UDG G, $X \subset V(G)$ and a root $s \in V(G)$, there exist two nodes x, y of $V(G)$ and hop-shortest paths $P_{s\sim x} = (s, \ldots, x)$ and $P_{s\sim y} = (s, \ldots, y)$ for which the removal of $N^1_G[P_{s\sim x}] \cup N^1_G[P_{s\sim y}]$ from G yields components each having at most $\frac{2}{3}|X|$ vertices from X.*

Theorem 13 in [12] actually only proves it for the case $X = V$. See Appendix A for discussion about why it holds for general $X \subseteq V$. This theorem serves as a counterpart to the well-known balanced shortest paths separator theorem in planar graphs by Lipton and Tarjan. However, it poses a challenge due to the fact that the separator is formed by the 1-neighborhoods of the paths rather than just the paths themselves. This means that we must not only remove the shortest paths but also all nodes within a distance of one from the nodes on the shortest paths. To tackle this challenge, we narrow our focus to cases where the average distance between clients and facilities is relatively large. By doing so, we can assume that clients and facilities, which end up on two different sides of the shortest path, always get connected via nodes in $V(P_{s\sim x} \cup P_{s\sim y})$. Note that, as stated in Theorem 11 (which is a slight modification of this theorem), the only exceptions to this assumption occur when the path between clients and facilities crosses the border using an edge whose endpoints are very close to nodes in $V(P_{s\sim x} \cup P_{s\sim y})$. However, using the fact that we can assume the average distance between clients and facilities is relatively large, we can force those paths to visit nodes in $V(P_{s\sim x} \cup P_{s\sim y})$ with a relatively tiny error.

In the following, we demonstrate that a slight modification of this theorem yields a balanced, yet partial in some sense, separator for UDGs. More precisely:

▶ **Theorem 11.** *For a UDG G, $X \subset V(G)$ and a source $s \in V(G)$, there exists two nodes x, y of $V(G)$ and hop-shortest paths $P_{s\sim x} = (s, \ldots, x)$ and $P_{s\sim y} = (s, \ldots, y)$ such that removing $V(P_{s\sim x} \cup P_{s\sim y})$ partitions the vertices $V(G \backslash (P_{s\sim x} \cup P_{s\sim y}))$ into two sets G_1, G_2 each having at most $\frac{2}{3}|X|$ vertices from X. Additionally, for any edge $ab \in E(G)$ with $a \in V(G_1)$ and $b \in V(G_2)$, there exists $c \in V(P_{s\sim x} \cup P_{s\sim y})$ such that $d_G(a, c), d_G(b, c) \leq 2$.*

This follows easily from Theorem 10 but we provide a proof in Appendix A for the sake of completeness. Note that we say $V(P_{s\sim x} \cup P_{s\sim y})$ is a separator between $V(G_1)$ and (G_2) if there are no edges in G that connect a vertex from set $V(G_1)$ to a vertex from set $V(G_2)$. However, here, we relax this condition and allow for the presence of such edges, provided that their endpoints are in close vicinity to the separator.

3 Proof of Theorem 1

We can now describe our QPTAS for Uncapacitated Facility Location in UDGs. Consider an instance of Facility Location, consisting of an edge-weighted UDG $G = (V, E)$, a set of clients $C \subseteq V$, and a set of facilities $F \subseteq V$ with opening costs f_i (for each $i \in F$). Let D^* denote the set of facilities opened by the optimal solution, with opt representing the cost of this solution. Additionally, let $\tilde{D}$ denote the $O(1/\epsilon)$-approximation solution as described in Section 2.1, and let $\text{cost}(\tilde{D})$ represent the cost of this solution. Note $\text{cost}(\tilde{D}) = \sum_{i \in \tilde{D}} \left(f_i + \sum_{j \in cluster(i)} d_G(j, i) \right) = O(\text{opt}/\epsilon)$. We assume that the instance satisfies the properties mentioned in Definition 7. Let $r = \epsilon^{-O(\epsilon^{-2})}$ and $N > 0$ denote the minimum distance between a client and its facility (cluster center) in $\tilde{D}$. It can be verified using the properties of Definition 7 that the inequalities $N \leq d_G(j, f) \leq rN$ and $avgcost(i) \leq rN$ hold for each $i \in \tilde{D}$ and each $j \in cluster(i)$ (these are essentially the conditions in Lemma 9 of [8] except that we can't do scaling in UDG instances, hence we have the factor N). Moreover, based on property (ii) (Definition 7) of the instance, it follows that $d_G(j, D^*) \leq 3Nr$ holds for each client $j \in C$.

Our next step is to decompose the instance into independent subinstances.

▶ **Lemma 12.** *At a loss of $O(\epsilon \cdot \text{opt})$ we can decompose the instance into a number m of independent instances $H_1, H_2, \ldots, H_m$ where each has diameter at most $O(rN/\epsilon^2)$.*

The full proof is deferred to Appendix A. Intuitively, it uses a BFS from a node s to define layers and partitions V by breaking the graph every $O(rN/\epsilon^2)$ layers using a random offset for this partitioning. Facilities of $\tilde{D}$ in the first $O(rN)$ layers of each part are opened, which has low cost on average over the random choice of offset. This chopping procedure is recursively applied $O(1)$ times, after which each component has edge-hop diameter $O(rN/\epsilon^2)$ by Theorem 3. In the proof, we note BFS distances approximate actual weighted UDG distances within a factor of 2 for pairs of non-adjacent nodes so the actual diameter under weighted UDG distances is also $O(rN/\epsilon^2)$ for each component.

It can also be seen that the sum of optimum solutions of all these instances costs at most opt since for each client in H_ℓ their optimum facility will also be in H_ℓ. The rest of our algorithm approximates solutions in each H_ℓ with the following guarantees.

▶ **Lemma 13.** *Given instance I for Facility Location on a UDG G let $\tilde{D}$ be an approximate solution as described above, consider the instances $H_1, \ldots, H_m$ as in Lemma 12. There is a quasi-polynomial algorithm that produces solutions for H_ℓ's such that the total cost of the solutions is at most $O(\epsilon^2 \cdot \text{cost}(\tilde{D})) + (1 + O(\epsilon)) \sum_\ell \text{opt}(\text{H}_\ell)$.*

From this, we immediately get Theorem 1 since the total error is at most $O(\epsilon \cdot \text{opt})$ since the first term in the expression above is at most $O(\epsilon \cdot \text{opt})$ and $\sum_\ell \text{opt}(\text{H}_\ell) \leq \text{opt}$.

We will treat each individual instance H_ℓ separately and will produce a solution for it of cost $(1+O(\epsilon))\text{opt}(\text{H}_\ell)+\text{E}_\ell$ such that $\sum_\ell E_\ell \leq O(\epsilon^2 \cdot \text{cost}(\tilde{D}))$. A key observation that we use to bound the sum of the additive error bound E_ℓ above is the following. Suppose that N is sufficiently large, i.e. $N > 1/\epsilon^2$ (we will handle the case of small N separately). Note that (as mentioned in the first paragraph of this Section) since $N \leq d_G(j, f)$, therefore $N \cdot |C| \leq \text{cost}(\tilde{D}) = O(\text{opt}/\epsilon)$; so if N is large ($N > 1/\epsilon^2$) and each client $c \in C$ is moved an extra $O(1)$ distance, then it adds at most $O(|C|) = O(\epsilon \cdot \text{opt})$ to the cost of an optimum solution for the modified instance. Hence, this instance still has a solution of cost at most $(1 + O(\epsilon))$opt. In our algorithm for each H_ℓ we might consider paying an extra $O(1)$ for each client. Using this argument, it can be seen that the total additive error for each H_ℓ will be $O(|C_\ell|)$; summing over all the instances H_ℓ the additive error is at most $O(\epsilon \cdot \text{opt})$. So from now on we assume our instance is one of the H_ℓ and we prove Lemma 13.

3.1 Hierarchical decomposition with portalization

In this section, we assume the instance is a low diameter instance as obtained by applying our chopping operation from the proof of Lemma 12, i.e. one of the H_ℓ. More specifically, at a loss of $O(\epsilon \cdot \text{opt})$ we assume: $N \leq d_G(j, i) \leq rN$ and $avgcost(i) \leq rN$ hold for each $i \in \tilde{D}$ and each $j \in cluster(i)$, and $d_G(j, D^*) \leq 3Nr$ for each client $j \in C$, and the diameter is is at most $3rN/\epsilon^2$.

We obtain a hierarchical decomposition of the instance and describe a Dynamic Programming (DP) algorithm based on this decomposition. This decomposition has a logarithmic depth, and each region within the hierarchy is obtained by applying two shortest paths separators to the parent region (for readers familiar with the standard decomposition obtained by a dissection of the plane used in designing PTASs for problems such as TSP and Facility Location on Euclidean planes: one can think of our shortest paths separators as the "line" that breaks the problem into two balanced instances; we will place "portal" at appropriate locations along the separator). One difference in our schema is that the region at each level might have a "boundary" that is composed of separators of all the ancestor of it; so it is bounded by $O(\log N)$ separators (whereas, for e.g., a region defined in the recursive decomposition for TSP is defined by 4 dissecting lines). This is the only reason our running time becomes quasi-polynomial.[1]

Recall $r = \epsilon^{-O(\epsilon^{-2})}$. Observe that if we have a UDG with diameter at most Δ then all the points must be in a bounding box of $2\Delta \times 2\Delta$. Using this, Theorem 4 provides a PTAS when the value of N is small (i.e. at most $1/\epsilon^2$). Therefore, we can assume that $N \geq 1/\epsilon^2$. We provide a solution that has multiplicative approximation factor $(1+O(\epsilon))$ and additive factor E_ℓ that can be charged to the number of clients in the instance (eventually we will have $\sum_\ell E_\ell \leq O(\epsilon^2 \text{cost}(\tilde{D}))$.)

Indeed, assuming that N is sufficiently large is vital for our approach, as it enables us to utilize the balanced "partly" separator described in Theorem 11. This separator allows for a hierarchical decomposition of the graph with a logarithmic depth, but it does permit direct interactions between points (within a small distance from the separator path) in the separated regions. By forcing the interaction through the separator nodes, we incur an additive error. However, when the minimum distance between clients and facilities is sufficiently large, this error becomes negligible.

Sparsifying H_ℓ

Let $\Gamma = 3rN/\epsilon^2 = O_\epsilon(N)$ denote the diameter of the graph. Our goal is to obtain a net of size poly(Γ) so that the number of points we deal with is in terms of N (instead of n), while we loose a small factor (compared to opt).

[1] We conjecture a more careful analysis of our scheme and applying the separator could imply a "boundary" that is defined by only a few separators. This would turn the whole algorithm into a PTAS.

▶ **Lemma 14.** *We can obtain a graph $G'(V', E')$ where $V' \subseteq V(H_\ell)$ with $|V'| = O(\Gamma^4)$, where:*

- *For any two $u, v \in V'$, $d_G(u, v) > 1/8$.*
- *If we move each client and facility in H_ℓ to its nearest point in V', the optimum solution increases by $O(|C_\ell|)$.*
- *Conversely, given a solution to this modified instance we can get a solution to H_ℓ with additional cost $O(|C_\ell|)$.*

We let $B(v)$ for $v \in V'$ denote all clients and facilities of H_ℓ that moved to v.

We can think of $B(v)$ as the set of nodes in a ball of radius $1/8$ around v. The proof is by a fairly standard net construction and is found in Appendix A. Note that the error described above when summed over all H_ℓ's, is at most $O(|C|) = O(\text{cost}(\tilde{D})/N) = O(\epsilon \cdot \text{opt})$ (since $N|C| \leq \text{cost}(\tilde{D})$).

Hierarchical decomposition of G'

We obtain a hierarchical decomposition of G' which will be associated with a rooted binary tree T where the root of T is V'. We will have a Dynamic Programming to solve Facility Location using this decomposition. This will be similar to the hierarchical decompositions obtained for PTAS's on the Euclidean instances (e.g. [2]) except that we use the separator in Theorem 11 instead of dissecting lines and that the leaf nodes in our decomposition tree in our case do not correspond to trivial instances (typically a leaf node is a box with only one point in it). In our case, each leaf node is an instance with certain properties which enables us to solve it in quasi-polynomial time at a small loss.

Let us describe the decomposition for G'. Our hierarchical decomposition tree T is associated with a labeling $\psi : V(T) \to 2^{V(G')}$: we set the label of the root of T to be $V(G')$. Each $t \in T$ represents a subgraph $G'[\psi(t)]$ with the property that if t_1, t_2 are children of t, then $\psi(t) = \psi(t_1) \cup \psi(t_2)$. We use $\text{bd}(t)$ to denote the boundary vertices of $\psi(t)$ and the rest of the vertices, $\psi(t) - \text{bd}(t)$, we call them the *core* vertices and are denoted by $X(t)$. The "boundary" is obtained by finding a separator using Theorem 11 and is added to the boundary that is inherited from the parent (details to follow). The boundary of the root node is the empty set (and so all the vertices of G' are core vertices for the root node of T). The overall idea is whenever a current leaf node $t \in T$ has $|X(t)| > 1$ we decompose $G'[\psi(t)]$ into two smaller subgraphs and we obtain children t_1, t_2 for t. More specifically, starting from when T is a single node t corresponding to $V(G')$, iteratively, for each leaf t of T, if $|X(t)| > 1$, apply Theorem 11 over $G'[\psi(t)]$ with $X = X(t)$ to obtain two graphs G'_1 and G'_2, along with the two shortest paths $P_{s\sim x}$ and $P_{s\sim y}$. We use the same vertex s to find the two shortest paths $P_{s\sim x}, P_{s\sim y}$ for each node $t \in T$ and we make sure this vertex s is passed down. Define $\psi(t_i)$ to be $G[V(G'_i) \cup V(P_{s\sim x}) \cup V(P_{s\sim y})]$ and $\text{bd}(t_i) = \text{bd}(t) \cup P_{s\sim x} \cup P_{s\sim y}$. This also defines $X(t_i)$ to be the subset of vertices of $X(t)$ that fall into $V(G'_i)$ and not in the boundary of t_i. Note that our separator $P_{s\sim x} \cup P_{s\sim y}$ separates the vertices $\psi(t)$ of our sub-instance into parts each of which has at most $\frac{2}{3}|X(t)|$ many core vertices.

Every time we find a separator $P_{s\sim x} \cup P_{s\sim y}$ to break the graph $\psi(t)$ (as described) we also designate $m = O_\epsilon(\log N)$ of the vertices of each of these two paths as *portals*. More specifically, let $\delta = O(\epsilon\Gamma/\log N)$ and designate some of the vertices of these paths as portal so that they are δ apart (note that as mentioned before, the hop-distance and UDG distances are within factor 2 of each other). Since these paths have hop-distance length at most Γ, we will have $O_\epsilon(\log N)$ portals per path. Our intention is that if two points $u \in X(t_1)$ and $v \in X(t_2)$ are to be connected they have to go through portals of the boundary. This is illustrated in Figure 1.

Observe that the depth of T is $h = O(\log \Gamma) = O_\epsilon(\log N)$ (recall that $|V(G')| = O(\Gamma^4)$). By construction, for each node $t \in T$, the region defined by $\psi(t)$ consists of core nodes in $X(t)$ and the boundary $\text{bd}(t)$ consists of (at most) h separators (which are shortest paths starting from s)

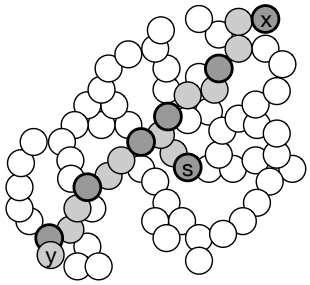


Figure 1 An depiction of two paths $P_{s\sim x}$ and $P_{s\sim y}$ from Theorem 11 in grey at the top level in the hierarchical decomposition. The balls are radius-1/2 balls around points in G', so two intersect if and only if they are adjacent. The darker grey nodes with thicker borders are evenly-spaced portals. Intuitively, at each portal we will keep track of the distance to the nearest facility we will open on either side of the separator.

obtained using Theorem 11 each of length proportional to the diameter of the graph, namely Γ. For any t let Π_t be the set of portals on the boundary of region t. Note that each region boundary is composed of at most $h = O_\epsilon(\log N)$ paths and each path has $O_\epsilon(\log N)$ portals, so each t has at most $O_\epsilon(\log^2 N)$ portals.

By Theorem 11 and the way we put the portals (since they are δ apart), it follows that for any two points $u' \in \psi(t_1)$ and $v' \in \psi(t_2)$ (where t_1, t_2 are children of a node $t \in T$), there exists a portal π in the separator that breaks t into t_1, t_2 for which $d_{G'}(u', \pi) + d_{G'}(v', \pi) \leq d_{G'}(u', v') + \delta + 4$. Now if $u \in B(u'), v \in B(v')$ are two vertices of H_ℓ (recall that $B(v')$ is the ball of v' that created net node v' in G'), the distance between u, v to go via their ball centers and then via the portal is bounded as well: $d_G(u, u') + d_{G'}(u', \pi) + d_{G'}(v', \pi) + d_G(v, v') \leq d_{G'}(u', v') + \delta + 4.25 \leq d_G(u, v) + \delta + 5$.

Suppose we require, for each node $t \in T$ with children t_1, t_2, the connection between points in $\psi(t_1), \psi(t_2)$ be via portals in the separator that broke t into t_1, t_2. As argued above, this detour adds at most $\delta + 5$ to the cost for each connection. Since the depth of the recursion tree T is $h = O(\log \Gamma)$, the accumulated error incurred to each client/facility for communicating via portals and centers of their balls (among all the levels of the decomposition) is at most $O((\delta + 5) \log \Gamma)$. Hence, there is a total error of at most $O(|C_\ell|\delta \log \Gamma)$ for all clients connections. By a suitable choice of ϵ' depending on ϵ, and setting $\delta = \frac{\epsilon' \Gamma}{\log \Gamma}$, we get $m = \frac{\log \Gamma}{\epsilon'} = \frac{\log O_\epsilon(N)}{\epsilon'}$, *the total error* for re-routing connections of clients to be via portals (among all levels of decomposition) between regions is bounded by:

$$4|C_\ell|(\epsilon' \Gamma + 5 \log \Gamma) = 4|C_\ell|(\epsilon' O_\epsilon(N) + O_\epsilon(\log N)). \tag{1}$$

This will be our additive error E_ℓ. Recall that we have $N \cdot |C| \leq O(\text{opt}/\epsilon)$, implying $|C| \leq O(\frac{\text{opt}}{\epsilon N})$. This, together with (1) and the fact that $\sum_\ell |C_\ell| \leq |C|$, imply that the total additive error for all H_ℓ's is bounded by $O(\epsilon \cdot \text{opt})$ if ϵ' is sufficiently small.

Note. The observation above (that if we re-route clients connections to be via portals adds only $O(\epsilon \cdot \text{opt})$ to the total cost) will be crucially used in our DP. In other words, if for every client/facility connection distance, we have a rounding error of δ in every level of T, then the total error across all H_ℓ's is bounded by $O(\epsilon \cdot \text{opt})$. In particular, imagine for each leaf node $t \in T$ we have moved all the points (clients/facilities) in the ball of each node in $\psi(t)$ to the nearest portal in Π_t. This adds an extra cost of $O(\epsilon \cdot \text{opt})$ over all levels of decomposition for all H_ℓ's. This simplifies our instance significantly and allows us to find a near optimum solution using Dynamic Programming. We present an overview of the DP procedure before going into details.

Overview of Dynamic Program based on T

Consider an arbitrary node $t \in T$ and portals $\pi_1, \ldots, \pi_{m'}$ (where $m' = O_\epsilon(\log^2 N)$) on the paths that define $\text{bd}(t)$. For each portal π_i we will have two values $in(\pi_i), out(\pi_i)$: $in(\pi_i)$ indicates (approximately) the distance to the nearest facility (to π_i) that is supposed to be open in the instance defined by $\psi(t)$, and $out(\pi_i)$ indicates (approximately) the distance to the nearest facility (to π_i) that will be open outside the region defined by $\psi(t)$ (and clients inside the balls of vertices of $\psi(t)$ can be connected to them via π_i by paying an additional distance cost of $out(\pi_i)$). These distances are "approximate" since we only keep multiples of δ, since having a precision parameter of δ (as argued above) results in total error $O(\epsilon \cdot \text{opt})$.

DP Table

For any $t \in T$ and any two vectors $\vec{in}, \vec{out}$ of dimensions m' (for the portals of t) we will have a DP table entry $A[t, \vec{in}, \vec{out}]$. This entry is supposed to store the (approximate) cost of an optimum solution to the Facility Location instance defined by the balls $B(\psi(t))$ (where $B(S) = \cup_{v \in S} B(v)$) subject to the following conditions:

- for each portal π_i there is an open facility in the solution with distance to π_i at most as specified by $in(\pi_i)$,
- for each client either it is served by an open facility inside, or is paying connection cost to go to a portal π_i and if $n(\pi_i)$ is the number of clients that go to portal π_i then they pay $n(\pi_i) \times out(\pi_i)$ to get serviced by a facility outside of $\psi(t)$ at distance $out(\pi_i)$. This will be part of the cost for the solution.

We say vector $\vec{in}$ (and similarly $\vec{out}$) is *valid* if it satisfies the following condition: for any two portals π, π', if their distance (rounded to the nearest multiple of δ) is z then $|in(\pi) - in(\pi')| \leq z + \delta$. This condition clearly must hold as if there is a facility with distance $in(\pi)$ from π then the nearest facility to π' cannot be further than $\in (\pi) + z + \delta$ away from it. Our DP table is computed only for valid vectors for each node $t \in T$.

Recurrence Overview

Suppose we have computed (approximate) solutions for each problem defined for each leaf node of T and each (valid) vectors $\vec{in}, \vec{out}$ for the portals. Our DP will compute the solution for the instance of root of T (i.e. $V(G')$) in a bottom up manner from the leaf nodes to the root. For each internal node t with children t_1, t_2 and vectors $\vec{in}, \vec{out}$ (for t) and $\vec{in}_1, \vec{out}_2, \vec{in}_2, \vec{out}_2$ (for t_1, t_2, respectively) it will see if the three solutions are "consistent". We will define this formally in the next section but at a high level this checks whether the solutions for t_1, t_2 (given the portal vectors) can be combined so that we get a solution for $\psi(t_1) \cup \psi(t_2)$ where it is consistent with the vectors of portals specified by t.

If one keeps the distances (stored in portal vectors) as as multiples of δ, since distances are bounded by $\Gamma = O_\epsilon(N)$, there will be $O_\epsilon(\log N)$ choices for each portal, which leads to a $(\log N)^{O_\epsilon(\log^2 N)}$ table size.[2] The base case will be solved using an exhaustive search by guessing

[2] We can reduce this using a trick that is also used earlier (e.g. see [2]) to show how to reduce the size of the portal vectors by storing only "smoothed" vectors. Informally, the observation that helps is that if we keep distances of the nearest facility (inside or outside) for a portal π_i as the multiple of δ then if the value for portal π_i is σ then the value for portal just before or after π_i on the separator path that π_i belongs to is in $\{\sigma - 1, \sigma, \sigma + 1\}$. Thus, the total number of vectors we need to consider for a node $t \in T$ is $O_\epsilon(\log^2 N \times 3^{m'}) = 2^{O_\epsilon(\log^2 N)}$, where there are $O_\epsilon(\log^2 N)$ choices for the first portal values and then for the subsequent portals there are only 3 choices for each distance.

a subset of portals and opening the cheapest facility on that portal followed by a check if the distance requirements for facilities are satisfied: the best consistent solution will be kept and we will store ∞ if there is no such solution.

3.2 Dynamic Program

In this section we describe the details of our dynamic program based on the decomposition T. As mentioned earlier, each node $t \in T$ corresponds to a set of vertices $\psi(t) \subset G'$ with boundary nodes $\text{bd}(t)$ compose of at most h separators, each of which is two paths initiating from s (using Theorem 11) and each containing m portals that are δ apart (for a total of $m' = O_\epsilon(\log^2 N)$ portals). Note that the diameter of the instance was Γ, so if we round a distance to the nearest multiple of δ, we get an integer of value at most $O(\Gamma/\delta) = O_\epsilon(\log N)$. As mentioned in the overview, for each pair of m'-dimension vectors $\vec{in}, \vec{out}$, where each entry $in(\pi_i), out(\pi_i)$ (for portal π_i) is an integer at most $O_\epsilon(\log N)$, we have an entry in our DP table $A[t, \vec{in}, \vec{out}]$. As mentioned before, we only consider valid vectors $\vec{in}, \vec{out}$.

The goal of this subproblem is to identify a set of facilities to open in $B(\psi(t))$ and to assign each client in $B(\psi(t))$ to either an open facility or bring to a portal π_i such that minimizes the total opening cost plus connection cost such that: (i) For each portal π_i, there is an open facility in $B(\psi(t))$ of distance at most $in(\pi_i)$ (rounded to the nearest multiple of δ); and (ii) For any portal π_i suppose $n(\pi_i)$ is the number of clients in $B(\psi(t))$ that are connected to π_i by the solution. The connection cost for these clients is the cost they pay to be connected to π_i plus $n(\pi_i) \times out(\pi_i)$.

A well-structured near-optimum solution

We first show the existence of a near optimum solution with certain properties such that our DP actually finds such a near optimum solution. Starting from an optimum solution D^*_ℓ on H_ℓ we make some changes to it to satisfy the properties we want, while increasing the cost a little only. First for each node $v \in V'$ with $D^*_\ell \cap B(v) \neq \emptyset$ we consider keeping the cheapest open facility in $B(v)$ and closing all the others and re-routing all the clients that were served by other facilities in $B(v)$ to that single open facility via v. This adds at most $1/4$ to the distance each client has to travel (via v) as nodes in $B(v)$ have distance at most $1/8$ to v. This increases the cost by at most $O(|C_\ell|)$ over all clients. We can assume the open facilities are on the centers of the balls now. Let's call this new (near optimum) solution D'_ℓ. Next we modify the solution further by the following process. We say a solution is t-adapted for a node $t \in T$ if (i) for each of its descendant t' the solution is t'-adapted, and (ii) for each client $c \in B(\psi(t))$, if c is connected to a facility outside of $B(\psi(t))$ then it is connected via a portal $\pi \in \Pi$, where Π is the set of portals of t.

We start with D'_ℓ and at leaf nodes of T and going up the tree, we make the solution t-adapted for each $t \in T$ at small increase in cost. Let us consider a leaf node $t \in T$ with $X(t) = w_t$ (the case when $X(t) = \emptyset$ is even easier) and boundary $\text{bd}(t)$ corresponds to paths with a total of m' portals $\Pi = \pi_1, \ldots, \pi_{m'}$. We consider w_t as a portal as well and add it to Π. Suppose $\Pi' = \pi_{a_1}, \pi_{a_2}, \ldots, \pi_{a_\sigma}$ is the subset of Π, where there is a facility in distance at most δ of π_{a_i} in $D' \cap \psi(t)$. For each such π_{a_i}, we assume we have kept open the cheapest facility within δ of it. For any client $c \in B(\psi(t))$ if c was connected to a facility in $B(\psi(t))$ in D' (note that all of those are within distance δ of some portal in Π') we consider routing c to the nearest portal in Π' (and then from there to the single cheap facility we kept open). Note that this increases the connection cost for each client by at most 2δ. If c was connected to a facility outside of $B(\psi(t))$ we re-route it first to a nearest portal π along its way and then from there to be connected to the facility it was connected to outside of $B(\psi(t))$ (i.e. we make the connection of c to go through a portal). This increases the connection cost for each c by at most $\delta + 5$ as argued earlier in the overview and the solution becomes t-adapted.

Now suppose $t \in T$ is a non-leaf node with children t_1, t_2 and supposed D'_ℓ is t_1-adapted and t_2-adapted. We make it t-adapted with small increase in the cost. For any client $c \in B(\psi(t))$, if c is connected to a facility in $\psi(t)$ we don't need to make any further changes. Otherwise we make a detour for the connection of c to go through one of the portals Π. This detour increases the connection cost of a client by at most 2δ.

One last change we do is in the calculation of the cost of the adapted solution to make it **portal vector** adapted: for each node $t \in T$, the t-adapted solution also induces vectors $\tilde{in}, \tilde{out}$ for portals of t in the following way. The clients that are served by facilities outside $\psi(t)$ are first going to a portal π (of t). The distance from that portal to the nearest open facility (outside $\psi(t)$) rounded up to the nearest multiple of δ is what induces a value for $\tilde{out}(\pi)$ at node t. We use this rounded (up) value instead of the actual distance in the calculation of the cost of the t-adapted solution. Similarly, for each portal π, the distance to the nearest open facility in $\psi(t)$ rounded to the nearest multiple of δ induces a value $\tilde{in}(\pi)$ for node t. Let $\tilde{in}, \tilde{out}$ correspond to the vectors induced by the t-adapted optimum solution. We assume the cost a client pays to go out of $\psi(t)$ to be connected to a facility via a portal π (from π) is $\tilde{out}(\pi)$. Note that our estimate of the nearest to π open facility inside or outside $\psi(t)$, described by $\tilde{in}(\pi), \tilde{out}(\pi)$ has an additive error of at most δ. Thus, the connection costs for each client at each node t can be larger by δ again. We call this new cost, portal vector adapted.

It is easy to see that if we make the solution t_0-adapted, where t_0 is the root of T, and consider the portal adapted cost (as described), then the increase in the connection cost for each client over all the nodes of T is at most $O(\delta \log \Gamma)$ (since the height of T is $O(\log \Gamma)$); summed over all the clients this was shown to be $O(\epsilon' |C_\ell| \Gamma)$. Thus, the best t_0-adapted and portal adapted solution has cost $\mathrm{opt}(\mathrm{H}_\ell) + \mathrm{O}(\epsilon' |\mathrm{C}_\ell| \Gamma)$, and for a suitable choice of ϵ' the additive error over all H_ℓ's will be add to at most $O(\epsilon \cdot \mathrm{opt})$.

Recurrence details

Let $\tilde{in}, \tilde{out}$ correspond to the vectors induced by the t_0-adapted near optimum solution. By this argument it is enough to find compute the entries $A[t_0, \vec{0}, \vec{0}]$ to obtain a $(1 + O(\epsilon))$-approximate solution.

Base case. Let us consider a leaf node $t \in T$ with $X(t) = w_t$ (the case when $X(t) = \emptyset$ is even easier) and boundary $\mathrm{bd}(t)$, which corresponds to paths with a total of m' portals $\Pi = \pi_1, \ldots, \pi_{m'}$. We consider w_t as a portal as well and add it to Π. For any subset $\Pi' = \pi_{a_1}, \pi_{a_2}, \ldots, \pi_{a_\sigma}$ of Π, where there is a facility in $B(\pi_{a_i})$, we consider opening the cheapest facility in $B(\pi_{a_i})$; let's call that facility $f(a_i)$. For any client $c \in B(\psi(t))$ we consider routing c to (i) nearest portal with an open facility, or (ii) to a portal π_{a_i} to be connected outside at a total cost $d_G(c, \pi_{a_i}) + out(\pi_i)$ if this is less than the distance to the nearest portal with an open facility. This will be considered a feasible solution if: for each portal π_i there is a portal $\pi_{a_j} \in \Pi'$ with an open facility such that $d_G(\pi_i, \pi_{a_j}) \leq in(\pi_i)$. The cost for $A[t, \vec{in}, \vec{out}]$ will be the cost of the cheapest feasible solution over all choices of Π' as described above. If there is no such solution (consistent with vectors $\vec{in}, \vec{out}$, we set $A[t, \vec{in}, \vec{out}] = \infty$. It is easy to see that we obtain the best t-adapted portal adapted solution.

Filling in the rest of DP table. Now consider an arbitrary (non-leaf) node $t \in T$ and vectors $\vec{in}, \vec{out}$ and suppose t has children t_1, t_2. Suppose all the entries of t_1, t_2 for all vectors of portals are computed. Let Π, Π_1, Π_2 be the set of portals of t, t_1, t_2, respectively. For vectors $\vec{in}_1, \vec{out}_1$ for portals of t_1, and vectors $\vec{in}_2, \vec{out}_2$ for portals of t_2 we say subproblems $(t, \vec{in}, \vec{out})$, $(t_1, \vec{in}_1, \vec{out}_1)$, $(t_2, \vec{in}_2, \vec{out}_2)$ are consistent if the following hold:

- For each portal $\pi \in \Pi$, either $\vec{in}_1(\pi) = \vec{in}(\pi)$ or $\vec{in}_2(\pi) = \vec{in}(\pi)$.
- For each portal $\pi \in (\Pi_1 \cap \Pi_2) - \Pi$, i.e. a portal that is on the separator of t that creates t_1, t_2: $\vec{in}_1(\pi) = \vec{out}_2(\pi)$ and $\vec{in}_2(\pi) = \vec{out}_1(\pi)$.
- for each $\pi \in (\Pi_1 \cap \Pi_2 \cap \Pi)$ we must have $\vec{out}_1(\pi) = \vec{out}_2(\pi) = \vec{out}(\pi)$.

First observe that checking consistency for the three subproblems can be done in time poly(m). Then

$$A[t, \vec{in}, \vec{out}] = \min\{A[t_1, \vec{in}_1, \vec{out}_1] + A[t_2, \vec{in}_2, \vec{out}_2]\},$$

where the min is over all vectors $\vec{in}_1, \vec{out}_1, \vec{in}_2, \vec{out}_2$ such that $(t, \vec{in}, \vec{out})$, $(t_1, \vec{in}_1, \vec{out}_1)$, $(t_2, \vec{in}_2, \vec{out}_2)$ are consistent. By induction, assuming that $\vec{in}, \vec{out}, \vec{in}_1, \vec{out}_1, \vec{in}_2, \vec{out}_2$ are induced portal vectors for a t-adapted near optimum solution D' and that $A[t_1, \vec{in}_1, \vec{out}_1]$ and $A[t_2, \vec{in}_2, \vec{out}_2]$ are computed correctly, one can see that we get that the cost of a solution at $A[t, \vec{in}, \vec{out}]$ that is no more than that of optimum t-adapted solution at induced on $B(\psi(t))$.

Run time analysis. Note that diameter of a (connected) UDG is at most n (and we assume the graph is connected since we can run the algorithm on each connected component). Thus $N = O(n)$ and hence the size of the DP is $2^{O_\epsilon(\log^2 N)} = 2^{O_\epsilon(\log^2 n)}$. To compute each base case it takes $2^{O_\epsilon(\log^2 n)}$ time and for each non-leaf node of t which children t_1, t_2 and vectors $\vec{in}, \vec{out}$, the solution $A[t, \vec{in}, \vec{out}]$ can be computed by comparing all triples of valid solutions for t, t_1, t_2; this also takes $2^{O_\epsilon(\log^2 n)}$. Overall the runtime is therefore $n^{O_\epsilon(\log n)}$, where the constant in $O_\epsilon(.)$ is $\epsilon^{-O(\epsilon^{-2})}$.

References

1 Ittai Abraham, Cyril Gavoille, Anupam Gupta, Ofer Neiman, and Kunal Talwar. Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 79–88, 2014. doi:10.1145/2591796.2591849.

2 Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for euclidean k-medians and related problems. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 106–113, 1998. doi:10.1145/276698.276718.

3 Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, January 1994. doi:10.1145/174644.174650.

4 Sergio Cabello and Miha Jejčič. Shortest paths in intersection graphs of unit disks. *Computational Geometry*, 48(4):360–367, 2015. doi:10.1016/J.COMGEO.2014.12.003.

5 Xiuzhen Cheng, Xiao Huang, Deying Li, Weili Wu, and Ding-Zhu Du. A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. *Networks*, 42(4):202–208, 2003. doi:10.1002/net.10097.

6 Vincent Cohen-Addad, Andreas Emil Feldmann, and David Saulpic. Near-linear time approximation schemes for clustering in doubling metrics. *Journal of the ACM (JACM)*, 68(6):1–34, 2021. doi:10.1145/3477541.

7 Vincent Cohen-Addad, Philip N Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in euclidean and minor-free metrics. *SIAM Journal on Computing*, 48(2):644–667, 2019. doi:10.1137/17M112717X.

8 Vincent Cohen-Addad, Michał Pilipczuk, and Marcin Pilipczuk. A polynomial-time approximation scheme for facility location on planar graphs. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 560–581. IEEE, 2019. doi:10.1109/FOCS.2019.00042.

9 Jittat Fakcharoenphol and Kunal Talwar. An improved decomposition theorem for graphs excluding a fixed minor. In *RANDOM-APPROX*, pages 36–46, 2003. doi:10.1007/978-3-540-45198-3_4.

10 Jie Gao and Li Zhang. Well-separated pair decomposition for the unit-disk graph metric and its applications. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 483–492, 2003. doi:10.1145/780542.780613.

11 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of algorithms*, 31(1):228–248, 1999. doi:10.1006/JAGM.1998.0993.

12 Elfarouk Harb, Zhengcheng Huang, and Da Wei Zheng. Shortest path separators in unit disk graphs. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 66:1–66:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.66.

13 Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 32(1):130–136, January 1985. doi:10.1145/2455.214106.

14 Harry B Hunt, Madhav V Marathe, Venkatesh Radhakrishnan, S.S Ravi, Daniel J Rosenkrantz, and Richard E Stearns. Nc-approximation schemes for np- and pspace-hard problems for geometric graphs. *Journal of Algorithms*, 26(2):238–274, 1998. doi:10.1006/jagm.1997.0903.

15 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Routing in unit disk graphs. *Algorithmica*, 80:830–848, 2018. doi:10.1007/S00453-017-0308-2.

16 Philip Klein, Serge A Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 682–690, 1993. doi:10.1145/167088.167261.

17 Philip Klein, Serge A. Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 682–690, New York, NY, USA, 1993. Association for Computing Machinery. doi:10.1145/167088.167261.

18 Fabian Kuhn, Tim Nieberg, Thomas Moscibroda, and Rogert Wattenhofer. Local approximation schemes for ad hoc and sensor networks. In *Proceedings of the 2005 joint workshop on Foundations of mobile computing*, pages 97–103, 2005. doi:10.1145/1080810.1080827.

19 James R. Lee. Separators in region intersection graphs, 2016. arXiv:1608.01612.

20 James R. Lee. Separators in Region Intersection Graphs. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*, volume 67 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:8, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2017.1.

21 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58, 2013. doi:10.1016/J.IC.2012.01.007.

22 Xiang-Yang Li, Wen-Zhan Song, and Yu Wang. Efficient topology control for ad-hoc wireless networks with non-uniform transmission ranges. *Wireless Networks*, 11(3):255–264, 2005. doi:10.1007/S11276-005-6609-4.

23 Tomomi Matsui. Approximation algorithms for maximum independent set problems and fractional coloring problems on unit disk graphs. In Jin Akiyama, Mikio Kano, and Masatsugu Urabe, editors, *Discrete and Computational Geometry*, pages 194–200, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

24 Imran A. Pirwani and Mohammad R. Salavatipour. A weakly robust PTAS for minimum clique partition in unit disk graphs. *Algorithmica*, 62(3-4):1050–1072, 2012. doi:10.1007/s00453-011-9503-8.

25 Erik Jan van Leeuwen. Approximation algorithms for unit disk graphs. In *Graph-Theoretic Concepts in Computer Science: 31st International Workshop, WG 2005, Metz, France, June 23-25, 2005, Revised Selected Papers 31*, pages 351–361. Springer, 2005. doi:10.1007/11604686_31.

26 Chenyu Yan, Yang Xiang, and Feodor F Dragan. Compact and low delay routing labeling scheme for unit disk graphs. *Computational Geometry*, 45(7):305–325, 2012. doi:10.1016/J.COMGEO.2012.01.015.

27 Chenyu Yan, Yang Xiang, and Feodor F. Dragan. Compact and low delay routing labeling scheme for unit disk graphs. *Computational Geometry*, 45(7):305–325, 2012. doi:10.1016/j.comgeo.2012.01.015.

A Missing Proofs

Proof of Theorem 3. Unit disk graphs are *string graphs* - intersection graphs of continuous arcs in the plane. Lemma 1.4 in [19] then implies each unit disk graphs is a so-called *region intersection graphs* over a planar graph (see the introduction in [19] for a definition). Since planar graphs exclude the complete graph K_5 as a minor, then Corollary 4.3 in [19] immediately implies the result. ◀

Proof of Theorem 10. We simply describe how to adapt the proof in [12] to the more general setting with $X \subseteq V$. Intuitively, [12] reduces their separators to those of finding separators in planar graphs. In fact, they use a vertex-weighted separator theorem in planar graphs in their proof (see Theorem 2 in [12]) to distinguish nodes of V from auxiliary nodes created in their reduction.

Precisely, in the proof of Lemma 11 of [12] one can make the following modification to the start of the second paragraph: for vertices $u_1, \ldots, u_{\ell(u)} \in V'$ that correspond to some vertex $u \in V$, if $u \in X$ then we give a weight of 1 to u_1 and a weight of 0 to $u_2, \ldots, u_{\ell(u)}$ and if $u \in V - X$ then give all $u_1, \ldots, u_{\ell(u)}$ a weight of 0. The subsequent appeal to finding balanced separating cycles vertex-weighted planar graphs ensures the final UDG separator leaves each component having at most $\frac{2}{3} \cdot |X|$ nodes from X. ◀

Proof of Theorem 11. Consider shortest paths $P_{s\sim x} = (r, \ldots, x)$ and $P_{s\sim y} = (r, \ldots, y)$ by Theorem 10, partition the components of $G \backslash (N^1_G[P_{s\sim x}] \cup N^1_G[P_{s\sim y}])$ into two graphs G'_1 and G'_2 each containing at most $\frac{2}{3}|X|$ vertices from X.

Partition $(N^1_G[P_{s\sim x}] \cup N^1_G[P_{s\sim y}]) \backslash (P_{s\sim x} \cup P_{s\sim y})$ into two sets S_1, S_2 such that each $G'_i \cup S_i$ (for $i = 1, 2$) has size at most $\frac{2}{3}|X|$ vertices from X: such a partition can be obtained greedily by starting with $S_1 = S_2 = \emptyset$ and then iteratively adding nodes from $(N^1_G[P_{s\sim x}] \cup N^1_G[P_{s\sim y}]) \backslash (P_{s\sim x} \cup P_{s\sim y})$ to either S_1 or S_2 while maintaining $|(V(G'_i) \cup S) \cap X| \leq \frac{2}{3} \cdot |X|$. Observe this property is true initially when $S_1 = S_2 = \emptyset$ and it is trivial to maintain by adding each new node to part with the fewest nodes from X.

We claim that $G_i = G'_i \cup S_i$ satisfies the required property. Let $ab \in E(G)$ be such that $a \in V(G_1), b \in V(G_2)$, then it cannot be that $a \in G'_1$ and $b \in G'_2$ by Theorem 10. Without loss of generality, say $b \notin G'_2$. Then there exists $c \in V(P_{s\sim x} \cup P_{s\sim y})$ such that $cb \in E(G)$. Then a, b, c is a path of length 2 from a to c. ◀

Proof of Lemma 12. We begin with the following observation: If T is a BFS tree from a vertex s in a UDG G, then for any two vertices u, v at levels $i, i+2$ of the tree respectively (for any i) we have their (weighted) distance in G is strictly larger than 1 (or else they would be adjacent and hence cannot be at two levels $i, i+2$ of the BFS tree) and no more than 2 (as any two adjacent vertices have distance at most 1), i.e. $1 < d_G(u, v) \leq 2$. Thus, the BFS will 2-approximate actual (weighted) shortest paths for any node that is not a neighbour of s.

Now suppose we run a BFS from an arbitrary node s and group the vertices into layers where layer i consists of all the nodes of BFS at levels between $(i-1)14Nr, \ldots, 14iNr - 1$. Note that the "thickness" of a layer is $14Nr$ levels of BFS, so for any two vertices u, v in layers $i, i+2$: $d_G(u, v) \geq 7Nr$. Since the distance of any client to their facilities in the optimum is at most $3Nr$, no path from a client to a facility (in the optimum) would cross an entire layer. Now we group consecutive layers into bundles of $\lceil \frac{1}{\epsilon^2} \rceil$ layers, with a random off-set chosen from $0, \ldots \lceil \frac{1}{\epsilon^2} \rceil$. Suppose we call the first layer of each bundle a "red" layer and all the other layers of a bundle are blue; so between every two "red" layers we have $\lceil \frac{1}{\epsilon^2} \rceil - 1$ blue layers. We open all the facilities of $\tilde{D}$ in the red layers and serve the clients in their cluster. Based on the random shift and the fact that $\tilde{D}$ was a $O(1/\epsilon)$-approximate solution, the total cost incurred to open these facilities and serve their clients is at most $O(\epsilon^2 \cdot \text{cost}(\tilde{D})) = O(\epsilon \cdot \text{opt})$. So we can assume all these facilities are

open (i.e. have zero opening cost) and we delete the clients they have served. For any other client left in the red layer, they can be partitioned into two parts: those in the top $7Nr$ levels are called *top* red clients and those in the bottom $7Nr$ levels are called *bottom* clients. Since for each client j, $d_G(j, D^*) \leq 3Nr$, the top clients cannot cross over the bottom $7Nr$ levels to be served by a facility. Similarly, the bottom clients cannot be crossing the top $7Nr$ levels to be served by a facility. We show how this breaks the instance into independent instances.

For the remaining (blue) layers in every bundle we consider the clients and facilities in those layers, together with the facilities and remaining clients in the nearest $7Nr$ levels of the two red layers above and below them. Note that these instances are now independent since for every client j, $d_G(j, D^*) \leq 3Nr$, so no client in a blue layer would need to pass beyond $7Nr$ levels into a red layer to reach its facility in optimum. Similarly, the remaining red clients will only need the facilities in the blue layers they are grouped with. This means we can solve the blue layers of each bundle (together with the facilities and clients in a strip of $7Nr$ layers above and below) independently. So we consider the blue layers of each bundle plus the $7Nr$ (red) layers above and below as one instance (recall the facilities in the red layers are open). This means we can assume we have deleted any connections (edges) between these independent instances. This is similar to one round of chopping in the proof of Theorem 3. We perform a sequence of $O(1)$ chopping rounds as above on the graph and utilizing Theorem 3, we can assume that the weak diameter of each independent instance generated is bounded by rN/ϵ^2 and the total cost paid for the facilities in the layers chopped is $O(\epsilon \cdot \text{opt})$; those facilities now have opening cost zero.

So from now on (at a loss of $O(\epsilon \cdot \text{opt})$) we focus on each independent instance where the weak diameter is bounded by rN/ϵ^2. Let's call these instances $H_1, H_2, \ldots$. For each such instance H_ℓ we use C_ℓ to denote the clients that belong to H_ℓ. It is easy to see that the C_ℓ's are disjoint. Next, we modify H_ℓ's so that they have bounded diameter (not just weak diameter): we add all the vertices of G that are within distance rN/ϵ^2 of some vertex of H_ℓ to H_ℓ. Now for each H_ℓ we have that the diameter (not just weak diameter) of H_ℓ is bounded by $3rN/\epsilon^2$. Note that the set of clients and facilities of H_ℓ is the same as before (we do not bring in the clients and facilities that were outside of H_ℓ when adding vertices to bound the diameter). ◀

Proof of Lemma 14. We construct a new graph G' by selecting a subset of vertices from $V(H_\ell)$, denoted as V' (V' will be a "net"): start by adding an arbitrary vertex of $V(H_\ell)$ to V' and then iteratively include nodes from $V(H_\ell)$ that have a minimum Euclidean distance of at least $1/8$ from nodes in V' (alternatively we can think of picking a node from $V(H_\ell)$ to be added to V' and then deleting all the nodes within distance $1/8$ of it from $V(H_\ell)$ iteratively). Note that after this round is done, $|V'|$ is $O(\Gamma^2)$. Furthermore, for each pair of vertices $u, v \in V'$, if there is a vertex u' within distance $1/8$ of u in $V(H_\ell) - V'$ and a vertex v' within distance $1/8$ of v in $V(H_\ell) - V'$ where $u'v' \in E(G)$ then add both u', v' to V'. Note that in this case, $uu', vv', u'v'$ all are edges in G'. This augmentation results in the graph G' (over V') with a total number of vertices of $O(\Gamma^4)$. We arbitrarily order the nodes of G' and for each node $v' \in G'$, we define $B(v')$ as the set of nodes in H_ℓ that lie inside the Euclidean ball of radius $1/8$ centered at v' but outside the balls of previously processed nodes. We focus on the UDG induced by G'.

To see why we can focus on the net G' and how it helps: each client and facility in the instance H_ℓ is moved to its nearest point in V' (keeping only the cheapest facility moved to a point $v \in V'$ if multiple move there). This instance has a solution of cost $O(|C_\ell|)$ more than with H_ℓ since each client moves at most two extra steps of distance $\leq 1/8$. Conversely, given any solution to this new instance G' we obtain a solution in H_ℓ by opening the same set of facilities except at their original locations. Again, clients move an additional $O(1)$ each when translating from the solution in G' to the solution in H_ℓ.

So the total error will be at most $O(|C_\ell|)$ (whereas the cost of optimum for H_ℓ was at least $O(N|C_\ell|)$). Also, it is easy to see that the size of the graph G', is in terms of N now ($O(\Gamma^4)$). ◀

B PTAS for Facility Location on UDG in Bounded Regions

In this section we present a PTAS for Facility Location in UDG in the special case that the point set P is contained within a bounding box of size $L \times L$ in the plane where L can be regarded as a constant, i.e. prove Theorem 4.

Consider an instance of Facility Location consisting of an edge-weighted graph $G = UDG(P)$, a set of clients $C \subseteq P$, and a set of facilities $F \subseteq P$ with opening costs f_i (for each $i \in F$). Let $D^* \subseteq F$ be the facilities in an optimum solution and let $i^*_j \in D^*$ denote the facility that serves the client j in D^*. Suppose we know D^* (this assumption will be removed) and let $\epsilon > 0$ be the error parameter. We greedily form an ϵ-*net* F' as follows:

- Sort the facilities in D^* by their opening costs in non-decreasing order.
- In this order, while there is some $i \in D^*$ such that $d_G(i, F') > \epsilon$, add i to F'.

This procedure ensures that the resulting F' forms an ϵ-net, where no facility in D^* is at a distance greater than ϵ from F'.

▷ Claim 15. $|F'| \leq O(L/\epsilon^2)$.

Proof. The balls of radius $\epsilon/2$ centered at each point in F' are interior-disjoint. These balls collectively occupy a total area of $\Omega(\epsilon^2 \cdot |F'|)$. The proof follows from the fact that the balls are entirely contained within a square of side length $L + \epsilon$. ◁

Now we divide the instance into sub-instances using a random grid of size $1/2$ that splits the bounding box into squares of size at most $1/2 \times 1/2$. Note $d_G(i, j) \leq 1$ for any two points i and j lying in the same cell. As a result, the metric between points within a cell can be treated as Euclidean distance. For any client $j \in P$, we say j is **cut** if j and i^*_j lie in different grid cells. Let $c^*_j = d_G(j, i^*_j)$.

▷ Claim 16. For any point $j \in P$, $\mathbf{Pr}[j \text{ is cut}] \leq 2c^*_j$.

Proof. This is obvious if $c^*_j \geq 1/2$, so let's assume $c^*_j < 1/2$. The probability that a horizontal line in the grid separates points p, q is at most $|pq|$. Same when considering vertical lines in the random grid. Since $c^*_j < 1/2$, then the centre serving j has a direct connection with j so $c^*_j = d_G(j, i^*)$ as well. ◁

For each grid cell c, let X_c be the restriction of the points in the input to cell c. Recall that, in the prize-collecting version of the facility location problem (Prize Collecting Facility Location), in addition to the input for facility location, each client j is associated with a penalty cost π_j. This penalty cost can be paid instead of the connection cost. The goal is to find an optimal solution that minimizes the total cost, including opening costs and both connection costs and penalties. Define an Euclidean Prize Collecting Facility Location instance for each cell c. For $j \in X_c$, its penalty is $\pi_j := d_G(j, F')$. Let $D^*_c := (D^* - F') \cap X_c$ be the optimum facilities in cell c that are not part of the net.

▷ Claim 17. The optimum Prize Collecting Facility Location solution for this instance has cost at most

$$\sum_{i \in D^*_c} f_i + \sum_{j \in X_c} c^*_j + \sum_{j \in X_c : j \text{ cut}} \epsilon.$$

Proof. Consider the solution that opens D^*_c. If a point j is not cut, we can directly connect it to its centre in D^*_c paying a cost of c^*_j (since the cell c has dimensions $1/2 \times 1/2$ then the direct connection is possible).

Otherwise, we can pay the penalty for j. Note this is upper bounded by moving j to its optimum centre in D^* (paying c_j^*) and then from there to the nearest net point (paying an additional ϵ). ◁

Proof of Theorem 4. Consider the following algorithm:

1. For all possible choices of $F' \subset F$ with $|F'| \leq O(L/\epsilon^2)$ do
   - Partition the instance into sub-instances using a random grid of size 1/2 . Let $\mathcal{C}$ be the corresponding cells.
   - For each $c \in C$ run a PTAS on the corresponding Euclidean Prize Collecting Facility Location instance [6].
   - Obtain a solution for the facility location instance: open all facilities in set F', as well as the facilities opened by PTASs in each cell. For every $j \in P$ that paid a penalty in its corresponding Prize Collecting Facility Location instance, assign j to its nearest (in the UDG metric) facility in F'.
2. Output a minimum cost solution, among the solutions obtained.

It is sufficient to show that ϵ-*net* F' satisfies the claim. Using the previous claims, we obtain the following. The cost of opening all facilities in ϵ-*net* F' plus expected total cost of all Facility Location solutions for all cells c is at most:

$$\sum_{i \in D^*} f_i + \sum_{j \in X_c} (1 + 2 \cdot \epsilon) \cdot c_j^* \leq (1 + O(\epsilon))\Big(\sum_{i \in D^*} f_i + \sum_{j \in P} c_j^*\Big)$$

using the fact that for each client j, we always pay c_j^* and, perhaps, an additional ϵ if j is cut. But j is cut with probability at most $2 \cdot c_j^*$. ◀

Approximation Algorithms for the Generalized Point-To-Point Problem

Zachary Friggstad
Department of Computing Science, University of Alberta, Edmonton, Canada

Mohammad R. Salavatipour
Department of Computing Science, University of Alberta, Edmonton, Canada

Hao Sun
Department of Computer Science, University of Houston, TX, USA

Abstract

We consider the Generalized Point-to-Point (GP2P) problem in which we have an edge-weighted graph G with (possibly negative) node charges $\phi(v) \in \mathbb{Z}$. The goal is to find a minimum-cost set of edges such that each component has nonnegative total charge. Viewing the positive charges as specifying supply and negative charges as demand quantities at various nodes, the problem is equivalent to build the cheapest network so that it is possible to satisfy all demands by routing supplies across the network.

This problem is a significant generalization of other network design problems such as the well-studied Steiner Forest problem. Even the special case of only having one single demand point (having charge $-k$ and all the other nodes having charge $+1$) is capturing the k-Minimum Spanning Tree problem. Earlier work by Hajiaghayi et al. (2016) [10] gave an $O(\log n)$ approximation in pseudo-polynomial time with further improved guarantees if the total supply is not much larger than the total demand, and also a 2-approximation if the total supply equals the total demand.

Our contributions are four-fold: (a) we show how known k-Minimum Spanning Tree approximations can be extended to GP2P approximations while losing only a ϵ-factor if the number of demand points in the instance is bounded by a constant, (b) we improve the running time to be Fixed-Parameter Tractable (FPT) in the number of demand points in constant-dimensional Euclidean metrics, (c) we give a 2-approximation in instances where edge costs are all 1 and $\phi(v) = \pm 1$ for each node v and show such instances are **APX**-hard, and (d) we show how the logarithmic approximations in earlier work can be modified to run in truly polynomial time.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Approximation algorithms analysis

Keywords and phrases Point-to-Point Network design, Approximation, Steiner Forest, k-MST

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.28

Funding *Zachary Friggstad*: Supported by NSERC.
Mohammad R. Salavatipour: Supported by NSERC.

1 Introduction

Consider the following setting in a network: some locations have a finite supply of goods and some locations have a given demand for goods. The goal is, naturally, to route supplies to the demand locations in order to satisfy all demands. For example, in the Minimum-Cost Transportation the goal is to do this in a way that minimizes the total travel cost of all goods. However, it is natural to consider variations of this problem. A network-design version would be to install a minimum-cost set of connections (edges between locations) so that it is possible to find a routing satisfying demands such that each unit of supply only travels along installed connections. That is, we only pay one for each link. Such a setting is captured by the following problem.

© Zachary Friggstad, Mohammad R. Salavatipour, and Hao Sun;
licensed under Creative Commons License CC-BY 4.0

▶ **Definition 1.** *In the* Generalized Point to Point Connection *problem (GP2P), we are given a tuple* $(G = (V, E), c, \phi)$ *where* G *is an undirected graph with edge costs* $c(u, v) \geq 0$*, for all* $e \in E$ *and (possibly negative) node charges* $\phi(v) \in \mathbb{Z}$*, for all* $v \in V$*. The goal is to find a minimum-cost* $F \subseteq E$ *such that* $\sum_{v \in C} \phi(v) \geq 0$ *for each connected component* $C \subseteq V$ *of* (V, F)*.*

A convenient view is that $\phi(v) < 0$ corresponds to a **demand point**, $\phi(v) > 0$ corresponds to a **supply point**, and $\phi(v) = 0$ is a location that is a **Steiner point**. For brevity, throughout this paper when the instance is clear from the context we let $\Phi = \sum_{v \in V} \max\{1, |\phi(v)|\}$. This way, Φ represents the total charge in absolute value but also ensures to count the zero-charge nodes.

It seems the first study of GP2P was actually for a directed variant of the problem and was conducted by Di Gaspero et al.[5] in the course of studying a practical problem related to scheduling shifts. The undirected version itself was formally proposed by Even, Kortsarz, and Slany [6] under the name Infinite Capacity Minimum Edge Cost Flow problem and an $O(\log \Phi)$-approximation was given. The most recent work on GP2P is by Hajiaghayi et al. [10] where they give a 2-approximation when $\phi(V) = 0$ and an $O(\log(\min\{n, \phi(V) + 2\}))$-approximation in general time that is polynomial in Φ. This is achieved by an adaptation of the primal-dual algorithm of [9] for the classic Steiner Forest problem and its generalization (e.g. when we have a proper function). As pointed out in [6], many special cases of GP2P were already well-studied. For example:

- Steiner Forest: We are given terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$ in an edge-weighted graph and the goal is to buy the cheapest set of edges so each pair has both endpoints in the same component. This is modelled by GP2P by letting $\phi(s_i) = 2^i$ and $\phi(t_i) = -2^i$ (and $\phi(v) = 0$ for all non-terminals). Goemans and Williamson give a 2-approximation for Steiner Forest [9].
- k-Minimum Spanning Tree (k-MST): We are given a particular *root* node $r \in V$ and an integer $k \geq 0$. The goal is to find the cheapest tree including r and at least k other nodes. This is modelled by GP2P by letting $\phi(r) = -k$ and $\phi(v) = 1$ for all other $v \in V - \{r\}$. Note, this instance of GP2P has only a single demand point. The history of approximating k-MST is storied, currently the best approximation is a 2-approximation [8, 4].
- Knapsack Covering: We are given n items with sizes $s_1, \ldots, s_n$ and costs $c_1, \ldots, c_n$ plus a demand value D. The goal is to find a minimum-cost set of items with total size at least D. This modelled by GP2P by using a star with center r and leaf nodes $v_1, \ldots, v_n$. The cost of rv_i is c_i and the charges are $\phi(r) = -D$ and $\phi(v_i) = s_i$. An FPTAS for Knapsack Covering is known, e.g. by slightly modifying the standard dynamic programming based FPTAS for standard for standard Knapsack.

Notably, all these cases have constant-factor approximations or better. An open problem is to determine if GP2P has a constant-factor approximation even if we allow pseudo-polynomial running time, i.e. running time that is polynomial in Φ; note for the purposes of getting an $O(1)$-approximation we can assume without loss of generality that the edge costs are bounded by a polynomial in n (see Lemma 19 in Appendix A). Even special cases of GP2P that generalize some of the classic problems stated above are open. For instance, for generalization of k-MST to the situation where we have more than one root node, say $r_1, r_2, \ldots, r_d$, and the goal is to find a minimum cost subgraph where the connected component of each r_i has at least k nodes, it is not known if one can get an $O(1)$-approximation or not. This is one of the cases we study in this paper.

1.1 Our Results

For brevity, we say an instance (G, c, ϕ) of GP2P is **metric** (Metric-GP2P) if G is a complete graph and edge costs satisfy the triangle inequality $c(u, v) \leq c(u, w) + c(w, v)$ for any distinct $u, v, w \in V$. As is usual with single-connectivity network design problems, e.g. Steiner Forest, assuming the graph is metric can be done without loss of generality. The standard proof is found in Lemma 18 in Appendix A. We also say an instance (G, c, ϕ) of GP2P is **graphical** (Graphical-GP2P) if all edge costs are 1 but G is not necessarily complete.

Our results need approximation algorithms for slight generalizations of k-MST.

▶ **Definition 2.** *In* **k-MST with required nodes** *(k-MST-R), instead of a single root node $r \in V$ we are given a subset $R \subseteq V$. The goal is to find the cheapest tree spanning R and at least k other nodes in $V - R$. In* **weighted k-MST-R** *(W-k-MST-R), each vertex $v \in V - R$ is given an integer weight $w_v \geq 0$ and the goal is to find the cheapest tree spanning R plus a subset of nodes in $V - R$ with total weight at least k.*

We note all k-MST approximations we consider in this paper can be readily adapted to the generalization k-MST-R. One can also extend them to W-k-MST-R that would run in pseudo-polynomial time (the simple reduction can be found in Appendix A).

▶ **Observation 3.** *If there is a polynomial-time α-approximation for k-MST-R then there is an α-approximation for W-k-MST-R whose running time is polynomial in the number of nodes and the total weight of all nodes.*

Let n_d be the number of demand points in a given instance, i.e. $|\{v \in V : \phi(v) < 0\}|$. Recall from above that k-MST is a special case of GP2P when there is only one demand point, i.e. $n_d = 1$. We consider the case of GP2P with $n_d \geq 1$. This is akin to generalizing k-MST to multiple roots, each with its own size requirement, but there is one key difference. One might imagine a multiple-root k-MST generalization would want to keep the trees disjoint but in GP2P we can have multiple demand points in a single component as long as the total supply in the component is at least the total demand in the component.

Our main result shows how an α-approximation for W-k-MST-R can be used to give a $(1 + \epsilon) \cdot \alpha$-approximation for GP2P.

▶ **Theorem 4.** *For any $\epsilon > 0$, given an α-approximation algorithm $\mathcal{A}$ for W-k-MST-R, there is a $(1 + \epsilon)\alpha$-approximation for GP2P. This algorithm makes $n^{O(n_d/\epsilon)}$ calls to $\mathcal{A}$. For D-dimensional Euclidean metrics the running time of the GP2P approximation can be improved to $O((16 \cdot n_d/\epsilon)^{D \cdot n_d})$ calls to $\mathcal{A}$.*

We also show how existing algorithms for k-MST (the $(2 + \epsilon)$-approximation for k-MST in [3] for general metrics and the PTAS for k-MST in constant-dimensional Euclidean metrics [2, 11]) can be suitably adapted to given the same approximation guarantees for W-k-MST-R, which combined with the above theorem imply the following:

▶ **Corollary 5.** *For any $\epsilon > 0$, there is a $(2 + \epsilon)$-approximation for GP2P that runs in time is $n^{O(n_d/\epsilon)}$. For D-dimensional Euclidean metrics there is a $(1 + \epsilon)$-approximation for GP2P with running time $O((16 \cdot n_d/\epsilon)^{D \cdot n_d} \cdot \Phi^{O(D/\epsilon)})$*

Note that the $(2 + \epsilon)$-approximation for W-k-MST-R and for GP2P runs in truly polynomial time for any fixed n_d but the $(1 + \epsilon)$-approximation for the Euclidean metrics is polynomial in Φ, hence only pseudo-polynomial, as it is not clear how to get a PTAS for W-k-MST-R in the Euclidean metrics.

We also mention that this would also extend to doubling metrics using the quasi-polynomial time $(1+\epsilon)$-approximation for k-MST in metrics with constant doubling dimension by Talwar [12]. The running time would be quasipolynomial in Φ but still FPT in n_d.

Our next collection of results is for Graphical-GP2P. First, we observe that particular restrictions of GP2P remain essentially as hard to approximate as the general problem.

▶ **Observation 6.** *If there is a polynomial-time α-approximation for instances of GP2P with $\phi(v) \in \{-1, +1\}$ for each $v \in V$, then there is a 2α-approximation for general instances of GP2P with running time being polynomial in Φ.*

▶ **Observation 7.** *If there is a polynomial-time α-approximation for instances of Graphical-GP2P with $\phi(v) \in \{-1, 0, +1\}$ for each $v \in V$, then for any $\epsilon > 0$ there is a $(1+\epsilon) \cdot \alpha$-approximation for general instances of GP2P with running time being polynomial in $1/\epsilon$ and Φ.*

These simple observations are proven in Appendix B. Our main contribution here is that the intersection of these two restrictions of GP2P is still **APX**-hard but does at least admit a simple approximation.

▶ **Theorem 8.** *The restriction of Graphical-GP2P with $\phi(v) = \{-1, +1\}$ for each $v \in V$ is **APX**-hard and admits a polynomial-time 2-approximation.*

Finally, we improve the running time of the logarithmic approximations in [10] by making them run in truly polynomial time (i.e. removing the dependence on Φ).

▶ **Theorem 9.** *There is an $O(\log(\min\{n, \phi(V) + 2\}))$-approximation for GP2P running in polynomial time.*

1.2 Notation

For a graph $G = (V, E)$ and a subset of edges $F \subseteq E$, we let $V(F)$ denote the set of all nodes in V that appear as the endpoint of at least one edge in F and say that F **spans** $V(F)$. We refer to a component of G as a subset of vertices corresponding to a connected component of G. For a given instance $(G = (V, E), c, \phi)$ of GP2P, let $V_s = \{v \in V : \phi(v) > 0\}$ be the supply points, $V_d = \{v \in V : \phi(v) < 0\}$ be the demand points, and $V_0 = \{v \in V : \phi(v) = 0\}$ be the Steiner points. Correspondingly, let n_s, n_d, and n_0 denote their sizes with $n := |V| = n_s + n_d + n_0$. We also use the convention $c(F) := \sum_{e \in F} c(e)$ for $F \subseteq E$ and, similarly, $\phi(C) := \sum_{v \in C} \phi(v)$ for $C \subseteq V$. Note $\phi(V)$ differs from $\Phi := \sum_{v \in V} \max\{1, |\phi(v)|\}$, the former measures the excess of positive charge in the input and the latter measures the absolute value of all charges while ensuring Steiner points are still counted.

1.3 Organization

The algorithms proving Theorem 4 appear in Section 2. Theorem 8 is proven in Section 3 and Section 4 then concludes the paper with the proof of Theorem 9. The proofs of some supporting results as well as Observations 6 and 7 appear in the appendix.

2 Approximations for Constant n_d

The algorithms proving Theorem 4 are obtained by reducing to W-k-MST-R. Recall that in W-k-MST-R, we are given a set of required nodes R along with (integer) node weights $w_v \geq 0$ and the goal is to find a minimum cost tree spanning R such that the total weight of the nodes in the tree is at least a given integer k. The special case of $|R| = 1$ and $w_v = 1$

for all $v \neq r$ is the traditional k-MST problem. Despite its generality, W-k-MST-R can be approximated as well (or nearly as well) as the simpler k-MST problem by adapting the known algorithms.

▶ **Theorem 10.** *The k-MST algorithm described [3] can be adapted to give a polynomial-time $(2+\epsilon)$-approximation for W-k-MST-R for any constant $\epsilon > 0$. Similarly, the PTAS for k-MST in D-dimensional Euclidean metrics described in [11] with running time $O(n^{O(D/\epsilon)})$ can be adapted to give a PTAS for k-MST-R with the same asymptotic running time and a $(1+\epsilon)$-approximation for W-k-MST-R with running time $O((n\Phi)^{O(D/\epsilon)})$.*

These adaptations are discussed in Appendix C. We emphasize both algorithms run in polynomial time (for constant ϵ) even if the number of required nodes R is not bounded by a constant. It is possible that the 2-approximations for k-MST in [8] or [4] could be extended to W-k-MST-R and that the PTAS for Euclidean k-MST in [2] could be similarly extended. We focus on these particular algorithms due to the ease in describing how to extend them.

We present the proof of Theorem 4 in two parts. First for the general metrics and then the improved FPT time for D-dimensional Euclidean metrics.

2.1 Theorem 4: General metrics

Throughout, let OPT denote the cost of an optimum solution. Fix $\epsilon > 0$ to be a sufficiently small constant, $\epsilon < 1/5$ suffices. From Lemma 18 (Appendix A), we may assume G is a complete graph with edge costs satisfying the triangle inequality. For any $r \geq 0$ and any $v \in V$ we let $B(v,r) = \{u \in V : c(u,v) \leq r\}$ be the **ball** of radius r around v.

The intuition for our algorithm is the following. If the optimum solution consisted of only a single tree containing all nodes with negative charge, we could treat it as a W-k-MST-R instance where $R = V_d$ and $k = -\phi(R)$ while setting $w_v = 0$ for $v \in R$ and $w_v = \phi(v)$ for $v \notin R$. However, this may not be the case. To cope, we show how to decompose the instance into disjoint subproblems that have single-tree optimal solutions whose total costs are close to OPT.

To perform the decomposition, we first show that a near-optimum solution to the original instance (G, c, ϕ) exists such that any two trees in this solution notably far apart. Then we guess a small "net" of points in each tree, the union of balls with a particular common radius around each net point covers all trees, but two balls from different trees are disjoint. This allows us to partition the instance into disjoint instances, one per tree in the near optimum solution.

The first step is to show some near-optimum solution has its components being bounded away from each other.

▶ **Lemma 11.** *There is a solution $F' \subseteq E$ with $c(F') \leq (1+\epsilon) \cdot OPT$ such that for any two components C_1, C_2 of (V, F') with $C_1 \cap V_d \neq \emptyset$ and $C_2 \cap V_d \neq \emptyset$ we have $c(u,v) \geq \frac{\epsilon}{n_d} \cdot OPT$ for any $u \in C_1, v \in C_2$.*

Proof. Let F' initially be an optimum solution. While there are two components C_1, C_2 of (V, F') with $C_1 \cap V_d \neq \emptyset$ and $C_2 \cap V_d \neq \emptyset$ such that $c(u,v) < \frac{\epsilon}{n_d} \cdot OPT$ for some $u \in C_1$ and $v \in C_2$, add uv to F'. Each such addition increases the cost of F' by at most $\frac{\epsilon}{n_d} \cdot OPT$ and this procedure will be executed fewer than n_d times since there are initially at most n_d components containing a node in V_d. ◀

Of course, we may also assume, by discarding edges, that F' is a minimal solution meaning $F' - \{e\}$ is not feasible for any $e \in F'$. So F' consists of $m \leq n_d$ vertex-disjoint trees, say $T_1, \ldots, T_m \subseteq F$ that collectively span all of V_d plus some other nodes in $V_0 \cup V_s$. Any other node v not in a tree has $\phi(v) \geq 0$ and forms an isolated component of (V, F').

Next, we identify a *net* of nodes in the trees $T_1, \ldots, T_m$ that will be small enough for our algorithm to guess.

▶ **Lemma 12.** *For any tree T and any $r > 0$ there is a set $N \subseteq V(T)$ with $|N| \leq 1 + c(T)/r$ such that $c(v, N) \leq r$ for each $v \in V(T)$.*

Similar constructions have been considered many times before, one example is in the $(2+\epsilon)$-approximation for k-MST [3]. We include a proof for completeness.

Proof. We prove this by induction on $|V(T)|$. Root T at an arbitrary node w. The base case is when $c(w, v) \leq r$ for each $v \in V(T)$. If so, we simply let $N = \{w\}$.

Otherwise, for two $u, v \in V(T)$ let $c_T(u, v)$ be the cost of the unique $u - v$ path in T. Pick any $v \in V(T)$ that has maximum value $c_T(w, v)$. Note $c_T(w, v) \geq c(w, v) > r$. Now let u be the furthest node along the $v - w$ path such that $c_T(v, u) \leq r$. Since $c(v, u) \leq c_T(v, u) \leq r$, then $u \neq w$. Let $p(u)$ be the parent of u in T. By our choice of u we also have $c_T(v, p(u)) > r$.

Let T_u denote the subtree of T rooted at u. For any $v' \in V(T_u)$ we have $c(v', u) \leq c_T(v', u) \leq c_T(v, u) \leq r$. Let T' be the tree obtained by removing the subtree rooted at u from T. This removes all edges of the $v - p(u)$ path, so $c(T') \leq c(T) - r$. By induction, we can find a set $N' \subseteq V(T')$ with $|N'| \leq 1 + c(T')/r \leq c(T)/r$ such that $c(w', N) \leq r$ for each $w' \in V(N')$.

Finally, set $N = \{u\} \cup N'$ and note $|N| = 1 + |N'| \leq 1 + c(T)/r$. Every $u' \in V(T)$ is now within distance r from some node in N, as required. ◀

The entire algorithm is summarized in Algorithm 1. The high-level idea is that it guesses a value ν very close to OPT and then guesses the "nets" $N_1, \ldots, N_m$ from Lemma 12 applied to each tree of the near-optimum solution F' using $r = \frac{\epsilon}{4 \cdot n_d} \cdot \nu$. Below, we show for the proper guess of $N_1, \ldots, N_m$ that the sets V_i obtained by the union of balls $B(v, r)$ for $v \in V_i$ are disjoint. This instance of GP2P then naturally decomposes into disjoint instances of W-k-MST-R. Supporting results demonstrating the performance of our algorithm are found below.

Algorithm 1 GP2P approximation.

if $F^0 = \{e \in E : c(e) = 0\}$ is feasible **then**
  return F^0
for each integer b such that $\nu := (1+\epsilon)^b \in [\min_e c(e), \sum_e c(e)]$ **do**
  for each $1 \leq m \leq n_d$ and each m-tuple $N_1, \ldots, N_m \subseteq V$ with $\sum_{i=1}^m |N_i| \leq 5 \cdot n_d/\epsilon$ **do**
    let $r := \frac{\epsilon}{4n_d} \cdot \nu$ and $V_i := \cup_{v \in N_i} B(v, r)$ for each $1 \leq i \leq m$
    if $V_i \cap V_j \neq \emptyset$ for distinct i, j or if $V_d \not\subseteq \cup_{i=1}^m V_i$ **then**
      continue to the next iteration
    for each $1 \leq i \leq m$ **do**
      construct the W-k-MST-R instance on the subgraph $G[V_i]$ with $R = V_d \cap V_i$ and $k = -\phi(R)$
      let T_i' be the tree obtained by running an α-approximation on this instance
    Record $\cup_{i=1}^m T_i'$ as a candidate solution.
return the cheapest candidate solution found over all iterations.

We first discuss the running time. The number of iterations of the outer loop is logarithmic in the ratio $c(E)/\min_e c(e)$, which is polynomial in the number of bits used to represent the costs in the instance. There are clearly only n_d possible values for m and the number of

m-tuples satisfying the stated bounds is at most $n^{O(5 \cdot n_d/\epsilon)}$. So when n_d is regarded as a constant, the total number of iterations is polynomial in the input size and, thus, the entire algorithm makes a polynomial number of calls to a k-MST-R approximation on instances with at most n nodes and, otherwise, runs in polynomial time.

Towards the performance guarantee, we show for the "correct" guess of values in the loops the algorithm will perform well.

▶ **Lemma 13.** *Let $\nu \in [OPT, (1+\epsilon) \cdot OPT]$ and $T_1, \ldots, T_m$ be the trees in the near-optimum solution F' from Lemma 11. For each $1 \leq i \leq m$, let N_i be the set identified by Lemma 12 when applied to T_i using $r = \frac{\epsilon}{4 \cdot n_d} \cdot \nu$. Also let $V_i = \cup_{v \in N_i} B(v, r)$ for each $1 \leq i \leq m$.*

Then (a) $\sum_{i=1}^{m} |N_i| \leq 5 \cdot n_d/\epsilon$, (b) $V_i \cap V_j = \emptyset$ for distinct i, j, and (c) $V_d \subseteq \cup_{i=1}^{m} V_i$.

Proof. For (a), we have

$$\sum_i |N_i| \leq m + \frac{c(F')}{r} \leq n_d + \frac{(1+\epsilon) \cdot OPT}{OPT} \cdot \frac{4n_d}{\epsilon} \leq \frac{5}{\epsilon} \cdot n_d.$$

For (b), if, say, $w \in V_i \cap V_j$ for some distinct i, j then there would be some $u \in N_i$ and $v \in N_j$ such that

$$c(u,v) \leq c(u,w) + c(w,v) \leq r + r = \frac{\epsilon}{2 \cdot n_d} \cdot \nu \leq \frac{\epsilon(1+\epsilon)}{n_d} \cdot OPT \leq \frac{\epsilon}{n_d} \cdot OPT.$$

But this contradicts Lemma 11, which showed $c(u,v) > \frac{\epsilon}{n_d} \cdot OPT$. Finally, (c) follows because the balls $B(v,r)$ for $v \in N_i$ collectively cover all of $V(T_i)$ and each node of V_d lies on some T_i. ◀

To finish the analysis, consider the iteration of the algorithm for the particular setting of ν and $T_1, \ldots, T_m$ described in Lemma 13. With these values, the algorithm proceeds to run the k-MST-R approximations. In the instance corresponding to V_i, we know T_i itself is a feasible solution so the returned tree T_i' satisfies $c(T_i') \leq \alpha \cdot c(T_i)$. Thus, the candidate GP2P solution found in this iteration has cost $\sum_{i=1}^{m} c(T_i') \leq \alpha \cdot c(F') \leq \alpha \cdot (1+\epsilon) \cdot OPT$.

2.2 Theorem 4: Euclidean metrics

We simply describe how to modify Algorithm 1. Clearly, we can use a $(1+\epsilon)$-approximation for W-k-MST-R in Euclidean metrics to make Algorithm 1 a $(1+\epsilon)$-approximation in Euclidean metrics. The pseudo-polynomial running time in the statement of Theorem 4 comes from the fact that we only know how to adapt k-MST PTASes to k-MST-R and then rely on Observation 3 to get a pseudo-polynomial time W-k-MST-R $(1+\epsilon)$-approximation. One small comment is that even though the distances are not necessarily rational numbers, the number of iterations of the outer loop is still polynomial in the number of bits used to describe the locations of the points in V.

To improve the running time to be FPT in n_d, we change how the nets are guessed. Let D be the dimension of the metric, recall that D is assumed to be a constant. For brevity, let $\delta := \epsilon/(16n_d)$. Note $4\delta \cdot \nu$ is the value r from Algorithm 1. The idea of the improvement is the following. For simplicy let's consider the case of $D = 2$. If one considers a square of side length L, the number of disjoint balls of radius $\epsilon \cdot L/n_d$ that can be placed inside that square is $O((n_d/\epsilon)^2)$. This simple packing argument can be used to bound the number of guessed points for the nets N to be bounded by $O((n_d/\epsilon)^D)$.

For each guess ν in the outer loop, we first let N be any $\delta \cdot \nu$-net of V. That is, every $v \in V$ has $c(v, N) \leq \delta \cdot \nu$ yet $c(u, v) > \delta \cdot \nu$ for any $u, v \in N$. Such a set N can be constructed by greedily adding points while maintaining the property that $c(u, v) > \delta \cdot \nu$ until no more points can be added. For each $v \in V$, let $\tau(v)$ be its closest point in N. So $\tau(v) = v$ for $v \in N$ and, otherwise, we at least know $c(v, \tau(v)) \leq \delta \cdot \nu$.

Now let $N_1, \ldots, N_m$ be the sets identified by applying Lemma 12 using $r = \delta \cdot \nu$ and let $N'_i = \{\tau(v) : v \in N_i\}$. For $u \in N'_i$ and $v \in N'_j$ for $i \neq j$ we still have that $B(u, \delta) \cap B(v, \delta) = \emptyset$. That is, suppose otherwise and let $u' \in N_i$ be such that $\tau(u') = u$ and $v' \in N_j$ be such that $\tau(v') = v$. Then when $\nu \in [OPT, (1 + \epsilon) \cdot OPT]$ we have

$$c(u', v') \leq c(u, u') + c(u', v') + c(v', u) \leq \delta \cdot \nu + 2\delta \cdot \nu + \delta \cdot \nu \leq \frac{\epsilon}{4n_d} \cdot \tau \leq \frac{\epsilon}{n_d} \cdot OPT$$

which contradicts Lemma 11 and the fact that u' and v' lie in different trees in F'.

So it suffices to guess $N'_1, \ldots, N'_m$ in the algorithm. But now we leverage packing property of Euclidean metrics to help reduce the number of guesses to a constant depending on D, n_d and ϵ.

▶ **Lemma 14.** *For each $v \in V$, $|B(v, \nu) \cap N|$ is bounded by $O((4/\delta)^D)$.*

Proof. The Euclidean balls of radius $\delta/2 \cdot \nu$ about points in $B(v, \nu) \cap N$ are disjoint by construction of N and are completely contained in the Euclidean ball of radius $(1+\delta/2) \cdot \nu \leq 2 \cdot \nu$ about v. The volume a D-dimensional ball with radius r is within an absolute constant factor of $f(R) := \frac{1}{\sqrt{D}} \cdot \left(\frac{2\pi e}{D}\right)^{D/2} \cdot R^D$. Therefore, $|B(v, \nu) \cap N|$ is at most a constant factor times $f(2\nu)/f(\delta/2 \cdot \nu) = (4/\delta)^D$. ◀

The steps for guessing $N'_1, \ldots, N'_m$ are to first try all ways to partition V_d into m nonempty groups, a coarse upper bound on the number of such choices is $n_d^{n_d}$. For each such partition, let $v_1, \ldots, v_m \in V_d$ be any particular representatives from the m parts. We try all tuples $N'_1, \ldots, N'_m$ where each $N'_i \subseteq B(v_i, \nu) \cap N$ such that $\sum_i |N'_i| \leq 17/\epsilon \cdot n_d$ (as opposed to $5/\epsilon \cdot n_d$ as in Lemma 13 since the radius δ is smaller). For each such tuple that passes the other requirements of Lemma 13, we partition the instance into disjoint Euclidean k-MST-R instances and approximate these with a PTAS. A coarse upper bound on the number of such tuples $N'_1, \ldots, N'_m$ is $O((4/\delta)^{D \cdot n_d})$.

3 Approximation Algorithms and Hardness for Graphical-GP2P with ± 1 Charges

Observations 6 and 7 show GP2P is not much easier to approximate if we assume either that $\phi(v) \in \{-1, +1\}$ for each $v \in V$ or that the instance is graphical and $\phi(v) \in \{-1, 0, +1\}$ for each $v \in V$. We begin this section by showing the common intersection of these two special cases does admit a 2-approximation. After this, we complete the proof of Theorem 8 by showing such GP2P instances remain **APX**-hard.

3.1 Graphical Instances with Unit Charges

Let $(G = (V, E), c, \phi)$ be an instance of Graphical-GP2P where $\phi(v) \in \{-1, +1\}$ for each $v \in V$. As usual, let OPT denote the optimum solution cost which, in this case, is just measuring the minimum size feasible solution $F \subseteq E$.

▶ **Theorem 15.** *Let $F \subseteq E$ be any* **minimal** *feasible solution (i.e. $F - \{e\}$ is not feasible for any $e \in F$). Then $|F| \leq 2 \cdot OPT$.*

A 2-approximation is then straightforward as one could simply start with F being any spanning tree and then iteratively try to drop an edge from F while retaining feasibility until no such drop is possible.

Proof. Recall $V_d = \{v \in V : \phi(v) = -1\}$ and $n_d = |V_d|$. So $V_s = V - V_d$ as no charge is zero in this case. We claim $n_d \leq OPT$ and that any minimal solution has size at most $2 \cdot n_d$. Thus, any minimal solution F satisfies $|F| \leq 2 \cdot n_d \leq 2 \cdot OPT$, as required.

For the first bound, let F^* be an optimum solution and C any connected component in (V, F^*) that contains at least one node in V_d. If C has, say, $n_d^C \geq 1$ nodes in V_d then C must contain at least n_d^C nodes in V_s as well. That is, $|C| \geq 2n_d^C$ so F^* contains at least $2n_d^C - 1 \geq n_d^C$ edges in component C. Summing over all components that contain at least one node in V_d, we see $|F^*| \geq n_d$.

Now let F be any minimal feasible solution. Let C be any connected component of (G, F) and let F_C be the edges of F in component C. If $C \cap V_d = \emptyset$ then minimality of F means C is a single node v with $\phi(v) = +1$ (i.e. it has no edges). If $C \cap V_d \neq \emptyset$, we claim $\phi(C) = 0$. If so, then $|C \cap V_d| = |C \cap V_s|$ so $|F_C| < 2 \cdot |C \cap V_d|$ as F_C is a tree (by minimality). Summing over all components would complete the claim that $|F| < 2 \cdot n_d$.

To see $\phi(C) = 0$, again recall F_C is a tree. Further, for each $e \in F_C$ we have that deleting e would produce a component with negative charge but it could not be that both components have negative charge since $\phi(C) \geq 0$. Consider the orientation of edges that directs each edge e toward the side that would have negative charge if e was deleted. Since (C, F_C) is a tree, the orientation of edges produces a directed acyclic graph. Let r be any source node in this DAG.

View the tree (C, F_C) as being rooted at r and say r has m children. Let $C_1, \ldots, C_m \subseteq C$ be such that C_i are the nodes in the subtree under the i'th child of r. Since $C \cap V_D \neq \emptyset$ and $\phi(C) \geq 0$, we know C has at least two nodes so $m \geq 1$.

By the orientation of edges, we have $\phi(C_i) \leq -1$ for each $1 \leq i \leq m$. Therefore

$$\phi(C) = \phi(r) + \sum_{i=1}^{m} \phi(C_i) \leq \phi(r) - m \leq \phi(r) - 1 \leq 0$$

which, by feasibility of F, means $\phi(C) = 0$. ◀

It may be possible to get a better-than-2 approximation through a more involved approach. For example, notice in our proof $n_d \leq OPT$ is only tight if all components C of the optimum solution F^* with $C \cap V_d \neq \emptyset$ had $|C| = 2$ (i.e. F^* is a matching). So if the optimal solution F^* has at least, say, $0.01 \cdot n_d$ nodes of V_d in components of size greater than 2 then any minimal solution would be a 1.99-approximation. Otherwise, nearly all nodes of V_d are in components that consist of a single edge. In this case, a maximum-size matching between V_d and V_s would then create components of charge 0 that collectively include most nodes of V_d. One can even show there is a way to "alternate" the matching so that a greedy algorithm yields a good approximation (i.e. by alternating so our matching edges largely agree with the optimum matching edges), but efficiently finding such an alternation seems challenging.

One would reasonably wonder if instances of GP2P with unit-cost edges and ± 1 charges are actually hard. Indeed, we conclude this section by showing **APX**-hardness.

▶ **Theorem 16.** *The restriction of* Graphical-GP2P *to instances with unit edge costs and* ± 1 *charges is* **APX***-hard.*

Proof. We reduce from the Minimum Vertex Cover Problem in simple, cubic graphs which is known to be **APX**-hard [1]. That is, for some constants $0 \leq \beta < \alpha \leq 1$ it is **NP**-hard to distinguish if a cubic graph on n vertices has a vertex cover of size at most $\beta \cdot n$ or if all vertex covers have size exceeding $\alpha \cdot n$.

So let $G = (V, E)$ be an n-vertex cubic graph with $m = 3n/2$ edges. We obtain a graph $H = (V', E')$ and set the charges of $v \in V'$ as follows.

Initially let $H = G$ and set $\phi(v) = +1$ for every $v \in V$. Then subdivide each edge e with a single vertex u_e with $\phi(u_e) = -1$. Then for each v we add new vertices $v^+, v^-, v_0, v_1, v_2, v_3$ and edges $\{v_0v, vv^+, v^+v^-, v^+v_1, v^+v_2, v^+v_3\}$. Here, v, v^+, v_1, v_2, v_3 all have positive charge and v_0, v^- have negative charge. See Figure 1 for an illustration of this process.

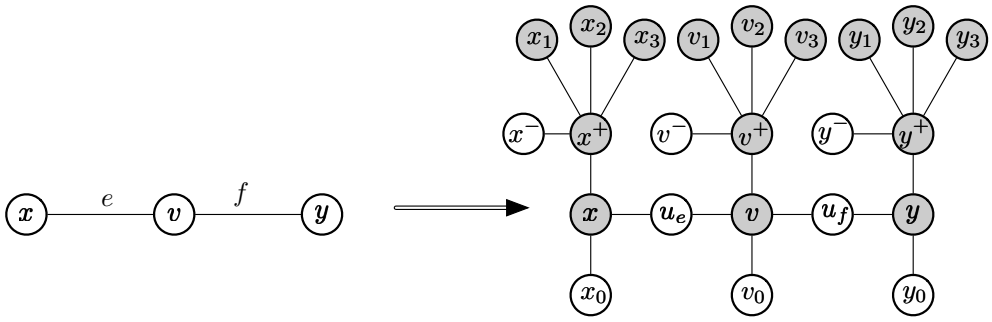


Figure 1 An illustration of the reduction to a vertex v and two if its neighbours. In the right picture, positively-charged nodes are shown in grey and negatively-charged nodes are shown in white. Intuitively, selecting vv^+ corresponds to using v in the vertex cover because it now permits using the positive charge nodes v_1, v_2, v_3 to satisfy the demands of some negative charge nodes u_e.

We claim G has a vertex cover of size k if and only the Graphical-GP2P instance given by H and ϕ has a solution of size $2n + 2m + k = 5n + k$. So if G has a vertex cover of size at most βn then there is a GP2P solution of size at most $(5 + \beta) \cdot n$ and if all vertex covers in G have more than $\alpha \cdot n$ then all solutions to this GP2P instance have size more than $(5 + \alpha) \cdot n$. This shows there is no $\frac{5+\alpha}{5+\beta}$-approximation for GP2P with unit edge costs and charges ± 1 unless $\mathbf{P} = \mathbf{NP}$.

To see the claim, first let $C \subseteq V$ be a vertex cover of G with size k. To get a corresponding Graphical-GP2P solution, buy the following edges:

- For each $v \in C$, purchase vv^+.
- For each $v \in V$, purchase v_0v and v^+v^-.
- For each $e \in E$, let v be any endpoint of e in C. Purchase u_ev and any edge of the form v_iv^+ that was not purchased by another edge this way. Since G is cubic, this is always possible.

In total, this purchases $k + 2n + 2m = k + 5n$ edges. This can be seen to be a feasible solution by matching each negative-charged vertex with a positively-charged vertex in its component in a one-to-one fashion. Such a mapping is immediate from the construction: each v_0 can be matched with v, each v^- can be matched with v^+, and each u_e can be matched with the corresponding v_i purchased in the description above.

For the converse, we first claim there is a well-structured optimal solution.

▷ Claim 17. There is an optimal solution F such that: (a) for each $v \in V$ both v_0v and v^+v^- are in F, (b) F has exactly m edges of the form v_iv^+ where $v \in V$ and $i = 1, 2, 3$, (c) each $e \in E$ has $vv^+ \in F$ for at least one endpoint v of e, and (d) for each edge $e = vw$ exactly one of $u_ev \in F$ or $u_ew \in F$.

If so, then $C = \{v : vv^+ \in F\}$ is a vertex cover of G and $|F| = |C| + 2n + 2m = |C| + 5n$, as required to complete the proof.

Proof of Claim 17. Let F be an optimal solution. First observe we must have $v_0 v \in F$ and $v^+v^- \in F$ for every $v \in V$ otherwise some negatively-charged node would be isolated. So in each component we must have the number of nodes of the form u_e for various $e \in E$ is at most the number of nodes of the form v_i for various $v \in V, i = 1, 2, 3$. By optimality of F and the fact each v_i is a pendant node, we have that these counts are in fact equal in each component.

Let τ be any minimum-cost pairing of nodes $u_e, e \in E$ with nodes of the form v_i lying in the same component as u_e. Here, the cost of pairing two nodes is the length of their shortest path using only edges in F. For each $e \in E$, let P_e denote the corresponding shortest path from u_e to $\tau(u_e)$. Finally, let $v(e)$ be the first vertex after u_e along P_e, notice $v(e)$ corresponds to an endpoint of e in the original cubic graph G.

Now consider an alternative pairing of the u_e vertices with the v_i vertices (which do not necessarily need to lie in the same component as u_e, for now). For each u_e, pair it with any vertex of the form $v(e)_i$ for some $i = 1, 2, 3$ that has not been paired before. Such a pairing is possible since G is cubic. Let F' be obtained by modifying F in the following way. From F, first delete all edges of the form $v^+v_i \in F$ then add all edges of the form v^+v_i where v_i is paired under the new pairing.

This does not change the size of F. To ensure it is feasible, do the following. For each $e \in E$ such that P was not initially paired with one of $v(e)_1, v(e)_2, v(e)_3$, it must have been that $v(e)u_{e'}$ was the second edge along P for some $e \neq e'$ and that $v(e') \neq v(e)$ (otherwise P_e and $P_{e'}$ both use $v(e)u_{e'}$ but in opposite directions, meaning we can uncross the paths to get a cheaper pairing than τ). So we remove $v(e)u_{e'}$ and add $v(e)v(e)^+$ (if it was not already there). Doing so for all $e \in E$ will ensure it is now feasible since each u_e can now reach the vertex it is paired with while not increasing the size of F because an edge of the form $v(e)v(e)^+$ is added only after an edge of the form $v(e)u_{e'}$ is removed. This also ensures all properties required by the claim now hold. ◁

◀

4 A Polynomial-Time Logarithmic Approximation

We show that a slight variation of algorithm of [10] yields an $O(\log n)$-approximation in truly polynomial time. The algorithm in [10] begins by observing, using standard metric embeddings [7], that it suffices to given an $O(1)$-approximation if the graph is a tree which, after rooting at some vertex, has exactly two children per node. Then they present an exact algorithm using dynamic programming where one index of the DP table considers values up to Φ. Roughly speaking, for every vertex v and every $-\Phi \leq p \leq \Phi$ they consider the subtree T_v rooted at v and compute the cheapest subset of edges in the tree such that the component with v has charge p and every other component in the subtree has nonnegative charge.

We simply point out that we can flip the roles of charges and costs in the DP table. First, we use the reduction in Lemma 19 (using, say, $\epsilon = 1/2$) to instances where each edge has its cost being bounded by a polynomial in n. It is easy to verify this reduction produces a tree if the input graph was a tree. Let (T, c, ϕ) be the resulting instance of GP2P, i.e. T is a tree and each edge cost is a positive integer bounded by a polynomial in n.

Root T at an arbitrary node r and for each node v let T_v be the subtree under v. Our dynamic programming table is the following: for each node v and each $0 \leq c \leq \sum_{e \in E} c_e$ let $f[v, c]$ be the maximum p such that in the subtree T_v, it is possible to purchase edges with total cost at most c such that the component with v has charge at least p and every other component has nonnegative charge. Given these values, the optimum solution cost is then the minimum c such that $f[r, c] \geq 0$.

To compute the $f[v, c]$ values:

- If v is a leaf node then $f[v, c] = \phi(v)$.
- Otherwise, say u, w are the two children of v. Intuitively, we try all subsets of $\{uv, wv\}$ to delete and try all ways to split the remaining budget between the subproblems and keep the best solution found overall. That is, we try all ways to purchase a subset $S \subseteq \{uv, wv\}$ such that $c(S) \leq c$ and all $0 \leq c_u, c_w$ such that $c_u + c_w + c(S) = c$ and such that $f[u, c_u] \geq 0$ if $uv \notin S$ and $f[w, c_w] \geq 0$ if $uw \notin S$ (i.e. if we do not purchase the corresponding parent edge, then the budget in the subproblem better be able to buy a feasible solution in the subtree).

  Then $f[v, c]$ is the maximum of the following expression over such S, c_u, c_w:

$$\phi(v) + \mathbb{I}[uv \in S] \cdot f[u, c_u] + \mathbb{I}[uw \in S] \cdot f[u, c_w]$$

where $\mathbb{I}[\cdot]$ is the $\{0, 1\}$-indicator for the logical expression enclosed by the brackets. The number of distinct subproblems is polynomial in n since the edge costs are integers at most n and there are at most $4 \cdot (c+1)$ different ways to select (S, c_u, c_v) in a subproblem, the algorithm runs in polynomial time. In particular, if C denotes the total edge cost in the tree then the number of distinct subproblems is $O(C \cdot n)$ and processing each entry $f[v, c]$ takes $O(c)$ time (including $O(c)$ recursive calls) so the total running time is $O(C^2 \cdot n)$. Finally, if one only permits recursive calls to subproblems $f[v, c]$ where c is at most the total edge cost in the subtree T_v and the loops over the split $c_u + c_v = c$ only iterate over values where c_u and c_v are at most the total edge cost of their respective subtrees, the running time is improved to $O(C^2)$.

This polynomial-time approximation for GP2P in trees can be used in a black-box fashion to improve the running time of the $O(\log(\min\{n, \phi(V)+2\}))$ in [10] to run in true polynomial time.

References

1. Paola Alimonti and Viggo Kann. Some apx-completeness results for cubic graphs. *Theoretical Computer Science*, 237(1):123–134, 2000. doi:10.1016/S0304-3975(98)00158-3.
2. Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, September 1998. doi:10.1145/290179.290180.
3. Sanjeev Arora and George Karakostas. A $2 + \epsilon$ approximation algorithm for the k-mst problem. *Math. Program.*, 107(3):491–504, July 2006.
4. Emmett Breen, Renee Mirka, Zichen Wang, and David P. Williamson. Revisiting garg's 2-approximation algorithm for the k-mst problem in graphs. In *2023 Symposium on Simplicity in Algorithms, SOSA 2023*, pages 56–68. SIAM, 2023. doi:10.1137/1.9781611977585.CH6.
5. Luca Di Gaspero, Johannes Gärtner, Guy Kortsarz, Nysret Musliu, Andrea Schaerf, and Wolfgang Slany. The minimum shift design problem. *Annals of operations research*, 155:79–105, 2007. doi:10.1007/S10479-007-0221-1.
6. Guy Even, Guy Kortsarz, and Wolfgang Slany. On network design problems: fixed cost flows and the covering steiner problem. *ACM Trans. Algorithms*, 1(1):74–101, July 2005. doi:10.1145/1077464.1077470.

7 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004. Special Issue on STOC 2003. doi:10.1016/j.jcss.2004.04.011.

8 Naveen Garg. Saving an epsilon: a 2-approximation for the k-mst problem in graphs. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 396–402, 2005. doi:10.1145/1060590.1060650.

9 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995. doi:10.1137/S0097539793242618.

10 Mohammadtaghi Hajiaghayi, Rohit Khandekar, Guy Kortsarz, and Zeev Nutov. On fixed cost k-flow problems. *Theor. Comp. Sys.*, 58(1):4–18, January 2016. doi:10.1007/s00224-014-9572-6.

11 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.

12 Kunal Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 281–290. ACM, 2004. doi:10.1145/1007352.1007399.

A Standard Reductions

Proof of Observation 3. Each $v \in V - R$ with $w_v \geq 1$ is replaced with $(n+1) \cdot w_v$ colocated copies and each $v \in V - R$ with $w_v = 0$ is left as well. In the new k-MST-R instance, use $k' := (n+1) \cdot k$. Consider any tree T (say, one obtained using a k-MST-R approximation) in the new instance that spans R and at least k' other nodes. By adding 0-cost edges if necessary, we may assume that T contains all colocated copies of any node it spans.

Since at most n nodes have $w_v = 0$, then T spans at least $(n+1) \cdot k - n > (n+1) \cdot (k-1)$ nodes from groups of colocated copies of original nodes. Since each group has a size that is a multiple of $n+1$, then T spans at least $(n+1) \cdot k$ such nodes, i.e. viewing T as a tree in the original graph after contracting colocated copies of nodes it spans yields a feasible solution. ◀

▶ **Lemma 18.** *Let $(G = (V, E), c, \phi)$ be an instance of GP2P and let $G' = (V, E')$ be the complete graph over V with metric edge costs $c'(uv)$ given by the minimum-cost $u - v$ path. Any feasible solution to the GP2P instance (G, c, ϕ) can be mapped to a feasible solution to the Metric-GP2P instance (G', c', ϕ) with no greater cost and vice-versa.*

Proof. For each $e \in E$ we have $c'(e) \leq c(e)$ since e is one possible path between its endpoints. So for any feasible solution F to (G, c, ϕ) we have $c'(F) \leq c(F)$, as required. Conversely, for any feasible solution $F' \subseteq E'$ to (G', c', ϕ) if we let F be the union of all shortest $u - v$ paths in G for each $uv \in F'$ then F then $c(F) \leq c'(F)$. Also, two nodes that were in the same connected component in (V, F') still lie in the same connected component of (V, F). That is, each connected component C of (V, F) is the union of one or more connected components $\{C'_1, \ldots, C'_a\}$ in (V, F') so $\phi(C) = \sum_{i=1}^{a} \phi(C') \geq 0$. ◀

▶ **Lemma 19.** *For any constant $\epsilon > 0$, if there is an α-approximation for instances of GP2P where every edge cost $c(e)$ is an integer at most $n^2/\epsilon + 1$, then there is a $(1+\epsilon) \cdot \alpha$-approximation for general instances of GP2P.*

Proof. Let $(G = (V, E), c, \phi)$ be a general instance of GP2P with optimum solution $F^* \subseteq E$ with cost OPT. By contracting 0-cost edges (which does not change the optimal solution value), we assume $c(e) > 0$ for each $e \in E$.

First, compute smallest value ν such that every connected component C in the graph G_ν with edges $\{e \in E : c(e) \leq \nu\}$ has $\phi(C) \geq 0$. We claim $\nu \leq OPT \leq n \cdot \nu$: the first bound is because any feasible solution must use at least one edge of cost $\geq \nu$ by our choice of ν and the other is because a spanning forest of G_ν is a feasible solution using fewer than n edges each of cost at most ν.

Let $E' = \{e \in E : c(e) \leq n \cdot \nu\}$; we have $F^* \subseteq E'$ since $OPT \leq n \cdot \nu$. Define new edge costs $c'(e) := \lceil n \cdot c(e)/(\epsilon \cdot \nu) \rceil$ for each $e \in E'$. Notice $c'(e)$ is a positive integer (since $c(e) > 0$) and $c'(e) \leq n \cdot c(e)/(\epsilon \cdot \nu) + 1 \leq n^2/\epsilon + 1$ by construction of E'.

Let OPT' denote the optimum solution cost for the GP2P instance $((V, E'), c', \phi)$. We have $OPT' \leq c'(F^*) \leq \sum_{e \in F^*}(n \cdot c(e)/(\epsilon \cdot \nu) + 1) = n \cdot OPT/(\epsilon \cdot \nu) + n$. Therefore, running an α-approximation on this new instance finds a set of edges $F' \subseteq E'$ with $c'(F') \leq \alpha \cdot (n \cdot OPT/(\epsilon \cdot \nu) + n)$. Since $c(e) \leq \epsilon \cdot \nu \cdot c'(e)/n$ for every $e \in E'$,

$$c(F') \leq \epsilon \cdot \nu \cdot c'(F')/n \leq \alpha \cdot (OPT + \epsilon \cdot \nu) \leq (1 + \epsilon) \cdot \alpha \cdot OPT.$$ ◀

B Restricted Instances of GP2P

Proof of Observation 6. Let $(G = (V, E), c, \phi)$ be an instance of GP2P. As noted in the introduction, we may assume this is an instance of Metric-GP2P. Finally, let $V' = V - V_0$ (i.e. the nodes with non-zero $\phi(v)$) and G' be the subgraph of G induced by V'. Notice the restriction of (G', c, ϕ) to H is an instance of Metric-GP2P.

Let $F^* \subseteq E$ be an optimal solution for the original Metric-GP2P instance. For each tree T in the forest F^*, let C_T be the tour obtained by doubling the edges of T and shortcutting the resulting Eulerian tour past nodes in V_0. In this way, C_T spans all nodes in $V - V_0$ that are spanned by T and $c(C_T) \leq 2 \cdot c(T)$. Thus, the optimal solution cost in the restriction to H is at most twice the optimal solution cost for (G, c, ϕ).

To complete the reduction, for each vertex v of H if $\phi(v) \geq 1$ then replace v with $\phi(v)$ collocated copies each having charge 1 and if $\phi(v) \leq -1$ then replace v with $-\phi(v)$ collocated copies each having charge -1. Note this steps takes pseudopolynomial time. ◀

Proof of Observation 7. Consider any constant $0 < \epsilon' \leq 1$. Let $(G = (V, E), c, \phi)$ be an instance of GP2P. First we consider some preprocessing. If the 0-cost edges form a feasible solution, then it must be optimal so there is nothing more to do. Otherwise, apply Lemma 19 with $\epsilon := \epsilon'/3$ and let $(G' = (V, E'), c', \phi)$ be the resulting graph with positive integer edges costs being bounded by a polynomial in n.

Let K be a positive integer to be specified later. Form $(G'' = (V'', E''), c'', \phi'')$ by performing the following operations to G'.

- Subdivide each $e \in E'$ into a path of length $K \cdot c(e)$ of unit cost edges. Each new vertex v' in the subdivision has $\phi''(v') = 0$.
- For each $v \in V'$ with $\phi(v) \geq 1$, append a path P_v to v using $\phi(v) - 1$ new vertices and edges: each edge has cost 1 and each vertex on P_v, including v itself, has $\phi''(v) = 1$. Note, the other endpoint of P_v is a pendant.
- Similarly each $v \in V'$ with $\phi(v) \leq -1$, append a path P_v to v using $-\phi(v) - 1$ new vertices and edges: each edge has cost -1 and each vertex on P_v, including v itself, has $\phi''(v) = -1$.
- Each $v \in V'$ with $\phi(v) = 0$ also has $\phi''(v) = 0$.
- Finally, each edge e of this new graph G'' has $c''(e) = 1$.

Observe (G'', c'', ϕ'') is an instance of Graphical-GP2P with $\phi(v) \in \{-1, 0, +1\}$ for each $v \in V''$.

Note a solution $F' \subseteq E'$ naturally maps to a solution in the final instance with cost at most $\Phi + K \cdot c'(F')$ by including all pendant paths P_v and all subdivided paths corresponding to edges in F'. Conversely, consider any solution $F'' \subseteq E''$. Let $F' \subseteq E$ be the set of edges of G' such that their entire subdivision is included in F''. It is easy to verify that F' is a feasible solution with cost at most $c''(F'')/K$.

Let OPT' be the optimal solution value for instance (G', c', ϕ) and let F'' be the result of using α-approximation on (G'', c'', ϕ''). By the preceding discussion, this yields a feasible solution F' to (G', c', ϕ) with

$$c'(F') \leq c''(F'')/K \leq \alpha \cdot (\Phi + K \cdot OPT')/K = \alpha \cdot OPT' + \alpha \cdot \Phi/K.$$

By setting $K = \lceil 3 \cdot \Phi/\epsilon' \rceil$ and noting that $1 \leq OPT$ as edge costs are positive integers in (G', c', ϕ), this is at most $(1 + \epsilon'/3) \cdot \alpha \cdot OPT'$.

Finally, by accounting for the application of Lemma 19 at the start of this proof we see that we would have an approximation for the original instance (G, c, ϕ) with guarantee $(1 + \epsilon'/3)^2 \cdot \alpha \leq (1 + \epsilon') \cdot \alpha$. ◀

C Adapting k-MST Approximations

We only sketch how the algorithms can be adapted. We refer the reader to the respective papers for their details.

▶ **Lemma 20** (Slight adaptation of Arora and Karakostas [3]). *There is a polynomial-time $(2 + \epsilon)$-Approximation for W-k-MST-R.*

Proof. The algorithm in [3] explicitly guesses a subset of vertices that appear in the optimum solution and builds that into the LP relaxation they write. So it can already handle the situation where we have a larger set of required nodes R. The only thing to mention is how it can be extended to handle node weights $w_v \geq 0$ for $v \notin R$. We can assume each node with weight w_v is implicitly a collection of w_v many nodes connected using 0-cost edges in a star fashion. The algorithm of [3] first guesses an additional set of size $O(1/\epsilon)$ of vertices of OPT to be required. The next step of the algorithm is to run a "primal dual" like algorithm to find a tree T. This step works without modification in our setting. ◀

The final step in [3] is to modify T appropriately. That is, the manner in which T was constructed actually provides us with two options: include some subset of nodes or not. One choice would result in fewer than k non-required nodes being spanned and the other would result in at least k non-required nodes being spanned. In [3], it is mentioned how to pick the correct number of nodes contiguously from this "optional" portion so that grafting them in to the remaining portion of T yields a feasible solution with the required number of nodes. The grafting only costs $O(\epsilon \cdot OPT)$ due to the guess of the net at the start of the algorithm. This can also be done in polynomial time if one implicitly maintains a 0-cost tree spanning each group of colocated points.

▶ **Lemma 21** (Slight adaptation of Arora [2]). *There is a PTAS for k-MST-R*

We first comment that a similar adaptation could be made to Mitchell's PTAS [11]. We chose this one because it was slightly easier to describe. We also emphasize that this adaptation is only for *unweighted* k-MST-R. Combining this with Observation 3 yields a pseudo-polynomial time approximation scheme for W-k-MST-R.

Proof. The PTAS for k-MST in constant-dimensional Euclidean plane uses a dynamic programming routine through a quadtree dissection of the plane (an in higher dimensions a 2^D-tree dissection in D-dimensional spaces). The DP table entries for each square, roughly speaking, describe the interface of the optimal solution across the boundary of that square through "portals" and also include the guess for how many nodes should be covered within the square. If there are required nodes R, we can use the same DP table and simply insist that the subproblem's solution also span any nodes of R in the square. The base cases are trivially extended to this setting and the combination of subproblems (i.e. the recurrence) for a non-base case is identical to before. ◀

Linear Layouts of Graphs with Priority Queues

Emilio Di Giacomo
University of Perugia, Italy

Walter Didimo
University of Perugia, Italy

Henry Förster
Technical University of Munich, Heilbronn, Germany

Torsten Ueckerdt
Karlsruhe Institute of Technology, Germany

Johannes Zink
Technical University of Munich, Heilbronn, Germany

Abstract

A *linear layout* of a graph consists of a linear ordering of its vertices and a partition of its edges into *pages* such that the edges assigned to the same page obey some constraint. The two most prominent and widely studied types of linear layouts are *stack* and *queue layouts*, in which any two edges assigned to the same page are forbidden to *cross* and *nest*, respectively. The names of these two layouts derive from the fact that, when parsing the graph according to the linear vertex ordering, the edges in a single page can be stored using a single stack or queue, respectively. Recently, the concepts of stack and queue layouts have been extended by using a double-ended queue or a restricted-input queue for storing the edges of a page. We extend this line of study to *edge-weighted* graphs by introducing *priority queue layouts*, that is, the edges on each page are stored in a priority queue whose keys are the edge weights. First, we show that there are edge-weighted graphs that require a linear number of priority queues. Second, we characterize the graphs that admit a priority queue layout with a single queue, regardless of the edge-weight function, and we provide an efficient recognition algorithm. Third, we show that the number of priority queues required independently of the edge-weight function is bounded by the pathwidth of the graph, but can be arbitrarily large already for graphs of treewidth two. Finally, we prove that determining the minimum number of priority queues is NP-complete if the linear ordering of the vertices is fixed.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Graph theory

Keywords and phrases linear layouts, recognition and characterization, priority queue layouts

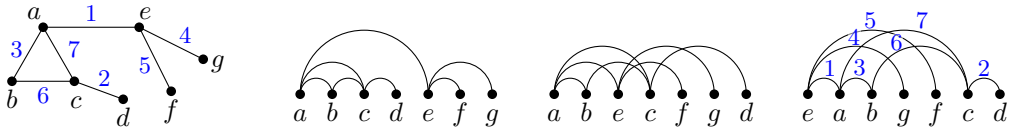
Digital Object Identifier 10.4230/LIPIcs.WADS.2025.29

Related Version *Full Version*: https://arxiv.org/abs/2506.23943 [20]

Funding Research of E. Di Giacomo and W. Didimo partially supported by: (i) Italian MUR PRIN Proj. 2022TS4Y3N – "EXPAND: scalable algorithms for EXPloratory Analyses of heterogeneous and dynamic Networked Data"; (ii) Italian MUR PRIN Proj. 2022ME9Z78 – "NextGRAAL: Next-generation algorithms for constrained GRAph visuALization"; (iii) Italian MUR PON Proj. ARS01 00540 – "RASTA: Realtà Aumentata e Story-Telling Automatizzato per la valorizzazione di Beni Culturali ed Itinerari".

Acknowledgements This work was initiated at the Annual Workshop on Graph and Network Visualization (GNV 2023), Chania, Greece, June 2023.

© Emilio Di Giacomo, Walter Didimo, Henry Förster, Torsten Ueckerdt, and Johannes Zink;
licensed under Creative Commons License CC-BY 4.0



(a) Edge-weighted graph G. **(b)** Stack layout of G. **(c)** Queue layout of G. **(d)** PQ-layout of G.

Figure 1 (a)–(c) Examples of stack and queue layouts on one page for the same graph G ignoring edge weights. (d) A PQ-layout on one page of G (edge weights are in blue).

1 Introduction

Stack layouts were introduced by Bernhart and Kainen in 1979 [13] with the motivation to study a "quite natural" edge decomposition technique. A stack layout is a *linear layout*, that is, the vertices are positioned on a line ℓ with the edges being embedded on several half-planes, called *pages*, delimited by ℓ. In a stack layout, edges embedded on the same page are forbidden to cross, a rule that contributed to their original designation as *book embeddings*. An example of stack layout on one page is depicted in Figure 1b. The minimum number of pages required to obtain a stack layout of a graph is called the *stack number*. After their introduction, stack layouts attracted notable interest in the field of theoretical computer science due to their capacity to model certain aspects in several domains, including VLSI design [17, 18], fault-tolerant multiprocessing [18, 41], RNA folding [28], and traffic-light control [33]. This led, in particular, to a series of papers investigating upper bounds on the stack number of planar graphs [16, 29, 32] and culminating at an upper bound of four [43]. Bekos et al. [11] and Yannakakis [44] independently proved that this upper bound is tight.

The name stack layout derives from the fact that the edges assigned to the same page can be stored using a stack as follows [17]. Assume that ℓ is a horizontal line and consider the left-to-right ordering $v_1, \ldots, v_n$ of the vertices. Now, consider a left-to-right sweep of ℓ and the set E_p of edges assigned to a page p. Initially, the stack storing E_p is empty. At vertex v_i, all edges that have v_i as their right endpoint are popped from the stack. Since all vertices are at the same side of the half-plane delimited by ℓ, the edges ending at v_i are necessarily the last ones that have started prior to encountering v_i, so they are precisely the edges on top of the stack. Afterwards, edges starting at v_i are pushed onto the stack. (Note that the order of the pushes is determined by the right endpoints of the corresponding edges.)

Based on this latter interpretation of stack layouts, Heath, Leighton and Rosenberg [30, 31] proposed to study *queue layouts* of graphs where the edges occurring on the same page of the linear layout can be represented by a queue. More precisely, edges ending at v_i are first dequeued from a queue, after which edges beginning at v_i are enqueued during the left-to-right sweep of ℓ. Thus, edges on the same page are allowed to cross but forbidden to properly nest. An example of a queue layout on one page is shown in Figure 1c. Surprisingly, these quite non-planar representations still found many practical applications, including scheduling [14], VLSI [35], and, a decade after their inception, 3D crossing-free graph drawing [21, 24]. Similarly to stack layouts, one is interested in minimizing the number of pages required to obtain a queue layout, which is known as the *queue number*. However, in contrast to the stack number, the queue number of planar graphs remained elusive for several decades during which only sublinear (but non-constant) upper bounds have been found [1, 5, 6, 22]. The first significant difference between planar and non-planar graphs was only demonstrated in 2019 when it was shown that bounded degree planar graphs have bounded queue number [9], whereas non-planar bounded degree graphs were already known to not have this property [42]. Recently, Dujmović et al. [23] proved that the queue number of planar graphs is bounded by a constant. Minor improvements have since been made [10, 26].

Another interesting recent direction in investigating linear layouts is to expand beyond stack and queue layouts by using other data structures for partitioning edges. For instance, *mixed linear layouts* allow the usage of both stack and queue pages [2, 19, 38], whereas *deque layouts* use *double-ended queues* which allow to insert and pull edges from the head and the tail [3, 4, 12], and *rique layouts* use *restricted-input queues* which are double-ended queues where insertions are allowed to occur only at the head [8].

Our contribution. The wide range of applications of linear layouts and the fact that they have been restricted so far to unweighted graphs, naturally motivate extensions of the notion of linear layout to edge-weighted graphs. For example, in scheduling applications, the edges of a graph can represent processes, each with an associated priority; each vertex v of the graph enforces that all processes corresponding to edges incident to v begin simultaneously; among the running processes, those with higher priority must be completed first. Edge weights can also model constraints in other classical applications of linear layouts, such as logistics networks modelled as switchyard networks [34, 37], where adjacent edges represent containers that must be pushed or popped simultaneously across different storage locations. The weight of an edge corresponds to the weight of its container, and it may be necessary to ensure that containers stacked on top of others are lighter.

We introduce a model called *priority queue layout*, or simply *PQ-layout*. This model utilizes *priority queues* for the storage of edges assigned to the same page, where the edge weights correspond to the priorities (i.e., keys) for the data structure. As for stack and queue layouts, in a PQ-layout all vertices of the graph lie on a horizontal line ℓ, and the edges are assigned to different priority queues according to a left-to-right sweep of ℓ. When an edge e is dequeued, it must have the minimum priority among the edges stored in the same priority queue as e. See Figure 1d for an example, and refer to Section 2 for a formal definition of our model. Note that, unlike stack and queue layouts, our model allows edges in the same page to cross or properly nest, provided that they have suitable weights.

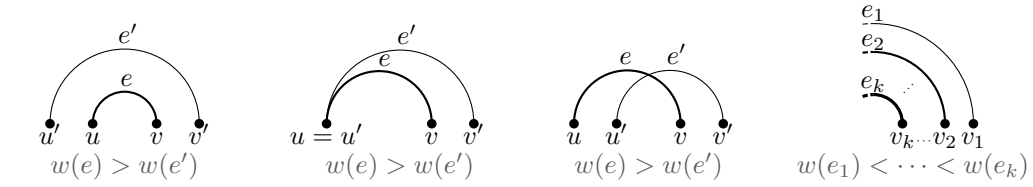
Following a current focus of research on linear layouts, we mainly study the *priority queue number* of edge-weighted graphs, that is, the number of pages required by their PQ-layouts:

(i) We provide non-trivial upper and lower bounds for the priority queue number of complete graphs (Section 3). (ii) We characterize the graphs that have priority queue number 1, regardless of the edge-weight function (Section 4). (iii) We investigate the priority queue number of graphs with bounded pathwidth and bounded treewidth (Section 5). (iv) We prove that deciding whether an edge-weighted graph has priority queue number k (for a non-fixed integer k) is NP-complete if the linear ordering of the vertices is fixed (Section 6).

We remark that, as far as we know, there is only one previous study in the literature about using linear layouts of edge-weighted graphs [7]; however, this study uses edge weights to further restrict stack layouts. For space restrictions, proofs of statements marked with a (clickable) ($\star$) are omitted or sketched. We provide the missing proofs in the full version [20].

2 PQ-Layouts

Let $\langle G, w\rangle$ be an *edge-weighted* graph, that is, $G = (V, E)$ is an undirected graph and $w\colon E \to \mathbb{R}$ is an *edge-weight function*. A *priority queue layout*, or simply a *PQ-layout*, Γ of $\langle G, w\rangle$ consists of a linear (left-to-right) ordering $v_1 \prec \cdots \prec v_n$ of the vertices of G and a partitioning of the edges of G into k sets $\mathcal{P}_1, \ldots, \mathcal{P}_k$, where each set is called a *page* and is associated with a priority queue. For each page $\mathcal{P}_i$ with $i \in \{1, \ldots, k\}$, the following must



(a) Forbidden Nesting. **(b)** Forbidden Pseudo-Nesting. **(c)** Forbidden Crossing. **(d)** k-Inversion.

Figure 2 (a)–(c) Arrangements in a PQ-layout that correspond to the Forbidden Configuration F.1. (d) A k-inversion, a forbidden arrangement in linear layouts with priority queue number $k-1$.

hold: when traversing the linear ordering $\prec$ (from left to right) and performing, at each encountered vertex $v \in V$, the operations O.1 and O.2 (in this order), condition C.1 must always hold.

O.1 Each edge $e \in \mathcal{P}_i$ having v as its right endpoint is pulled from the priority queue. We do not count e as active any more.

O.2 Each edge $e \in \mathcal{P}_i$ having v as its left endpoint is inserted into the priority queue of $\mathcal{P}_i$ with key $w(e)$. We say that e becomes *active*. Two edges being active at the same time (even though they may have become active in different steps) are called *co-active*.

C.1 There is no pair of co-active edges e and e' such that e has v as its right endpoint, e' has a distinct right endpoint, and $w(e) > w(e')$. In other words, among the active edges of $\mathcal{P}_i$, the edges having v as their right endpoint have the smallest edge weights.

PQ-layouts can be equivalently defined by the absence of the forbidden configuration F.1.

F.1 For some $i \in \{1, \ldots, k\}$, there are two edges $e = uv$ and $e' = u'v'$ such that $e, e' \in \mathcal{P}_i$, $w(e) > w(e')$, and $u, u' \prec v \prec v'$.

The order of u and u' is irrelevant for F.1. For completeness, the resulting three arrangements of the involved vertices are illustrated in Figures 2a–2c. Note that the forbidden pseudo-nesting in Figure 2b requires $u = u'$. In contrast, the symmetric case where $v = v'$ is never forbidden. This is a direct consequence of the asymmetry between the insert and pull operations of priority queues. Thus, unlike for stack and queue layouts, the reverse vertex ordering does not necessarily provide a PQ-layout.

We refer to the linear ordering $\prec$ of vertices also by the ordering along the *spine*. For a PQ-layout Γ of any graph, let $\mathrm{pqn}(\Gamma)$ denote the number of priority queues in Γ. Now, the *priority queue number* $\mathrm{pqn}(G, w)$ of $\langle G, w\rangle$ is the minimum integer $k \geq 0$ for which there exists a PQ-layout Γ of $\langle G, w\rangle$ with $\mathrm{pqn}(\Gamma) = k$.

For any given graph G, there always exists an edge-weight function $w\colon E \to \mathbb{R}$ such that $\mathrm{pqn}(G, w) = 1$. It is enough to assign the same weights to all edges but it is also possible to use distinct edge weights. Namely, define any linear ordering of the vertices $v_1 \prec \cdots \prec v_n$ and visit the vertices in this ordering. When a vertex v with $v \neq v_1$ is visited, assign consecutive (non-negative) integer weights to all edges for which v is the right endpoint such that the weights monotonically increase. Hence, any graph with any given linear ordering of its vertices can have priority queue number 1 if we can choose the edge-weight function. However, the edge-weight function may be provided as part of the input. Thus, it is interesting to study bounds on the priority queue number of families of graphs that hold independently of the edge-weight function. To this end, for a graph $G = (V, E)$ without any prescribed edge-weight function, the *priority queue number* $\mathrm{pqn}(G)$ is the minimum integer $k \geq 0$ such that for every possible edge-weight function $w\colon E \to \mathbb{R}$ we have $\mathrm{pqn}(G, w) \leq k$.

Some of our proofs utilize configurations where every pair of edges must be assigned to distinct priority queues. Let $\langle G, w\rangle$ be an edge-weighted graph, let $\prec$ be a vertex ordering of G, and let k be an integer such that $k \geq 1$. We define a *k-inversion* to be a sequence $(e_1, \ldots, e_k)$ of k edges in G with pairwise distinct right endpoints $v_1, \ldots, v_k$, respectively, such that (i) $v_k \prec \cdots \prec v_1$, where v_k and v_1 are called the *left* and *right end*, respectively, (ii) the left endpoint of each of $e_1, \ldots, e_k$ appears to the left of v_k in $\prec$, and (iii) $w(e_1) < \cdots < w(e_k)$. See Figure 2d for an illustration. Any two edges in a k-inversion form a forbidden configuration and thus must be assigned to distinct priority queues. In other words, if, for edge weights w of G, every vertex ordering of $\langle G, w\rangle$ contains a k-inversion, then $\mathrm{pqn}(G, w) \geq k$ and hence $\mathrm{pqn}(G) \geq k$.

3 Priority Queue Number of Complete Graphs

In this section, we concentrate on complete graphs and complete bipartite graphs. We start with a simple upper bound for the complete graph K_n on n vertices.

▶ **Observation 1.** $\mathrm{pqn}(K_n) \leq n - 1$.

Proof. Choose an arbitrary ordering of the vertices $v_1 \prec \cdots \prec v_n$ and define the sets (priority queues) $\mathcal{P}_1, \ldots, \mathcal{P}_{n-1}$. For each $j \in \{2, \ldots, n\}$, assign all edges whose right endpoint is v_j to $\mathcal{P}_{j-1}$. Clearly, no forbidden configuration occurs, independent of the edge-weight function. ◀

Next, we consider bipartite graphs, that is, the graphs whose vertex sets are the union of two disjoint independent sets A and B. In the literature on linear layouts, *separated* linear layouts, where vertices of A precede the vertices of B or vice versa, have been considered for bipartite graphs [25]. The *separated priority queue number* $\mathrm{spqn}(G)$ of a bipartite graph G is the minimum k such that, for any edge-weight function, there is a PQ-layout Γ that is a separated linear layout and $\mathrm{pqn}(\Gamma) = k$. We can easily determine the separated priority queue number of the complete bipartite graph $K_{n,n}$. At the end of this section, we will use the separated priority queue number to give a lower bound on the priority queue number of complete and complete bipartite graphs.

▶ **Theorem 2.** $\mathrm{spqn}(K_{m,n}) = \min\{m, n\}$.

Proof. First observe that $\mathrm{spqn}(K_{m,n}) \leq \min\{m, n\}$: place the vertices of the larger set, say A, before the vertices of the smaller set, say B. Then, independent of the edge weight function, for each vertex $b \in B$, all edges ending at b can be assigned to a separate page.

To show $\mathrm{spqn}(K_{m,n}) \geq \min\{m, n\}$, assume w.l.o.g. that the vertices in A precede the vertices in B. Consider an edge weight function where, at every vertex $b \in B$, each edge weight in $\{1, 2, \ldots, \min\{m, n\}\}$ occurs at least once among b's incident edges. In any bipartite PQ-layout Γ of $K_{m,n}$, denote the vertices of B by $b_1, b_2, \ldots$ in the order they appear along the spine. In Γ, b_1 is incident to an edge with weight $\min\{m, n\}$, b_2 is incident to an edge with weight $\min\{m, n\} - 1$, etc., $b_{\min\{m,n\}}$ is incident to an edge of weight 1. This is a $\min\{m, n\}$-inversion requiring at least $\min\{m, n\}$ pages. ◀

Theorem 2 also provides an upper bound for the (non-separated) priority queue number of $K_{n,n}$. Next, we give a corresponding linear lower bound showing that $\mathrm{pqn}(K_{n,n}) \in \Theta(n)$.

▶ **Theorem 3 (⋆).** $\mathrm{pqn}(K_{n,n}) \geq \left\lceil \frac{3-\sqrt{5}}{4} n \right\rceil \approx 0.191n$.

Proof Sketch. For $K_{n,n}$, we define an edge-weight function w that has, at each vertex, for every $i \in \{1, \ldots, n\}$, exactly one incident edge of weight i.[1] To this end, we partition the edge set into n perfect matchings $M_1, \ldots, M_n$, and, for each edge $e \in M_i$, we set $w(e) = i$.

For any PQ-layout Γ of $\langle K_{n,n}, w\rangle$, we can find a sublayout Γ' that is a *bipartite* PQ-layout of $\langle K_{\frac{n}{2},\frac{n}{2}}, w\rangle$ – either take the first $\frac{n}{2}$ vertices of A and the last $\frac{n}{2}$ vertices of B or vice versa.[2]

Note that we cannot directly apply Theorem 2 to Γ' because we have already fixed an edge-weight function. Instead, we analyze the possible distribution of the edge weights incident to vertices in Γ' by a representation as a black-and-white grid;[3] see the the full version [20] for an illustration and a more extensive description. Our grid has $\frac{n}{2}$ columns that represent the vertices of the latter partition (in order from left to right as they appear in Γ') and it has n rows that represent the edge weights $\{1, \ldots, n\}$ (in order from bottom to top). A grid cell in column i and row j is colored black if the i-th vertex is incident to an edge of weight j, and it is colored white otherwise. Observe that, in this grid, a *strictly monotonically decreasing path* of black cells having length k is equivalent to a k-inversion in Γ'. We can show that there always exists such a path of length $\left\lceil \frac{3-\sqrt{5}}{4} n \right\rceil$: the candidates for being the first black cell of the path are those who do not have another black cell in their top left region of the grid. We find these cells, color them white and iteratively repeat this process to find the candidates for being the second, third, etc. black cell of the path. We can prove that we need at least $\left\lceil \frac{3-\sqrt{5}}{4} n \right\rceil$ iterations until the grid is white. ◀

Since K_n contains $K_{\lfloor \frac{n}{2} \rfloor, \lfloor \frac{n}{2} \rfloor}$ as a subgraph, Theorem 3 immediately implies a linear lower bound on $\mathrm{pqn}(K_n)$:

▶ **Corollary 4.** $\mathrm{pqn}(K_n) \geq \left\lfloor \frac{3-\sqrt{5}}{8} n \right\rfloor \approx 0.0955n.$

4 Characterizing Graphs with Priority Queue Number 1

We study which graphs have priority queue number 1. We will provide a characterization of the graphs with priority queue number 1 in terms of forbidden minors (Section 4.4). We first show that the family of graphs with priority queue number 1 is minor-closed (Section 4.1). Then, we describe families of graphs with priority queue number 1 (Section 4.2) and a set of forbidden minors (Section 4.3). The characterization of Section 4.4 follows by proving that every graph that does not contain any of the forbidden minors of Section 4.3 falls in one of the families of Section 4.2. This characterization leads to an efficient recognition algorithm.

4.1 PQ-Layouts of Minors

A *minor* G' of a graph $G = (V, E)$ is obtained from G by a series of edge contractions and by removing vertices and edges. Formally, for an edge $e = uv$ in G, the *contraction of edge* e yields the graph G/e, obtained by replacing vertices u and v by a single vertex x_{uv} with incident edges $\{x_{uv}y \mid uy \in E \text{ or } vy \in E\}$. Many important graph classes (e.g., planar graphs) are *minor-closed*, that is, if we take a minor of a graph lying in such a class, the resulting graph belongs to this class as well. Every minor-closed graph class is defined by a finite set of forbidden minors and can be recognized efficiently [39, 40].

[1] In contrast to the proof of Theorem 2, we cannot arbitrarily assign weights to edges incident to each vertex $b \in B$ because this may give twice the same edge weight at a vertex of partition A.

[2] For simplicity, we assume that n is even.

[3] A similar, although not identical, representation is used by Alam et al. [36] to prove the mixed page number of $K_{n,n}$.

▶ **Lemma 5** (⋆). *Let $G = (V, E)$ be a graph with $\mathrm{pqn}(G) = 1$ and let $G' = G/e_0$ be obtained from G by contracting an edge $e_0 = uv \in E$. Then, $\mathrm{pqn}(G') = 1$.*

Proof Sketch. We prove $\mathrm{pqn}(G', w') = 1$ for any given edge-weight function w'. We construct a PQ-layout Γ of $\langle G, w\rangle$ using one priority queue where $w(e) = w'(e)$ except for $w(e_0) = W < \min_{e \in E \setminus \{e_0\}} w(e)$. Assuming $u \prec v$ in Γ, we construct a PQ-layout Γ' of $\langle G', w'\rangle$: We place vertex x_{uv} obtained from contracting e_0 at the position of u in Γ. Vertices preceding u or succeeding v in Γ retain their positions in Γ', and vertices between u and v in Γ are sorted according to the weight of their edges shared with x_{uv}. See the full version [20] for details. ◀

▶ **Corollary 6.** *The family of graphs with priority queue number 1 is minor-closed.*

4.2 Graphs with Priority Queue Number 1

We begin by showing how to construct PQ-layouts of trees:

▶ **Lemma 7.** *Let $\langle T = (V, E), w\rangle$ be an n-vertex edge-weighted tree with root $r \in V$. Then, a PQ-layout Γ of $\langle T, w\rangle$ with $\mathrm{pqn}(\Gamma) = 1$, where for each vertex $v \neq r$ its parent $p(v)$ precedes v in the linear ordering $\prec$ along the spine, can be constructed in $\mathcal{O}(n \log n)$ time.*

Proof. We describe an algorithm that constructs a PQ-layout Γ of T; see Figure 3 for an example. For each vertex $v \in V \setminus \{r\}$, we define its weight $w(v)$ as $w(v) = w(vp(v))$. As an initial layout, we first place r as the leftmost vertex and the children of r in increasing order of their weights to the right of r. During our algorithm, we keep track of the *last expanded vertex* $v^\star$, which is the rightmost vertex whose children (if any) have already been placed. Note that initially $v^\star = r$. We now maintain the following invariants:

T.1 For each vertex v with $v \preceq v^\star$, all children have already been placed.

T.2 For each vertex v with $v^\star \prec v$, no child has been placed yet. In addition, $p(v) \preceq v^\star$.

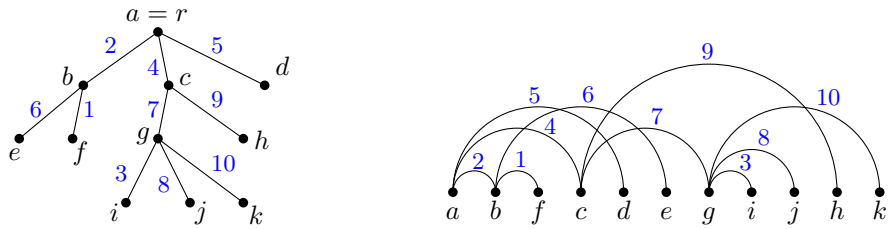
T.3 Let S denote the set of already placed vertices succeeding $v^\star$ in $\prec$. Then, the vertices in S occur in $\prec$ in increasing order of their weights.

T.4 The already constructed PQ-layout has priority queue number 1.

It is easy to see that the invariants hold for the initial layout. We now iteratively consider the vertex v' immediately succeeding $v^\star$ in $\prec$ and perform the following operations. For each child c of v', we place c to the right of v' such that $S \cup \{c\} \setminus \{v'\}$ is ordered by weight. This can be easily done since by **T.3**, S (and thus also $S \setminus \{v'\}$) is already ordered by weight, i.e., there is a unique position for each c and **T.3** holds again after v' has become the new $v^\star$. Next, consider edge $v'c$. By **T.4**, the only way that the resulting linear layout has not priority queue number 1 is if $v'c$ is included in a forbidden configuration. Consider an edge uv with $u \prec v$ that is co-active with $v'c$. Clearly, $v \in S$ and by **T.2**, we have that $u \preceq v'$ and $u = p(v)$. By **T.3**, it follows that c and v are sorted by their weights, which are equal to $w(v'c)$ and $w(uv)$, thus, uv and $v'c$ form no forbidden configuration, i.e., **T.4** is maintained.

After placing all children of v', we set $v^\star = v'$. Clearly, **T.1** is maintained as we placed all children of v' whereas by induction the children of the vertices preceding v' have already been placed. Moreover, **T.2** is maintained as we only placed all children of v' to the right of v' whereas by induction the children of other vertices of S are not placed yet.

For every vertex, we need to find its position depending on the weight. This can be done in $\mathcal{O}(\log n)$ time if we maintain a suitable search data structure. The rest of the insertion can be done in constant time. Overall, this results in a running time in $\mathcal{O}(n \log n)$. ◀



(a) A weighted rooted tree T. **(b)** A PQ-layout Γ of T.

Figure 3 A PQ-layout of a weighted rooted tree computed with the algorithm in Lemma 7.

Next, we construct a specific layout of caterpillars, which are a special subclass of trees. Namely, a *caterpillar* C consists of an *underlying path* $P(C)$ and additional leaves, each having exactly one neighbor on $P(C)$. Be aware that, for the same caterpillar C, there are several choices for $P(C)$, and in particular $P(C)$ might end in a degree-1 vertex of C.

▶ **Lemma 8.** *Let $\langle C, w\rangle$ be an n-vertex edge-weighted caterpillar with underlying path $P(C)$ and let r be one of the two degree-1 vertices in $P(C)$. Then, a PQ-layout Γ of $\langle C, w\rangle$ with $\mathrm{pqn}(\Gamma) = 1$ where r is the rightmost vertex can be constructed in $\mathcal{O}(n)$ time.*

Proof. We label $P(C)$ as $p_1 p_2 \ldots p_k = r$ and arrange the vertices of $P(C)$ in the order $p_1 \prec p_2 \prec \ldots \prec r$. Then for $i \in \{1, \ldots, k\}$, we place all leaves at p_i between p_{i-1} and p_i in an arbitrary order (note that for p_1, we place its leaves before p_1). It is easy to see that the resulting layout contains no forbidden configuration and the statement follows. ◀

We now shift our attention to graphs containing cycles. First, we consider cycles alone.

▶ **Lemma 9.** *Let $\langle C = (V, E), w\rangle$ be an n-vertex edge-weighted cycle, and let v be any given vertex of V. Then, a PQ-layout Γ of $\langle C, w\rangle$ with $\mathrm{pqn}(\Gamma) = 1$ where v is the leftmost vertex can be constructed in $\mathcal{O}(n)$ time.*

Proof. We maintain two *candidate* vertices a and b to be placed next. We call the neighbor of a (b, resp.) that has already been placed *anchor* vertex a' (b', resp.). Initially, a and b are the two neighbors of the leftmost vertex v and $v = a' = b'$. We iteratively append the candidate vertex whose edge to its anchor vertex has the smallest weight to the right of the linear order (if both weights are equal, we take any of both candidates). If $w(a'a) \leq w(b'b)$, we append a while we do not yet place b. After appending a, we set $a' = a$ and the other neighbor of a becomes the new candidate vertex a. We proceed symmetrically with b if $w(a'a) > w(b'b)$. When $a = b$, we place the last remaining vertex a as the rightmost vertex.

Clearly, every vertex is placed. We show that no forbidden configuration occurs. Suppose that there are two co-active edges uv (where $u \prec v$) and xy (where $x \prec y$) with $w(uv) > w(xy)$ and $v \prec y$. In the course of our algorithm, after x and u have been placed and before v and y have been placed, the candidate vertices were v and y whose anchor vertices were u and x. There could not have been a different anchor or candidate vertex because C is a cycle. Then, however, we would not have placed v first because $w(uv) > w(xy)$ – a contradiction.

Note that the running time of the initialization, termination and each iterative step is constant. Therefore, the overall running time is in $\mathcal{O}(n)$. ◀

We then consider the family of legged cycles. Namely, a *legged cycle* L consists of a single *underlying cycle* $C(L)$ and additional leaves, each having exactly one neighbor on $C(L)$.

▶ **Lemma 10** (⋆). *Let $\langle L, w\rangle$ be an n-vertex edge-weighted legged cycle with underlying cycle $C(L)$. Then, a PQ-layout Γ of $\langle L, w\rangle$ with $\mathrm{pqn}(\Gamma) = 1$ can be constructed in $\mathcal{O}(n \log n)$ time.*

Proof Sketch. We remove the heaviest edge $e^\star$ of $C(L)$ and all leaves where the weight of the incident edge is greater than $w(e^\star)$. We lay out the remaining graph using Lemma 8 such that the endpoints of $e^\star$ are the first and the last vertex. We reinsert $e^\star$ and, in increasing order of the weights of their incident edges, we add the initially removed leaves to the right. ◀

Next, we show that we may attach one caterpillar (Lemma 11) or two caterpillars to a single quadrangle (Lemma 12) or a single triangle (Corollary 13).

▶ **Lemma 11.** *Let $\langle G, w\rangle$ be an n-vertex edge-weighted graph consisting of a cycle O and a caterpillar C with underlying path $P(C)$ such that $V(O) \cap V(C) = \{r\}$ is a degree-1 vertex of $P(C)$. Then, a PQ-layout Γ of $\langle G, w\rangle$ with $\mathrm{pqn}(\Gamma) = 1$ can be constructed in $\mathcal{O}(n)$ time.*

Proof. We first draw C and O separately using Lemmas 8 and 9, respectively. Then, we identify the two occurrences of r in C and O with each other, which yields our PQ-layout Γ of G. This does not result in a forbidden configuration since r is the rightmost vertex of C and the leftmost vertex of O and, hence, there is no edge spanning over r. ◀

▶ **Lemma 12** (⋆). *Let $\langle G, w\rangle$ be an n-vertex edge-weighted graph consisting of:*
- *A 4-cycle $\square = abcd$.*
- *A caterpillar C_a with underyling path $P(C_a)$ starting at vertex $a \in \square$.*
- *A caterpillar C_c with underyling path $P(C_c)$ starting at vertex $c \in \square$.*

Then, a PQ-layout Γ of $\langle G, w\rangle$ with $\mathrm{pqn}(\Gamma) = 1$ can be constructed in $\mathcal{O}(n \log n)$.

Proof Sketch. We arrange the cycle such that only the heaviest edge is co-active with the other three edges. We use Lemma 7 for laying out the one caterpillar with designated leftmost vertex and we use Lemma 8 for laying out the other caterpillar with designated rightmost vertex. In the full version [20], we argue that we can combine these three subgraphs, potentially adding some leaves later on, while still using only one priority queue. ◀

By Lemma 5, if G is a minor of a graph as described in Lemma 12, then also $\mathrm{pqn}(G) = 1$.

▶ **Corollary 13.** *If G consists of a triangle $\triangle = abc$ and two caterpillars C_a and C_b with underlying paths starting at vertex $a \in \triangle$ and $b \in \triangle$, respectively, then $\mathrm{pqn}(G) = 1$.*

Finally, we consider graphs that contain more than one cycle. If we exclude disconnected graphs, there exist exactly two such graphs with priority queue number 1, the complete bipartite graph $K_{2,3}$ and the graph obtained from K_4 by removing an edge. We will later see that these are the only ones.

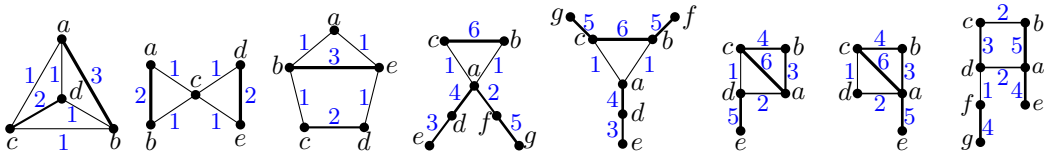
▶ **Lemma 14** (⋆). $\mathrm{pqn}(K_{2,3}) = 1$.

In addition, by Lemma 5, we obtain the following from Lemma 14.

▶ **Corollary 15.** $\mathrm{pqn}(K_4 - e) = 1$ *where $K_4 - e$ is obtained from K_4 by removing one edge.*

4.3 Forbidden Minors for Graphs with Priority Queue Number 1

Each of the graphs $\mathcal{F}_1, \ldots, \mathcal{F}_8$ shown in Figure 4 has priority queue number at least two. The numbers at the edges specify edge-weight functions for which one priority queue does not suffice. In the full version [20], we prove the following observation:



(a) $\mathcal{F}_1$ (b) $\mathcal{F}_2$ (c) $\mathcal{F}_3$ (d) $\mathcal{F}_4$ (e) $\mathcal{F}_5$ (f) $\mathcal{F}_6$ (g) $\mathcal{F}_7$ (h) $\mathcal{F}_8$

Figure 4 Forbidden minors for graphs with priority queue number 1. The graphs with the provided edge weights do not admit a linear layout with priority queue number 1. The thickness of an edge indicates the weight of that edge.

▶ **Lemma 16** (⋆). *In any PQ-layout* Γ *of an edge-weighted cycle* $\langle C, w\rangle$ *with* $\mathrm{pqn}(\Gamma) = 1$*, the last vertex on the spine is incident to an edge of maximum weight in* $\langle C, w\rangle$*.*

We now use Lemma 16 to prove the main result of this subsection:

▶ **Lemma 17** (⋆). *For each graph* $\mathcal{F} \in \{\mathcal{F}_1, \ldots, \mathcal{F}_8\}$ *(see Figure 4),* $\mathrm{pqn}(\mathcal{F}) > 1$ *holds.*

Proof of Cases $\mathcal{F}_1$–$\mathcal{F}_3$. To show $\mathrm{pqn}(\mathcal{F}) > 1$, it suffices to show $\mathrm{pqn}(\mathcal{F}, w) > 1$ for some edge weighting w. We claim that the edge weightings in Figure 4 require more than one priority queue. We consider each of the eight graphs individually and suppose for a contradiction that there is a PQ-layout Γ with $\mathrm{pqn}(\Gamma) = 1$. We consider $\mathcal{F}_1$–$\mathcal{F}_3$ here and $\mathcal{F}_4$–$\mathcal{F}_8$ in the full version [20].

$\mathcal{F}_1$: Consider the triangle $\triangle abd$. By Lemma 16, a or b is the last vertex of $\triangle abd$ (on the spine) since ab is its heaviest edge. In the triangle $\triangle bcd$, cd is the heaviest edge, which implies that c or d is the last vertex of $\triangle bcd$. Hence, neither b nor d can be the (overall) rightmost vertex. Since the heaviest edge of the 4-cycle $abcd$ is again ab, a is the rightmost vertex. In the triangle $\triangle acd$, cd is the heaviest edge, which implies that a cannot be the rightmost vertex; a contradiction.

$\mathcal{F}_2$: By Lemma 16 and symmetry, we may assume that a and d are the last vertices of $\triangle abc$ and $\triangle cde$, respectively. Moreover, assume by symmetry that $a \prec d$, i.e., $c \prec a \prec d$. However, then edge ab ends below the lighter edge cd; a contradiction.

$\mathcal{F}_3$: By Lemma 16, c or d is the last vertex of the 5-cycle $abcde$. In the 4-cycle $bcde$, however, b or e is the last vertex; a contradiction. ◀

4.4 Characterization and Recognition

We will now combine the results shown in the previous subsections to show the following:

▶ **Theorem 18** (⋆). *Let* G *be a graph. Then,* $\mathrm{pqn}(G) = 1$ *if and only if* G *does not contain any* $\mathcal{F}_i$ *for* $i \in \{1, \ldots, 8\}$ *as a minor.*

Proof Sketch. We analyze the structure of G and we see that either G is a graph that has $\mathrm{pqn}(G) = 1$ due to a result from Section 4.2, or G has some $\mathcal{F}_i$ for $i \in \{1, \ldots, 8\}$ as a minor.

If G is a tree, then $\mathrm{pqn}(G) = 1$ by Lemma 7. If G contains exactly one cycle C, then $G - E(C)$ is a forest F, and we consider each component of F as rooted at C. If every component is an isolated vertex or a rooted star, then $\mathrm{pqn}(G) = 1$ by Lemma 10. If some component is more complex than a rooted star or a rooted caterpillar, then G contains $\mathcal{F}_4$ as a minor. Now suppose some component K is a rooted caterpillar. If all other components are isolated vertices, then $\mathrm{pqn}(G) = 1$ by Lemma 11. If there are two further non-trivial components, then G contains $\mathcal{F}_5$ as a minor. So assume that there is exactly one further

non-trivial component $K' \neq K$. If the roots of K and K' have distance at least 3 in one direction along C, then G contains $\mathcal{F}_8$ as a minor. If none of the above applies, either C is a triangle and $\mathrm{pqn}(G) = 1$ by Corollary 13, or C is a quadrangle and the roots of K and K' are opposite on C, and $\mathrm{pqn}(G) = 1$ by Lemma 12. The case where G contains at least two cycles remains. If G has two edge-disjoint cycles, then G contains $\mathcal{F}_2$ as a minor. If two cycles have at least two vertices but no edge in common, then G contains $\mathcal{F}_7$ as a minor. If any two cycles share at least one edge, then either $G = K_{2,3}$ or $G = K_4 - e$ and we have $\mathrm{pqn}(G) = 1$ by Lemma 14 and Corollary 15, or G contains $\mathcal{F}_1$, $\mathcal{F}_6$, or $\mathcal{F}_7$ as a minor. ◀

Theorem 18 implies that the structure of a graph G with $\mathrm{pqn}(G) = 1$ is quite limited. In fact, forbidden minors $\mathcal{F}_1, \mathcal{F}_2$ and $\mathcal{F}_3$ imply that G can contain at most three cycles. More precisely, if G contains two cycles, it is necessarily a $K_{2,3}$ or a $K_4 - e$ since forbidden minors $\mathcal{F}_6$ and $\mathcal{F}_7$ forbid any other edge. Moreover, if G contains a single cycle, forbidden minor $\mathcal{F}_4$ implies that no non-trivial tree (i.e., a tree that is more than a caterpillar) can be attached to the cycle. Finally, forbidden minors $\mathcal{F}_5$ and $\mathcal{F}_8$ dictate how legs can be combined with an attached caterpillar. $\mathcal{F}_5$ implies that at most one vertex of the cycle can have legs whereas $\mathcal{F}_8$ implies that, for a cycle of length 4, legs can only be attached to the vertex opposite of the vertex attached to the caterpillar whereas for cycles of lengths larger than 4, no legs are allowed. These observations in conjunction with Lemmas 7, 8, 10, 12, and 14 and Corollaries 13 and 15 imply the following:

▶ **Corollary 19.** *Given a graph G, it can be decided in $\mathcal{O}(n)$ time if $\mathrm{pqn}(G) = 1$. Moreover, if $\mathrm{pqn}(G) = 1$, a PQ-layout Γ of $\langle G, w\rangle$ with $\mathrm{pqn}(\Gamma) = 1$ can be computed in $\mathcal{O}(n \log n)$ time for a given edge-weight function w.*

5 PQ-Layouts of Graphs with Bounded Pathwidth and Treewidth

We present results for graphs of bounded pathwidth and bounded treewidth. We assume familiarity with these concepts; otherwise see the full version [20] for formal definitions.

▶ **Theorem 20.** *Let G be a graph with pathwidth at most p. Then, $\mathrm{pqn}(G) \leq p + 1$.*

Proof. We may assume w.l.o.g. that $G = (V, E)$ is edge-maximal of pathwidth p, i.e., G is an interval graph with clique number $\omega(G) = p + 1$ [15]. Let $\{I_v = [a_v, b_v]\}_{v \in V}$ be an interval representation of G with distinct interval endpoints. In particular $b_u \neq b_v$ for any $u \neq v \in V$.

We take the ordering $v_1, \ldots, v_n$ of vertices given by their increasing right interval endpoints. That is, $b_{v_1} < \cdots < b_{v_n}$. Crucially, for any $i < j < k$ with $v_i v_k \in E$, also $v_j v_k \in E$.

To define the partition of E into $p + 1$ priority queues $\mathcal{P}_1, \ldots, \mathcal{P}_{p+1}$, we consider a proper vertex coloring of G with $\chi(G) = \omega(G) = p + 1$ colors. This exists, as G is an interval graph. Now for $c = 1, \ldots, p + 1$ let $\mathcal{P}_c$ be the set of all edges in G whose right endpoint in the vertex ordering has color c. In fact, each $\mathcal{P}_c$ contains none of the forbidden configurations in Figure 2, and hence is a priority queue independent of the edge-weights, since in each forbidden configuration in Figure 2 there would be an edge in G between the right endpoints v_r and $v_{r'}$ of the two edges e and e' forming the forbidden configuration. But this would imply different colors for v_r and $v_{r'}$ and therefore different priority queues for e and e'. ◀

We remark that Corollary 4 implies that there are graphs with pathwidth p and priority queue number at least $\left\lfloor \frac{3-\sqrt{5}}{8}(p+1) \right\rfloor$ as K_{p+1} has pathwidth p.

Next, we shall construct for every integer $p \geq 1$ a graph $G = G_p$ of treewidth 2 together with edge weights w so that every vertex ordering σ of G contains a p-inversion. If every vertex ordering of $\langle G, w\rangle$ contains a p-inversion, then $\mathrm{pqn}(G, w) \geq p$ and hence $\mathrm{pqn}(G) \geq p$.

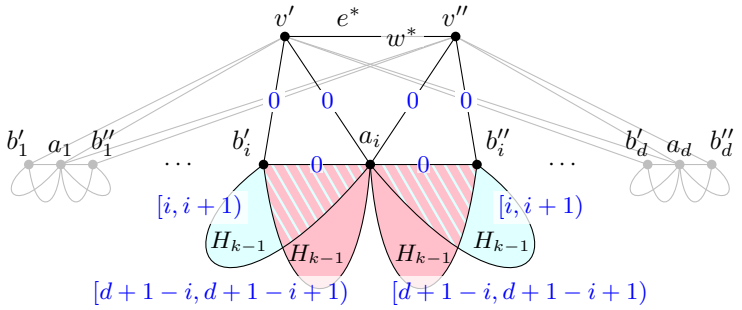


Figure 5 Illustration of the 2-tree H_k with edge-weighting w_k, for $k \geq 2$.

▶ **Theorem 21** (⋆). *For every p there is a graph G_p of treewidth 2 and $\mathrm{pqn}(G_p) \geq p$.*

Proof Sketch. Let $p \geq 1$ be a fixed integer. Our desired graph G_p will be a *rooted 2-tree*, i.e., an edge-maximal graph of treewidth 2 with designated root edge, which are defined inductively through the following construction sequence:

- A single edge $e^* = v'v''$ is a 2-tree rooted at e^*.
- If G is a 2-tree rooted at e^* and $e = uv$ is any edge of G, then adding a new vertex t with neighbors u and v is again a 2-tree rooted at e^*. We say that t is *stacked onto* uv.

We define a vertex ordering σ of a 2-tree rooted at $e^* = v'v''$ to be *left-growing* if no vertex (different from v', v'') appears to the right of both its parents. First we show that it is enough to force a p-inversion (and hence $\mathrm{pqn}(G) \geq p$) in every left-growing vertex ordering.

▷ Claim 22 (⋆). If $\langle G, w\rangle$ is an edge-weighted rooted 2-tree so that every left-growing vertex ordering contains a p-inversion, then there exists an edge-weighted rooted 2-tree $\langle \tilde{G}, \tilde{w}\rangle$ so that every vertex ordering contains a p-inversion.

Proof Sketch. Intuitively, whenever in the construction sequence of G a vertex t is stacked onto an edge uv, in $\tilde{G}$ we instead stack p^2 "copies" of t called $t_1, \ldots, t_{p^2}$ onto uv (treating each t_i like t henceforth). The weights $\tilde{w}(ut_i)$ and $\tilde{w}(vt_i)$, $i = 1, \ldots, p^2$, are chosen very close to $w(ut)$ and $w(vt)$ but increasing with i for ut_i and decreasing with i for vt_i. This way, if all p^2 copies of t lie right of u and v, we get a p-inversion among the ut_i's or vt_i's. ◁

We then construct a sequence $\langle H_1, w_1\rangle, \langle H_2, w_2\rangle, \ldots$ of edge-weighted rooted 2-trees, so that for every k, every left-growing vertex ordering of $\langle H_k, w_k\rangle$ contains a k-inversion. The construction of $\langle H_k, w_k\rangle$ is recursive, starting with $\langle H_1, w_1\rangle$ being just a single edge of any weight. For $k \geq 2$, we start with $d = k^2$ vertices $a_1, \ldots, a_d$ stacked onto the root edge $e^* = v'v''$, and stacking a vertex b'_i onto each $v'a_i$, as well as a vertex b''_i onto $v''a_i$ for $i = 1, \ldots, d$. All these edges have weight 0. Then, each b'_ia_i and b''_ia_i is used as the base edge of two separate copies of $\langle H_{k-1}, w_{k-1}\rangle$, where however the edge-weights w_{k-1} of each copy are scaled and shifted to be in a specific half-open interval $[x, y) \subset \mathbb{R}$ depending on the current base edge (b'_ia_i or b''_ia_i). See Figure 5 for an illustration.

By induction, there is a $(k-1)$-inversion in each such copy of H_{k-1}. Using the pigeon-hole principle, and the Erdős–Szekeres theorem, we then identify an edge a_iv' (or a_iv'') which forms a k-inversion either together with $(k-1)$-inversion of one copy of H_{k-1}, or with $k-1$ edges, each from the $(k-1)$-inversion of a different copy of H_{k-1}.

To finish the proof, it is then enough to take $\langle H_p, w_p\rangle$ as constructed above, and then apply Claim 22 to obtain the desired edge-weighted 2-tree $\langle G_p, w_p\rangle$ with $\mathrm{pqn}(G_p, w_p) \geq p$. ◀

As graphs of treewidth 2 are always planar graphs, we conclude with the following result that contrasts similar results for the queue and stack number [23, 43]:

▶ **Corollary 23.** *The priority queue number of planar graphs is unbounded. In particular, there is a planar graph G with n vertices and* $\mathrm{pqn}(G) \geq \log_4 n$.

Proof. We observe that H_k in the proof of Theorem 21 contains 4 copies of H_{k-1} while H_1 contains 2 vertices. Thus, there are at least 4^p vertices in $G := H_p$, which has treewidth p. ◀

6 Complexity of Fixed-Ordering PQ-Layouts

In the light of Corollary 19, one might wonder whether deciding if a given graph G has priority queue number k is polynomial-time solvable for all values of k. However, we show that it is NP-complete if the ordering of the vertices is already fixed.

▶ **Theorem 24.** *Given an edge-weighted graph $\langle G, w\rangle$ with $G = (V, E)$, a linear ordering σ of V, and a positive integer k, it is* NP*-complete to decide whether $\langle G, w\rangle$ admits a PQ-layout with linear ordering σ and a partitioning of E into k priority queues.*

Proof. Containment in NP is clear as we can check in polynomial time whether a given assignment of edges to k priority queues corresponds to a valid PQ-layout.

We show NP-hardness by reduction from the problem of coloring circular-arc graphs, which is known to be NP-complete if k is not fixed [27]. A circular-arc graph is the intersection graph of arcs of a circle (see Figure 6 on the left), that is, the arcs are the vertices and two vertices share an edge if and only if the corresponding arcs share a point along the circle.

For a given circular-arc graph H, we next describe how to obtain an edge-weighted graph $\langle G, w\rangle$ and a vertex ordering σ such that a k-coloring of H directly corresponds to a PQ-layout of $\langle G, w\rangle$ under σ with k priority queues. Without loss of generality, we assume that we have a circular-arc representation of H where all endpoints of the arcs are distinct.

We "cut" the circle at some point, which gives an interval representation where some vertices S of H refer to two intervals; in Figure 6, the circular arcs a, b, and d are cut into intervals a_1 and a_2, b_1 and b_2, and d_1 and d_2, respectively. We let the endpoints of all intervals be the vertices of G, whose ordering σ along the spine is taken from the ordering of the corresponding endpoints of the intervals (for the intervals belonging to S, the order is arbitrary). Further, for each interval, we have an edge in G connecting its two endpoints. To obtain w, we assign to the edges of G the numbers 1 to m, where $m = |V(H)| + |S|$, in decreasing order of the right endpoints of the intervals. This way, every pair of edges whose corresponding intervals share a common point need to be assigned to distinct priority queues.

So far, there is no mechanism that assures that a_1 and a_2, b_1 and b_2, etc., are assigned to the same priority queue. To this end, we add, for each such pair $\langle a_1, a_2\rangle$, an edge from the left endpoint of a_1 to the right endpoint of a_2. We call these $|S|$ new edges *synchronization edges*, and we assign them the weights $m+1, \ldots, m+|S|$ in decreasing order of their right endpoints. Furthermore, we add $k-1, k-2, \ldots$ *heavy* edges below the leftmost, second leftmost, ... vertex on the spine, respectively, as well as, $k-1, k-2, \ldots$ heavy edges below the rightmost, second rightmost, ... vertex on the spine, respectively, see Figure 6. Clearly, $k \geq |S|$, otherwise we have a no-instance. The heavy edges have distinct right endpoints and their weights are chosen such that (i) they are greater than M, where $M = m + |S|$ is the largest weight used so far, and (ii) they are chosen in decreasing order from left to right such that each pair of heavy edges needs to be assigned to distinct priority queues if they overlap.

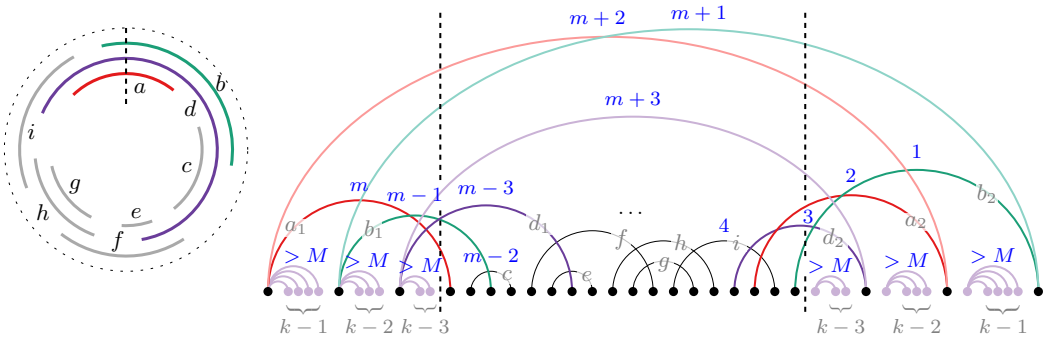


Figure 6 Illustration of our reduction.

▷ Claim 25 (⋆). Let Γ be a PQ-layout of $\langle G, w\rangle$ under vertex ordering σ having k priority queues. For each vertex $a \in S$ ($\subseteq V(H)$), the two corresponding edges a_1 and a_2 in G are assigned to the same priority queue in Γ.

Proof Sketch. Consider the vertices of G from the outside to the inside. At the outer endpoint of a_1 (a_2, resp.), the edge a_1 (a_2, resp.) and the synchronization edge get the same color by the pigeonhole principle because all but one priority queue is occupied by the heavy edges and the previously considered edges, which all induce pairwise forbidden configurations. ◁

If we have a PQ-layout of $\langle G, w\rangle$ under vertex ordering σ with k priority queues, we obtain, due to the choice of w and σ, a coloring of H by using the priority queues as colors. In particular, due to Claim 25, both parts of the "cut" circular arcs get the same color. Conversely, we can easily assign the edges of $\langle G, w\rangle$ to k priority queues if we are given a k-coloring of H. Clearly, our reduction can be implemented to run in polynomial time. ◀

7 Open Problems

We conclude by summarizing a few intriguing problems that remain open. (i) Improve the upper and lower bounds for $\mathrm{pqn}(K_n)$. (ii) Determine the complexity of deciding $\mathrm{pqn}(G) \leq k$ for $k > 1$, or deciding $\mathrm{pqn}(G, w) \leq k$ for a given edge weighting w when k is a constant or no vertex ordering is given. (iii) Find graph families with bounded priority queue number; what about outerplanar graphs? (iv) Can the edge density of a graph G be upper bounded by a term in $\mathrm{pqn}(G)$? (v) One can also investigate the structure of edge-weighted graphs $\langle G, w\rangle$ with a fixed vertex-ordering $\prec$. Edge-partitions of G into k priority queues are equivalent to proper k-colorings of an associated graph H, while t-inversions correspond to t-cliques in H. Is $\chi(H)$ bounded by a function in terms of $\omega(H)$? Is determining $\omega(H)$ NP-complete?

References

1 Jawaherul Md. Alam, Michael A. Bekos, Martin Gronemann, Michael Kaufmann, and Sergey Pupyrev. Queue layouts of planar 3-trees. *Algorithmica*, 82(9):2564–2585, 2020. doi:10.1007/s00453-020-00697-4.

2 Patrizio Angelini, Michael A. Bekos, Philipp Kindermann, and Tamara Mchedlidze. On mixed linear layouts of series-parallel graphs. *Theor. Comput. Sci.*, 936:129–138, 2022. doi:10.1016/j.tcs.2022.09.019.

3 Christopher Auer, Christian Bachmaier, Franz-Josef Brandenburg, Wolfgang Brunner, and Andreas Gleißner. Plane drawings of queue and deque graphs. In Ulrik Brandes and Sabine Cornelsen, editors, *Proc. 18th International Symposium on Graph Drawing and Network Visualization (GD 2010)*, volume 6502 of *Lecture Notes in Computer Science*, pages 68–79. Springer, 2010. doi:10.1007/978-3-642-18469-7_7.

4 Christopher Auer and Andreas Gleißner. Characterizations of deque and queue graphs. In Petr Kolman and Jan Kratochvíl, editors, *Proc. 37th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2011)*, volume 6986 of *Lecture Notes in Computer Science*, pages 35–46. Springer, 2011. doi:10.1007/978-3-642-25870-1_5.

5 Michael J. Bannister, William E. Devanny, Vida Dujmović, David Eppstein, and David R. Wood. Track layouts, layered path decompositions, and leveled planarity. *Algorithmica*, 81(4):1561–1583, 2019. doi:10.1007/s00453-018-0487-5.

6 Giuseppe Di Battista, Fabrizio Frati, and János Pach. On the queue number of planar graphs. *SIAM J. Comput.*, 42(6):2243–2285, 2013. doi:10.1137/130908051.

7 Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Marco Tais. Schematic representation of large biconnected graphs. *J. Graph Algorithms Appl.*, 25(1):311–352, 2021. doi:10.7155/jgaa.00560.

8 Michael A. Bekos, Stefan Felsner, Philipp Kindermann, Stephen G. Kobourov, Jan Kratochvíl, and Ignaz Rutter. The rique-number of graphs. In Patrizio Angelini and Reinhard von Hanxleden, editors, *Proc. 30th International Symposium on Graph Drawing and Network Visualization (GD 2022)*, volume 13764 of *Lecture Notes in Computer Science*, pages 371–386. Springer, 2022. doi:10.1007/978-3-031-22203-0_27.

9 Michael A. Bekos, Henry Förster, Martin Gronemann, Tamara Mchedlidze, Fabrizio Montecchiani, Chrysanthi N. Raftopoulou, and Torsten Ueckerdt. Planar graphs of bounded degree have bounded queue number. *SIAM J. Comput.*, 48(5):1487–1502, 2019. doi:10.1137/19M125340X.

10 Michael A. Bekos, Martin Gronemann, and Chrysanthi N. Raftopoulou. An improved upper bound on the queue number of planar graphs. *Algorithmica*, 85(2):544–562, 2023. doi:10.1007/s00453-022-01037-4.

11 Michael A. Bekos, Michael Kaufmann, Fabian Klute, Sergey Pupyrev, Chrysanthi N. Raftopoulou, and Torsten Ueckerdt. Four pages are indeed necessary for planar graphs. *J. Comput. Geom.*, 11(1):332–353, 2020. doi:10.20382/jocg.v11i1a12.

12 Michael A. Bekos, Michael Kaufmann, Maria Eleni Pavlidi, and Xenia Rieger. On the deque and rique numbers of complete and complete bipartite graphs. In Denis Pankratov, editor, *Proc. 35th Canadian Conference on Computational Geometry (CCCG 2023)*, pages 89–95, 2023. URL: https://wadscccg2023.encs.concordia.ca/assets/pdf/CCCG_2023_proc.pdf#section.0.11.

13 Frank Bernhart and Paul C. Kainen. The book thickness of a graph. *J. Comb. Theory, Ser. B*, 27(3):320–331, 1979. doi:10.1016/0095-8956(79)90021-2.

14 Sandeep N. Bhatt, Fan R. K. Chung, Frank Thomson Leighton, and Arnold L. Rosenberg. Scheduling tree-dags using FIFO queues: A control-memory trade-off. *J. Parallel Distributed Comput.*, 33(1):55–68, 1996. doi:10.1006/jpdc.1996.0024.

15 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

16 Jonathan F. Buss and Peter W. Shor. On the pagenumber of planar graphs. In Richard A. DeMillo, editor, *Proc. 16th Annual ACM Symposium on Theory of Computing (STOC 1984)*, pages 98–100. ACM, 1984. doi:10.1145/800057.808670.

17 Fan R. K. Chung, Frank Thomson Leighton, and Arnold L. Rosenberg. DIOGENES: A methodology for designing fault-tolerant VLSI processor arrays. In *Proc. 13th Int. Conf. Fault-Tolerant Computing*, pages 26–32, 1983.

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

19 Philipp de Col, Fabian Klute, and Martin Nöllenburg. Mixed linear layouts: Complexity, heuristics, and experiments. In Daniel Archambault and Csaba D. Tóth, editors, *Proc. 27th International Symposium on Graph Drawing and Network Visualization (GD 2019)*, volume 11904 of *Lecture Notes in Computer Science*, pages 460–467. Springer, 2019. doi:10.1007/978-3-030-35802-0_35.

20 Emilio Di Giacomo, Walter Didimo, Henry Förster, Torsten Ueckerdt, and Johannes Zink. Linear layouts of graphs with priority queues. arXiv preprint, 2025. doi:10.48550/arXiv.2506.23943.

21 Emilio Di Giacomo, Giuseppe Liotta, and Henk Meijer. Computing straight-line 3D grid drawings of graphs in linear volume. *Comput. Geom.*, 32(1):26–58, 2005. doi:10.1016/j.comgeo.2004.11.003.

22 Vida Dujmović. Graph layouts via layered separators. *J. Comb. Theory, Ser. B*, 110:79–89, 2015. doi:10.1016/j.jctb.2014.07.005.

23 Vida Dujmović, Gwenaël Joret, Piotr Micek, Pat Morin, Torsten Ueckerdt, and David R. Wood. Planar graphs have bounded queue-number. *J. ACM*, 67(4):22:1–22:38, 2020. doi:10.1145/3385731.

24 Vida Dujmović, Pat Morin, and David R. Wood. Layout of graphs with bounded tree-width. *SIAM J. Comput.*, 34(3):553–579, 2005. doi:10.1137/S0097539702416141.

25 Vida Dujmović, Attila Pór, and David R. Wood. Track layouts of graphs. *Discret. Math. Theor. Comput. Sci.*, 6(2):497–522, 2004. doi:10.46298/DMTCS.315.

26 Henry Förster, Michael Kaufmann, Laura Merker, Sergey Pupyrev, and Chrysanthi N. Raftopoulou. Linear layouts of bipartite planar graphs. In Pat Morin and Subhash Suri, editors, *Proc. 18th Algorithms and Data Structures Symposium (WADS 2023)*, volume 14079 of *Lecture Notes in Computer Science*, pages 444–459. Springer, 2023. doi:10.1007/978-3-031-38906-1_29.

27 M. R. Garey, David S. Johnson, Gerald L. Miller, and Christos H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM J. Algebraic Discret. Methods*, 1(2):216–227, 1980. doi:10.1137/0601025.

28 Christian Haslinger and Peter F. Stadler. RNA structures with pseudo-knots: Graph-theoretical, combinatorial, and statistical properties. *Bull. Math. Biol.*, 61(3):437–467, 1999. doi:10.1006/bulm.1998.0085.

29 Lenwood S. Heath. Embedding planar graphs in seven pages. In *Proc. 25th Annual Symposium on Foundations of Computer Science (FOCS 1984)*, pages 74–83. IEEE Computer Society, 1984. doi:10.1109/SFCS.1984.715903.

30 Lenwood S. Heath, Frank Thomson Leighton, and Arnold L. Rosenberg. Comparing queues and stacks as mechanisms for laying out graphs. *SIAM J. Discret. Math.*, 5(3):398–412, 1992. doi:10.1137/0405031.

31 Lenwood S. Heath and Arnold L. Rosenberg. Laying out graphs using queues. *SIAM J. Comput.*, 21(5):927–958, 1992. doi:10.1137/0221055.

32 Sorin Istrail. An algorithm for embedding planar graphs in six pages. *Iasi University Annals, Mathematics-Computer Science*, 34(4):329–341, 1988.

33 Paul C. Kainen. The book thickness of a graph, II. *Congressus Numerantium*, 71:127–132, January 1990.

34 Donald E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms*. Addison-Wesley, 1968.

35 Frank Thomson Leighton and Arnold L. Rosenberg. Three-dimensional circuit layouts. *SIAM J. Comput.*, 15(3):793–813, 1986. doi:10.1137/0215057.

36 Jawaherul Md. Alam, Michael A. Bekos, Martin Gronemann, Michael Kaufmann, and Sergey Pupyrev. The mixed page number of graphs. *Theoretical Computer Science*, 931:131–141, 2022. doi:10.1016/j.tcs.2022.07.036.

37 Vaughan R. Pratt. Computing permutations with double-ended queues, parallel stacks and parallel queues. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 268–277. ACM, 1973. doi:10.1145/800125.804058.

38 Sergey Pupyrev. Mixed linear layouts of planar graphs. In Fabrizio Frati and Kwan-Liu Ma, editors, *Proc. 25th International Symposium on Graph Drawing and Network Visualization (GD 2017)*, volume 10692 of *Lecture Notes in Computer Science*, pages 197–209. Springer, 2017. doi:10.1007/978-3-319-73915-1_17.

39 Neil Robertson and Paul D. Seymour. Graph minors. XIII. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995. doi:10.1006/JCTB.1995.1006.

40 Neil Robertson and Paul D. Seymour. Graph minors. XX. Wagner's conjecture. *J. Comb. Theory, Ser. B*, 92(2):325–357, 2004. doi:10.1016/J.JCTB.2004.08.001.

41 Arnold L. Rosenberg. The diogenes approach to testable fault-tolerant arrays of processors. *IEEE Trans. Computers*, C-32(10):902–910, 1983. doi:10.1109/TC.1983.1676134.

42 David R. Wood. Bounded-degree graphs have arbitrarily large queue-number. *Discret. Math. Theor. Comput. Sci.*, 10(1), 2008. doi:10.46298/dmtcs.434.

43 Mihalis Yannakakis. Embedding planar graphs in four pages. *J. Comput. Syst. Sci.*, 38(1):36–67, 1989. doi:10.1016/0022-0000(89)90032-9.

44 Mihalis Yannakakis. Planar graphs that need four pages. *J. Comb. Theory, Ser. B*, 145:241–263, 2020. doi:10.1016/j.jctb.2020.05.008.

Convolution and Knapsack in Higher Dimensions

Kilian Grage
Kiel University, Germany

Klaus Jansen
Kiel University, Germany

Björn Schumacher
Kiel University, Germany

Abstract

In the Knapsack problem, one is given the task of packing a knapsack of a given size with items in order to gain a packing with a high profit value. As one of the most classical problems in computer science, research for this problem has gone a long way. One important connection to the $(\max, +)$-convolution problem has been established, where knapsack solutions can be combined by building the convolution of two sequences. This observation has been used in recent years to give conditional lower bounds but also parameterized algorithms.

In this paper we carry these results into higher dimensions. We consider Knapsack where items are characterized by multiple properties – given through a vector – and a knapsack that has a capacity vector. The packing must not exceed any of the given capacity constraints. In order to show a similar sub-quadratic lower bound we consider a multidimensional version of $(\max, +)$-convolution. We then consider variants of this problem introduced by Cygan et al. and prove that they are all equivalent in terms of algorithms that allow for a running time sub-quadratic in the number of entries of the array.

We further develop a parameterized algorithm to solve higher dimensional Knapsack. The techniques we apply are inspired by an algorithm introduced by Axiotis and Tzamos. We will show that even for higher dimensional Knapsack, we can reduce the problem to convolution on one-dimensional, concave sequences, leading to an $\mathcal{O}(dn + dD \cdot \max\{\Pi_{i=1}^{d} t_i, t_{\max} \log t_{\max}\})$ algorithm, where D is the number of different weight vectors, t the capacity vector and d is the dimension of the problem. Then, we use the techniques to improve the approach of Eisenbrand and Weismantel to obtain an algorithm for Integer Linear Programming with upper bounds with running time $\mathcal{O}(dn) + D \cdot \mathcal{O}(d\Delta)^{d(d+1)} + T_{\text{LP}}$.

Finally, we give an divide-and-conquer algorithm for ILP with running time $n^{d+1} \cdot \mathcal{O}(\Delta)^d \cdot \log(\|u - \ell\|_\infty)$.

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

Keywords and phrases Knapsack, Convolution, Integer Linear Programming

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.30

Related Version *Full Version*: `https://arxiv.org/abs/2403.16117` [12]

Funding *Klaus Jansen*: Supported by the German Research Foundation (DFG) project JA 612/25-1.
Björn Schumacher: Supported by the German Research Foundation (DFG) project JA 612/25-1.

Acknowledgements We want to thank the anonymous referees for their time and helpful feedback.

1 Introduction

The Knapsack problem is one of the core problems in computer science. The task of finding a collection of items that fits into a knapsack but also maximizes some profit is NP-hard and as such the aim is to find approximation algorithms, parameterized algorithms, and determine lower bounds for the running time of exact algorithms.

© Kilian Grage, Klaus Jansen, and Björn Schumacher;
licensed under Creative Commons License CC-BY 4.0

Cygan et al. [9] and Künnemann et al. [16] among others have used the following relationship between Knapsack and the $(\max,+)$-convolution problem. Assume we are given two disjoint item sets A and B and a knapsack of size t. If we additionally know the optimal profits for all knapsack sizes $t' \leq t$ we then can calculate the maximum profits for all sizes t' for $A \cup B$ by using convolution. This connection was used in order to show that the existence of a sub-quadratic (in terms of capacity) algorithm for Knapsack is equivalent to the existence of a sub-quadratic algorithm for $(\max,+)$-convolution.

In this work, we consider these problems in higher dimensions. The Knapsack problem is simply generalized by replacing the single value weight and capacity by vectors. We then look for a collection of items whose summed up vectors do not exceed the capacity vector in any position. The natural question arises whether a similar quadratic time lower bound exists for this problem. We answer this question positively by giving a generalization of convolution in higher dimensions as well. Using this generalization and techniques introduced by Bringmann [4] such as color-coding we are able to achieve similar subquadratic lower bounds as in the one-dimensional case.

1.1 Problem Definitions and Notations

We define $[k] \coloneqq \{i \in \mathbb{N} : 1 \leq i \leq k\}$, $[k]_0 \coloneqq [k] \cup \{0\}$, and $[a,b] \coloneqq \{i \in \mathbb{Z} : a \leq i \leq b\}$ for $k, a, b \in \mathbb{Z}$. In the following, we write for two vectors $v, u \in \mathbb{R}^d$ that $v \leq u$ (resp. $v < u$) if for all $i \in [d]$ we have that $v_i \leq u_i$ (resp. $v_i < u_i$). Further we denote with $v_{\max} = \max_{i \in [d]} v_i$ for any vector $v \in \mathbb{R}^d$. With $\vec{\mathbf{k}} \in \mathbb{R}^d$ we denote the vector that has $k \in \mathbb{R}$ in every position.

▶ **Definition 1.** *Let $L \in \mathbb{N}^d$ be a d-dimensional vector and $A = (A_{i_1 i_2 \cdots i_d})$ be a $L_1 \times L_2 \times \cdots \times L_d$* array*. We call the vector L the* size *of A and denote the number of entries in A with $\Pi(L) \coloneqq \prod_{i=1}^{d} L_i$.*

We call a vector $v \in \mathbb{Z}$ with $\vec{\mathbf{0}} \leq v \leq L - \vec{\mathbf{1}}$ position of array A. For ease of notation, we will denote for any array position v of A the respective array entry $A_{v_1 v_2 \cdots v_d}$ with A_v.

We note that in this definition, array positions lie in between $\vec{\mathbf{0}}$ and $L - \vec{\mathbf{1}}$. This makes it easier to work with positions for convolution and also for formulating time complexity bounds, as $\Pi(L)$ will be the main parameter we consider.

▶ **Definition 2** (Maximum Convolution)**.** *Given two d-dimensional arrays A, B with equal size L, the $(\max,+)$-convolution of A and B denoted as $A \oplus B$ is defined as an array C of size L with $C_v \coloneqq \max_{u \leq v} A_u + B_{v-u}$ for any $v < L$.*

d-MaxConv	
Input:	Two d-dimensional arrays A, B with equal size L.
Problem:	Compute the array $C \coloneqq A \oplus B$ of size L.

Note that in the following we will refer to this problem as "Convolution". We specifically limit ourselves to the special truncated case where both input arrays and the output array have the same size. In a more general setting, we could allow arrays of sizes $L^{(1)}, L^{(2)}$ as input and compute an array of size $L^{(1)} + L^{(2)} - \vec{\mathbf{1}}$.

To measure the running time of our algorithms, we mainly consider the size or rather the number of entries from the resulting array because we need to calculate a value for every position. Therefore, in terms of theoretical performance, working with different sizes or calculating an array of combined size will not make a difference. By only considering arrays of the same size, we avoid many unnecessary cases and the dummy values of $-\infty$.

There is a quadratic time algorithm for d-MaxConv like in the one-dimensional case. This algorithm simply iterates through all pairs of positions in time $\mathcal{O}(\Pi(L)^2) \subseteq \mathcal{O}(L_{\max}^{2d})$.

Further we define an upper bound test for the convolution problem. In this problem, we are given a third input array and need to decide whether its entries are upper bounds for the entries of the convolution.

d-MaxConv UpperBound	
Input:	Three d-dimensional arrays A, B, C with equal size L.
Problem:	Decide whether $(A \oplus B)_v \leq C_v$ for all $v < L$.

We further generalize the notion of superadditivity to multidimensional arrays.

d-SuperAdditivity Testing	
Input:	One d-dimensional array A of size L.
Problem:	Decide whether A is superadditive, i.e., $A_v \geq (A \oplus A)_v$ for all $v < L$.

The next problem class we consider is the d-Knapsack problem. In the one-dimensional case, one is given a set of items with weights and profits and one knapsack with a certain weight capacity. The goal is now to pack the knapsack such that the profit is maximized and the total weight of packed items does not exceed the weight capacity.

The natural higher dimensional generalization arises when we have more constraints to fulfill. When going on a journey by plane, one for example has several further requirements such as a maximum amount of allowed liquid or number of suitcases. By imposing more similar requirements, we can simply identify each item by a vector and also define the knapsack by a capacity vector. This leads to the following generalization of Knapsack into higher dimensions. We further differentiate between two problems 0/1 d-Knapsack and Unbounded d-Knapsack, depending on whether we allow items to be only used one time or an arbitrary number of times.

0/1 d-Knapsack	
Input:	Set I of n items each defined by a profit $p_i \in \mathbb{R}_{\geq 0}$ and weight vector $w^{(i)} \in \mathbb{N}^d$, along with a knapsack of capacity $t \in \mathbb{N}^d$.
Problem:	Find subset $S \subseteq I$ such that $\sum_{i \in S} w^{(i)} \leq t$ and $\sum_{i \in S} p_i$ is maximal.

Unbounded d-Knapsack	
Input:	Set I of n items each defined by a profit $p_i \in \mathbb{R}_{\geq 0}$ and weight vector $w^{(i)} \in \mathbb{N}^d$, along with a knapsack of capacity $t \in \mathbb{N}^d$.
Problem:	Find a multi-set $S \subseteq I$ such that $\sum_{i \in S} w^{(i)} \leq t$ and $\sum_{i \in S} p_i$ is maximal.

The running time for these problems is mainly dependent on the dimension d, the number of items n of an instance and the number of feasible capacities $\Pi(t + \vec{\mathbf{1}})$ and we will further study the connection of these in regards to the convolution problems.

1.2 Related Work

Cygan et al. [9] as well as Künnemann et al. [16] initiated the research and were the first to introduce this class of problems and convolution hardness. In particular, Cygan et al. showed for $d = 1$ that all these problems are equivalent in terms of whether they allow for a subquadratic algorithm. This allows to formulate a conditional lower bound for all these problems under the hypothesis that no subquadratic algorithm for d-MaxConv exists.

▶ **Conjecture 3** (MaxConv-hypothesis). *There exists no $\mathcal{O}(\Pi(L)^{2-\varepsilon})$ time algorithm for any $\varepsilon > 0$ for d-MaxConv with $d = 1$.*

(max, +)-Convolution. The best known algorithm to solve convolution on sequences without any further assumption takes time $n^2/2^{\Omega(\sqrt{\log n})}$. This result was achieved by Williams [21], who gave an algorithm for APSP in conjunction with a reduction by Bremner et al. [3]. However, the existence of a truly subquadratic algorithm remains open.

Research therefore has taken a focus on special cases of convolution where one or both input sequences is required to have certain structural properties such as monotony, concavity or linearity. Chan and Lewenstein [6] gave a subquadratic $\mathcal{O}(n^{1.864})$ algorithm for instances where both sequences are monotone increasing and the values are bound by $\mathcal{O}(n)$. Chi et al. [8] improved this further with a randomized $\widetilde{\mathcal{O}}(n^{1.5})$ algorithm.

Axiotis and Tzamos [1] showed that Convolution with only one concave sequence can be solved in linear time. Gribanov et al. [13] studied multiple cases and gave subquadratic algorithms when one sequence is convex, piece-wise linear or polynomial.

Knapsack. Convolution has proven to be a useful tool to solve other problems as well, in particular the Knapsack problem. In fact one of the reductions of Cygan et al. [9] was an adapted version of Bringmann's algorithm for subset sum [4]. Bringmann's algorithm works by constructing sub-instances, solving these and then combining the solutions via Fast Fourier Transformation (FFT). The algorithm by Cygan et al. to solve Knapsack works the same way, but uses Convolution instead of FFT.

Axiotis and Tzamos follow a similar approach but choose their sub-instances more carefully, by grouping items with the same weight. That way, the solutions make up concave profit sequences which can be combined in linear time [1]. This yields in total an $\mathcal{O}(n + Dt)$ algorithm, where D is the number of different item weights.

Polak et al. [19] gave an $\mathcal{O}(n + \min\{w_{\max}, p_{\max}\}^3)$ algorithm, where $w_{\max}, p_{\max}$ denote the maximum weight and profit respectively. They achieved this by combining the techniques from Axiotis and Tzamos with proximity results from Eisenbrand and Weismantel [11]. Further, Chen et al. [7] improved this to a time of $\widetilde{\mathcal{O}}(n + w_{\max}^{2.4})$. Recently, Ce Jin [14] gave an improved $\widetilde{\mathcal{O}}(n + w_{\max}^2)$ algorithm. Independently to these results for 0/1 Knapsack, Bringmann gave an $\widetilde{\mathcal{O}}(n + w_{\max}^2)$ algorithm for Bounded Knapsack.

Doron-Arad et al. [10] showed that there are constants $\zeta, d_0 > 0$, such that for every integer $d > d_0$ there is no algorithm that solves 0/1 d-Knapsack in time $\mathcal{O}((n + t_{\max})^{\zeta \frac{d}{\log d}})$.

Integer Linear Program. d-dimensional Knapsack problems are a special case of integer linear programs.

Integer Linear Program (ILP)	
Input:	A matrix $A \in \mathbb{Z}^{d \times n}$, a target vector $b \in \mathbb{Z}^d$, a profit vector $c \in \mathbb{Z}^n$ and an upper bound vector $u \in \mathbb{N}^d$.
Problem:	Find $x \in \mathbb{Z}^n$ with $Ax = b$, $0 \leq x \leq u$ and such that $c^\mathsf{T} x$ is maximal.

Lower bounds $\ell \in \mathbb{N}^d$ may be present but can easily be removed by changing the right side to $b - A\ell$ and the upper bound to $u - \ell$. If we limit A to be a matrix with only non-negative entries and $u \coloneqq \vec{\mathbf{1}}$ then solving the above defined ILP is equivalent to 0/1 d-Knapsack. If we omit the $x \leq u$ constraint, the resulting problem is Unbounded d-Knapsack.

A very important part in research of ILPs has been on *proximity*. The proximity of an ILP is the distance between an optimal solution and an optimal solution of its relaxation. The relaxation of an ILP is given by allowing the solution vector x to be fractional.

Eisenbrand and Weismantel [11] have proven that for an optimal solution of the LP-relaxation $x^* \in \mathbb{R}^n$ there is an optimal integral solution $z^* \in \mathbb{Z}^n$ with $||x^* - z^*||_1 \leq d(2d\Delta+1)^d$ with Δ being the largest absolute entry in A. They used this result to give an algorithm that solves ILPs without upper bounds in time $(d\Delta)^{\mathcal{O}(m)} \cdot \|b\|_\infty^2$ and ILPs as defined above in time $\mathcal{O}(n \cdot \mathcal{O}(d)^{(d+1)^2} \cdot \mathcal{O}(\Delta)^{d(d+1)} \cdot \log^2(d\Delta))$. They additionally extended their result to Bounded Knapsack and obtained the running time $\mathcal{O}(n^2 \cdot \Delta^2)$.

There have been further results on proximity such as Lee et al. [17], who gave a proximity bound of $3d^2 \log(2\sqrt{d} \cdot \Delta_m^{1/m}) \cdot \Delta_m$ where Δ_m is the largest absolute value of a minor of order m of matrix A. Celaya et al. [5] also gave proximity bounds for certain modular matrices.

Rohwedder and Węgrzycki [20] conjecture that there is no $2^{\mathcal{O}(m^{2-\varepsilon})}\text{poly}(n)$ time algorithm for ILP with $\Delta = \mathcal{O}(1)$. They also show that there are several problems that are equivalent with respect to this conjecture.

1.3 Our Results

We begin by expanding the results of Cygan et al. [9] into higher dimensions. The natural question is whether similar relations shown in their work also exist in higher dimensions and in fact they do. In the first part of this paper we show that the same equivalence – regarding existing subquadratic time algorithms – holds among the higher dimensional problems.

▶ **Theorem 4.** *For any fixed d and any $\varepsilon > 0$, the following statements are equivalent:*

1. *There exists an $\mathcal{O}(\Pi(L)^{2-\varepsilon})$-time algorithm for d-MaxConv.*
2. *There exists an $\mathcal{O}(\Pi(L)^{2-\varepsilon})$-time algorithm for d-MaxConv UpperBound.*
3. *There exists an $\mathcal{O}(\Pi(L)^{2-\varepsilon})$-time algorithm for d-SuperAdditivity Testing.*
4. *There exists an $\mathcal{O}(n + \Pi(t)^{2-\varepsilon})$-time algorithm for Unbounded d-Knapsack.*
5. *There exists an $\mathcal{O}(n + \Pi(t)^{2-\varepsilon})$-time algorithm for 0/1 d-Knapsack.*

Some of these reduction incur a multiplicative factor of $\mathcal{O}(2^d)$. This is a natural consequence due to the exponentially larger amount of entries that our arrays hold and that we need to process. As an example, where it was sufficient in the one-dimensional case to split a problem in two sub-problems, we may now need to consider 2^d sub-problems. For this reason we require d to be fixed, so we can omit these factors. We will prove this statement through a ring of reductions. We note that one of the reductions uses an algorithm or a generalization of it from Bringmann [4, 9] that is randomized. Part of this algorithm, involving so-called color-coding can be derandomized, but a full derandomization is still an open problem.

We note that under the MaxConv-hypothesis, there does not exist an $\mathcal{O}(\Pi(L)^{2-\varepsilon})$-time algorithm for 1-MaxConv. If there exists any d such that d-MaxConv admits a sub-quadratic algorithm, then it would also solve the problem in sub-quadratic time for any $d' \leq d$ and especially $d' = 1$, hence contradicting the MaxConv-hypothesis, because we can extend any d'-dimensional array to a d-dimensional one by adding dimensions with size one.

We also discuss how to use lower order improvements for 1-dimensional convolution for the d-dimensional convolution via a standard argument using linearization of the vector and adding appropriate padding. As this is a standard trick we omit the proof in the main body but we state it in Section A for completeness.

▶ **Lemma 5.** *Given an algorithm for 1-dimensional convolution with running time $T(n)$ and two d-dimensional arrays A, B with size L, we can calculate $A \oplus B$ in time $T(2^d \prod(L))$.*

Together with the discussion above, Lemma 5 shows that 1-dimensional and d-dimensional convolution are equivalent for fixed d or up to a factor $2^{\mathcal{O}(d)}$.

In the second part of our paper, we complement our conditional lower bound with a parameterized algorithm. To achieve this, we generalize an algorithm by Axiotis and Tzamos [1]. Our algorithm will also have a running time dependent on the number of different item weights, that we denote with D and the largest item weight $\Delta \coloneqq \max_{i \in I} \|w^{(i)}\|_\infty$. This algorithm will also group items by weight vector and solve the respective sub-instances. The resulting partial solutions are then combined via d-MaxConv. However, solving general d-MaxConv needs quadratic time in the number of entries. We can overcome this barrier by reducing our problem to convolution on one-dimensional, concave sequences.

▶ **Theorem 6.** *There is an algorithm for* Bounded d-EqualityKnapsack *with running time* $\mathcal{O}(dn + d \cdot D \cdot \max\{\prod(t+\vec{\mathbf{1}}), t_{\max} \log t_{\max}\}) \subseteq \mathcal{O}(dn + d \cdot \min\{n, (\Delta+1)^d\} \cdot \max\{\prod(t+\vec{\mathbf{1}}), t_{\max} \log t_{\max}\}))$.

In the general case our algorithm will achieve a running time of $\mathcal{O}(dn + D \cdot \Pi(t))$ as the $t_{\max} \log t_{\max}$ part only becomes relevant when we have a slim knapsack, that is very large in one component, but comparatively small in the other. We note that our algorithm achieves the lower bound proposed in Theorem 4 since D is also upper bounded by $\Pi(t)$.

We use the techniques for Theorem 6 to improve the approach of Eisenbrand and Weismantel [11] to obtain the following running time for ILP.

▶ **Theorem 7.** *There is an algorithm for ILP with running time* $\mathcal{O}(dn) + D \cdot \mathcal{O}(d\Delta)^{d(d+1)} + T_{\text{LP}}$.

Our approach reduces the dependency on the number of dimensions ($\mathcal{O}(d)^{d(d+1)}$ instead of $\mathcal{O}(d)^{(d+1)^2}$) and removes the logarithmic factor $\log^2(d\Delta)$. Further, our algorithm is mostly independent on n but relies on the number of different columns of the given ILP.

In addition to their conjecture Rohwedder and Węgrzycki [20] showed that the quadratic dependence on the number of constraints in the exponent can be avoided if the term n^d is allowed in the runtime. We improve their algorithm by removing d from the base.

▶ **Theorem 8.** *There is an algorithm that can solve ILP in time* $n^{d+1} \cdot \mathcal{O}(\Delta)^d \cdot \log(\|u - \ell\|_\infty)$.

The algorithm is a divide and conquer algorithm which repeatedly halves the upper bounds.

1.4 Organization of the Paper

In Section 2 we present an overview of the reductions to proof Theorem 4. However, the concrete proofs are in the full version [12] as the proofs are similar to the ones by Cygan et al. [9]. Next we give the parameterized algorithm as well as the generalization to ILPs in Section 3. The proof of the divide-and-conquer algorithm for Theorem 8 is given in Section 4.

2 Reductions

For the Convolution problems, we will formulate the running time via $T(\Pi(L), d)$, where d is the dimension and L is the size of the result array - meaning the first parameter resembles the number of entries in the array. For the Knapsack problems, we will add the number of items n as parameter and denote the running time of an algorithm via $T(n, \Pi(t+\vec{\mathbf{1}}), d)$, again using $\Pi(t+\vec{\mathbf{1}})$ to denote the number of possible knapsack capacities.

Similar to Cygan et al. [9], we mainly look at functions satisfying $T(\Pi(L), d) = c^{\mathcal{O}(d)} \cdot \Pi(L)^\alpha$ for some constants $c, \alpha > 0$. Therefore, we remark that for all constant $c' > 1$ we can write $T(\Pi(c' \cdot L), d) \in \mathcal{O}(T(\Pi(L), d))$ since d is fixed and we then have $\Pi(c'L)^\alpha = c'^{d \cdot \alpha} \cdot \Pi(L)^\alpha$.

For all the mentioned problems we assume that all inputs, that is also any value in any vector, consist of integers in the range of $[-W, W]$ for some $W \in \mathbb{Z}$. This W is generally omitted as a running time parameter and polylog(W) are omitted or hidden in T functions. We remark that – unavoidably – during the reductions we may have to deal with larger values than W. We generally will multiply a factor of polylog(λ) in cases where the values we have to handle increase up to λW. Note that if λ is a constant, we again omit it.

We follow the same order as Cygan et al. [9], because that makes the reduction increasingly more complex. For the first reduction from Unbounded d-Knapsack to $0/1$ d-Knapsack, we use the same reduction as in the one-dimensional case. The idea is to simply encode taking a multitude of items via binary encoding.

▶ **Theorem 9** (Unbounded d-Knapsack $\to$ $0/1$ d-Knapsack)**.** *A $T(n, \Pi(t), d)$ algorithm for 0/1 d-Knapsack implies an $\mathcal{O}(T(n \log(t_{\max}), \Pi(t), d))$ algorithm for Unbounded d-Knapsack, where $t \in \mathbb{N}^d$ is the capacity of the knapsack.*

For the next reduction we reduce d-SuperAdditivity Testing to a special case where each array entry is non-negative and values fulfill a monotony property.

▶ **Theorem 10** (d-SuperAdditivity Testing $\to$ Unbounded d-Knapsack)**.** *A $T(n, \Pi(t), d)$ algorithm for Unbounded d-Knapsack implies the existence an algorithm that solves d-SuperAdditivity Testing in time $\mathcal{O}(T(2\Pi(L), \Pi(2L), d) \operatorname{polylog}(dL_{\max}))$ for an array A with size L.*

The next reduction differs from Cygan et al. [9]. We also combine our input arrays together, but in the context of arrays we need to handle a number of different combinations more than in the one-dimensional case. We therefore add a block of negative values in an initial dummy block. This way, any combination that is not relevant to the actual upper bound test, will result positively when tested in the upper bound test.

▶ **Theorem 11** (d-MaxConv UpperBound $\to$ d-SuperAdditivity Testing)**.** *A $T(\Pi(L), d)$ algorithm for d-SuperAdditivity Testing implies an $\mathcal{O}(T(4\Pi(L), d))$ algorithm for d-MaxConv UpperBound.*

For the next reduction from d-MaxConv to d-MaxConv UpperBound, we will prove that we can use an algorithm for d-MaxConv UpperBound to also identify a position that violates the upper bound property.

▶ **Theorem 12** (d-MaxConv $\to$ d-MaxConv UpperBound)**.** *A $T(\Pi(L), d)$ algorithm for d-MaxConv UpperBound implies an $\mathcal{O}(2^d \Pi(L) \cdot T(2^d \sqrt{\Pi(L)}, d) \cdot d \cdot \log(L_{\max}))$ algorithm for d-MaxConv.*

The last reduction is based on Cygan et al. [9] and Bringmann [4]. To make their approach work even in higher dimensional cases, we require a new more refined distribution of items into layers. With this new partition of items many used techniques such as color-coding naturally translate into higher dimensions.

▶ **Theorem 13** ($0/1$ d-Knapsack $\to$ d-MaxConv)**.** *If d-MaxConv can be solved in time $T(\Pi(L), d)$ then 0/1 d-Knapsack can be solved with a probability of at least $1 - \delta$ in time $\mathcal{O}(T(\Pi(12t), d) \log(d \cdot t_{\max}) \log^3(\log n/\delta) \cdot d \cdot \log n)$ for any $\delta \in (0, 1)$.*

We remark that this also yields a respective algorithm for $0/1$ d-Knapsack with the $\Pi(L)^2$ algorithm for d-MaxConv. Lower order improvements like the results of Bremner [3] or Chan and Lewenstein [6] can be applied by calculating the convolution using Lemma 5.

3 Parameterized Algorithm for Multi-Dimensional Knapsack

In this part, we will consider solving Knapsack such that the capacity is completely utilized. This enables a concise presentation and is not a restriction as discussed after the concrete problem definition.

Bounded d-EqualityKnapsack	
Input:	Set I of n items each defined by a profit $p_i \in \mathbb{R}_{\geq 0}$, weight vector $w^{(i)} \in \mathbb{N}^d$, and upper bound $u_i \in \mathbb{N}$ along with a knapsack of capacity $t \in \mathbb{N}^d$.
Problem:	Find a vector $x \in \mathbb{N}^n$ such that $x \leq u$, $\sum_{i=1}^{n} x_i w^{(i)} = t$, and $\sum_{i=1}^{n} x_i p_i$ is maximal.

We will not only solve this problem for the given capacity t but for all smaller capacities. In detail, we will construct a solution array A with size $t + \vec{\mathbf{1}}$ such that for any $v \leq t$ we have that A_v is the maximum profit of an item set whose total weight is exactly v. Note that we can construct a solution for Bounded d-Knapsack after solving Bounded d-EqualityKnapsack by remembering the highest profit achieved in any position.

The algorithm we will show is based on the algorithm of Axiotis and Tzamos [1]. They solved one-dimensional Knapsack by splitting all items into subsets with equal weight. They proceed then to solve these resulting sub-instances. These sub-instances can be easily solved by gathering the highest profit items that fit into the knapsack. Finally, they combine these resulting partial solutions via convolution. The profits of one such partial solution forms a concave sequence, which allows each convolution to be calculated in linear time.

▶ **Definition 14** (Concave Sequences). *Let $a = (a_i)_{i \leq n}$ be a sequence of numbers of length n. We call the sequence a* concave *if for all $i \leq n-2$ we have $a_{i+1} - a_i \geq a_{i+2} - a_{i+1}$.*

▶ **Lemma 15** ([1, Lemma 9]). *Given an arbitrary sequence of length m and a concave sequence of length n we can compute the $(\max, +)$ convolution of a and b in time $\mathcal{O}(m+n)$.*

We achieve a similar result and calculate higher dimensional convolutions in linear time. In fact, we will reduce our problem to calculating convolution of one-dimensional concave sequences. This way we can calculate the convolution of our sub-solutions in linear time in the number of entries in our array.

▶ **Theorem 6.** *There is an algorithm for Bounded d-EqualityKnapsack with running time $\mathcal{O}(dn + d \cdot D \cdot \max\{\prod(t+\vec{\mathbf{1}}), t_{\max} \log t_{\max}\}) \subseteq \mathcal{O}(dn + d \cdot \min\{n, (\Delta+1)^d\} \cdot \max\{\prod(t+\vec{\mathbf{1}}), t_{\max} \log t_{\max}\}))$.*

We remark, that in most cases $V := \prod(t + \vec{\mathbf{1}}) > t_{\max} \log t_{\max}$ so generally this algorithm will have a running time of $\mathcal{O}(n + DV)$. However, in the special case of a slim array that has size t with one large value t_j and other values being very small $t_i \ll t_j$ for $i \neq j$ our algorithm might have a slightly worse running time of $\mathcal{O}(n + Dt_{\max} \log t_{\max})$. Note that is algorithm also works for d-Knapsack by setting every upper bound to $t_{\max}$.

Proof of Theorem 6. Since we have D different weights, let $w(1), \ldots, w(D)$ be the different weights. We partition the items I into D sets of the same weight, $I_{w(i)} := \{j \in I : w(i) = w^{(j)}\}$ for all $i \in [D]$. We start with an empty solution array R^0 of size $t + \vec{\mathbf{1}}$ such that $R^0_{\vec{\mathbf{0}}} = 0$ and $R^0_v = -\infty$ for $v \neq \vec{\mathbf{0}}$. We will calculate solution arrays R^i for $i \in [D]$ where R^i is the solution array for instance restricted to the items $\bigcup_{j=1}^{i} I_{w(j)}$. Next we describe how we can calculate R^i from R^{i-1}.

Consider an optimal solution for some position v in R^i, i.e., containing only items from $\bigcup_{j=1}^{i} I_{w(j)}$. The solution contains a certain number of items of weight $w(i)$. Let this number be k. We have $v' \coloneqq v - kw(i) \geq \vec{\mathbf{0}}$. Thus, the other items in the solution are from $\bigcup_{j=1}^{i-1} I_{w(j)}$ and have combined profit $R^{i-1}_{v'}$ otherwise our initial solution is not optimal or R^{i-1} is not correct. Also, the k items with weight $w(i)$ in the optimal solution have the highest profit possible, otherwise the solution is not optimal. Therefore, we have

$$R^i_v = \max_{\substack{k \geq 0 \\ v - k \cdot w(i) \geq \vec{\mathbf{0}}}} R^{i-1}_{v - k \cdot w} + f_w(k) \tag{1}$$

where $f_{w(i)}(k)$ is the largest profit for taking k items of weight $w(i)$. Hence, only position pairs v, u such that their difference is an integer multiple of $w(i)$ ($v - u \in w(i)\mathbb{Z}$) are relevant for the calculation of R^i_v or R^i_u. This condition forms an equivalence relation and thus we can calculate the values of R^i per equivalence class. Every equivalence class has the structure $\{v, v + w(i), v + 2 \cdot w(i), v + k_v \cdot w(i)\}$ with $v - w(i) \not\geq \vec{\mathbf{0}}$. Define the sequences $(r_j)_{j \leq k_v}$ with $r_j \coloneqq R^{i-1}_{v + j \cdot w(i)}$ and $(a_j)_{j \leq k_v}$ with $a_j \coloneqq f_{w(i)}(j)$. Thus, we can calculate R^i by $R^i_{v + j \cdot w(i)} \coloneqq (r \oplus a)_j$ for every $j \in [k_v]$. This is sufficient by Equation (1). Note that, we only have to calculate the convolution once as we can use the result for every element of the equivalence class. We always have $k_v \leq t_{\max}$ and $(a_j)_{j \leq k_v}$ is a concave sequence.

The items can be partitioned in the sets $I_{w(i)}$ in time $\mathcal{O}(dn + \Delta)$. In every set the $t_{\max}$ most profitable items can be calculated in total time $\mathcal{O}(n)$ by using [2]. Calculating all values for $f_{w(i)}(j)$ with $i \in [D]$ and $j \in t_{\max}$ takes time $\mathcal{O}(Dt_{\max} \log(t_{\max}))$. Every convolution can be calculated in linear time by Lemma 15. Since the equivalence classes partition the set of positions, we can calculate R^i from R^{i-1} in time $d \cdot \mathcal{O}(\max\{V, t_{\max} \log(t_{\max})\})$ where the d factor is due to multidimensional indices. We may assume $\Delta \leq t_{\max}$ which shows the claimed running time. ◀

3.1 Generalisation to ILPs

As we mentioned before, Knapsack is a special case of ILP. We now want to discuss how to extend our results to ILPs. In the case that all matrix entries are non-negative ILP is equivalent to Bounded d-EqualityKnapsack. We discuss later how to handle negative entries in the constraint matrix. Negative item profits are sensible in Bounded d-EqualityKnapsack as those may be required to reach the capacity. The algorithm presented in Theorem 6 also works with negative profits. The running time of Theorem 6 depends on the capacity of the knapsack. To bound this value for ILPs we use the proximity bound by Eisenbrand and Weismantel [11] and improve their algorithmic approach. We start by giving a short summary of their approach.

For an ILP Instance with constraint matrix $A \in \mathbb{Z}^{d \times n}$, right side $b \in \mathbb{Z}^d$, and upper bounds $u \in \mathbb{N}^n$, let $\Delta \coloneqq \|A\|_{\max}$ be the biggest absolute value of an entry in the constraint matrix. Eisenbrand and Weismantel [11] calculate an optimal solution x^* for the LP-relaxation, which is possible in polynomial time. They proved that there exists an integral optimal solution z^*, such that the distance in ℓ_1-norm is small, more precisely $\|z^* - x^*\|_1 \leq d \cdot (2d\Delta + 1)^d$. They then proceed to take $\lfloor x^* \rfloor$ as an integral solution ($\|z^* - \lfloor x^* \rfloor\|_1 \leq d \cdot (2d\Delta + 1)^d + d =: L$) and construct the following ILP to find an optimal solution.

$$\begin{aligned} \max c^\mathsf{T} x \text{ subject to} \\ Ax &= b - A\lfloor x^* \rfloor \\ \|x\|_1 &\leq L \\ \ell' \leq x &\leq u' \\ x &\in \mathbb{Z}^n \end{aligned}$$

Note that we may have $\ell'_i < 0$ for some $i \in [n]$. Combining an optimal solution to this ILP with $\lfloor x^* \rfloor$ yields an optimal solution to the original ILP. For any $x \in \mathbb{Z}^n$ with $\|x\|_1 \leq L$ we have $\|Ax\|_\infty \leq \Delta L \leq \mathcal{O}(d\Delta)^{d+1}$ due to the bound on the ℓ_1-norm. Eisenbrand and Weismantel construct an acyclic directed graph and find the longest path in that graph which is equivalent to an optimal solution to the ILP above. They bound the size of the graph using the bound on the ℓ_∞-norm above. We avoid such a graph construction for our algorithm which allows us to improve the running time. For two $i, j \in [n]$ with $A_i = A_j$ and $(c_i, i) > (c_j, j)$ we may assume that $x^*_j > 0$ implies $x^*_i = u_i$. Otherwise, we can modify x^* to obtain a solution that adheres to this property with at least the same value. This structure allows us to reformulate the ILP by grouping variables together with the same column in A as follows

$$\begin{aligned} \max \sum\nolimits_{i \in [D]} f_i(x_i) \text{ subject to} \\ A'x &= b - A\lfloor x^* \rfloor \\ \|x\|_1 &\leq L \\ \ell'' \leq x &\leq u'' \\ x &\in \mathbb{Z}^D \end{aligned}$$

where f_i is a concave function for all $i \in [D]$. We calculate an optimal solution to this ILP using the convolution approach from Theorem 6.

▶ **Theorem 7.** *There is an algorithm for ILP with running time $\mathcal{O}(dn) + D \cdot \mathcal{O}(d\Delta)^{d(d+1)} + T_{\text{LP}}$.*

Proof. For $i \in [D]_0$ let U^i be the best possible values using the variables x_j for $j \leq i$. The size of U^i in every dimension is $2\Delta L + 1$. To calculate U^i_v from U^{i-1} for some position $v \in \mathbb{Z}^d$ such that $\|v\|_\infty \leq \Delta L$ we need to calculate

$$U^i_v = \max_{\substack{\ell''_i \leq j \leq u''_i \\ \|v - jA'_i\|_\infty \leq \Delta L}} U^{i-1}_{v - jA'_i} + f_i(j).$$

As in the proof of Theorem 6 we can consider the equivalence classes. Let S be the equivalence class of v, it has the structure $\{v', v' + A'_i, v' + 2A'_i, \ldots, v' + (|S|-1)A'_i\}$. Define the sequence $(a_j)_{j \leq 2|S|-2}$ with $a_j \coloneqq U^{i-1}_{v' + jA'_i}$ for $j < |S|$ and $a_j \coloneqq \infty$ for $j \geq |S|$. Further we define the sequence $(b_j)_{j \leq 2|S|-2}$ with $b_j \coloneqq f_i(j - |S| + 1)$. We calculate the convolution of the sequences $c \coloneqq a \oplus b$ in time $\mathcal{O}(|S|)$ using Lemma 15. Then we get

$$\begin{aligned} U^i_{v' + jA'_i} &= \max_{\substack{\ell''_i \leq k \leq u''_i \\ \|v' + jA'_i - kA'_i\|_\infty \leq \Delta L}} U^{i-1}_{v' + jA'_i - kA'_i} + f_i(k) \\ &= \max_{j - |S| + 1 \leq k \leq j} U^{i-1}_{v' + (j-k)A'_i} + f_i(k) \\ &= \max_{j \leq k \leq j + |S| - 1} U^{i-1}_{v' + (j-k+|S|-1)A'_i} + f_i(k - |S| + 1) \\ &= \max_{0 \leq k \leq |S|-1} U^{i-1}_{v' + (j-k-j+|S|-1)A'_i} + f_i(k + j - |S| + 1) \\ &= \max_{0 \leq k \leq |S|-1} U^{i-1}_{v' + (|S|-1-k)A'_i} + f_i(j + k - |S| + 1) \\ &= \max_{0 \leq k \leq |S|-1} U^{i-1}_{v' + kA'_i} + f_i(j - k) \\ &= \max_{0 \leq k \leq |S|-1} U^{i-1}_{v' + kA'_i} + f_i(j - k + |S| - 1 - |S| + 1) \end{aligned}$$

$$
\begin{aligned}
&= \max_{0\le k\le |S|-1} a_k + b_{j-k+|S|-1} \\
&= \max_{0\le k\le j+|S|-1} a_k + b_{j-k+|S|-1} && (a_k = -\infty \text{ for } k \ge |S|) \\
&= c_{j+|S|-1}
\end{aligned}
$$

for $j \in [|S|-1]_0$. Next discuss how to obtain the running time of $\mathcal{O}(d\Delta)^{d(d+1)}$ instead of the expected $d\mathcal{O}(d\Delta)^{d(d+1)}$ as in the proof of Theorem 6. To do this we add buffer zones of size Δ at the start and the end of every dimension. Then the size for every dimension is still bounded by $\mathcal{O}(d\Delta)^{d+1}$. Next we save the d-dimensional array as a 1-dimensional array by concatenating the dimensions. We mark the entries in the buffer zones. For a fixed column we can calculate a step size in the 1-dimensional array that corresponds to a step with the column in the d-dimensional array. Using the marked buffer zones we can check if we stepped outside the valid area which allows us to find the equivalence class of a position in linear time of the size of the class. Therefore, we can find all equivalence classes in time $\mathcal{O}(d\Delta)^{d(d+1)}$, and we can calculate the convolution in the same time.

First we need to solve the linear relaxation of the ILP. Denote the necessary time by T_{LP}. Next we partition the columns by weight in time $\mathcal{O}(dn)+\Delta$. We can find the L most profitable picked and unpicked items for every column type in total time $\mathcal{O}(n)$ using [2]. With this we can calculate all relevant values for every f_i in time $D \cdot L \log L$. As explained above every convolution can be calculated in time $\mathcal{O}(d\Delta)^{d(d+1)}$ which yields the claimed running time. An optimal solution vector can be calculated by backtracking in time $D \cdot L \le D \cdot d \cdot \mathcal{O}(d\Delta)^{d+1}$ ◀

4 Divide and Conquer Algorithm

In this section we show Theorem 8. We start by showing the central structural property.

▶ **Lemma 16.** *Let $A \in \mathbb{Z}^{d\times n}$, $b \in \mathbb{Z}^d$, and $x, u \in \mathbb{N}^d$. If $Ax = b$ and $x \le u$, there exists $x' \in \mathbb{N}^d$ such that*

$$\|2Ax' - b\|_\infty \le 2n\Delta, \quad x' \le u' := \left\lfloor \frac{u-1}{2} \right\rfloor \quad \text{and} \quad 0 \le x_i - 2x'_i \le u_i - 2u'_i \le 2 \text{ for } i \in [n].$$

Proof. Let $0 \le x \le u$ with $Ax = b$. We define the new solution x' component wise as follows.

$$x'_i := \begin{cases} 0 & \text{if } x_i = 0, \\ \lfloor \frac{x_i}{2} \rfloor & \text{if } u_i \text{ is odd}, \\ \lfloor \frac{x_i - 1}{2} \rfloor & \text{otherwise} \end{cases}$$

for $i \in [n]$. We start by showing $x' \le u'$. Let $i \in [n]$. If u_i is odd, we have

$$x'_i = \left\lfloor \frac{x_i}{2} \right\rfloor \le \frac{x_i}{2} \le \frac{u_i}{2} = \frac{2u'_i + 1}{2} = u'_i + \frac{1}{2} \implies x'_i \le u'_i$$

and

$$0 \le x_i - 2x'_i = x_i - 2\left\lfloor \frac{x_i}{2} \right\rfloor \le x_i - 2\frac{x_i - 1}{2} = 1 = u_i - 2u'_i \le 2.$$

Otherwise, if u_i is even, we have

$$x'_i = \left\lfloor \frac{x_i - 1}{2} \right\rfloor \le \frac{x_i - 1}{2} \le \frac{u_i - 1}{2} = \frac{2u'_i + 1}{2} = u'_i + \frac{1}{2} \implies x'_i \le u'_i$$

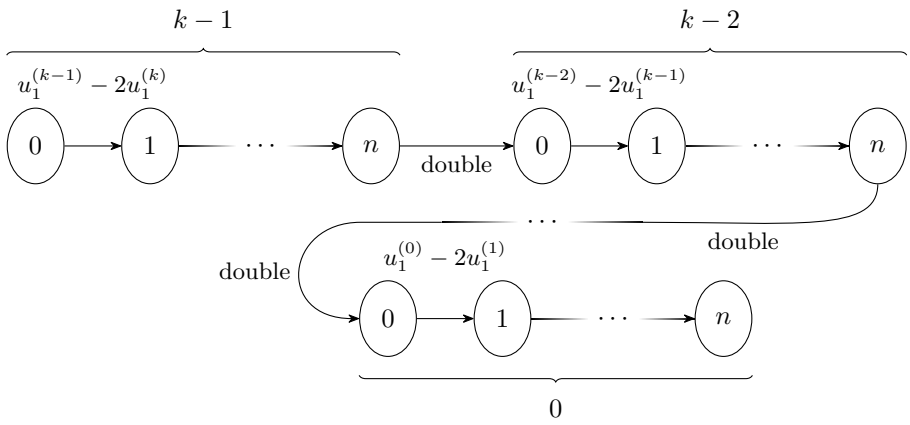


Figure 1 Structure of the graph in the proof of Theorem 8.

and

$$0 \leq x_i - 2x'_i = x_i - 2\left\lfloor \frac{x_i - 1}{2} \right\rfloor \leq x_i - 2\frac{x_i - 2}{2} = 2 = u_i - 2u'_i \leq 2.$$

Next, let $b' \coloneqq Ax'$. We have

$$|2b'_j - b_j| = \left|\sum_{i=1}^{n} A_{j,i}(2x'_i - x_i)\right| \leq \sum_{i=1}^{n} |A_{j,i}(2x'_i - x_i)| = \sum_{i=1}^{n} |A_{j,i}||2x'_i - x_i| \leq 2n\Delta$$

for every $j \in [d]$ and thus $\|2b' - b\|_\infty \leq 2n\Delta$. ◀

This lemma implies that *every* solution $x \in \mathbb{N}^n$ to $Ax = b$ with $x \leq u$ can be decomposed into $x', x'' \in \mathbb{N}^n$ such that $x = 2x' + x''$, x' is a solution to $Ax = b'$ and $x'' \leq u - 2u'$. Note that u' does not depend on the concrete solution but is the same for all.

We can iterate Lemma 16 to obtain a series of solutions and upper bounds until we reach a point where all upper bounds are zero. Define $x^{(j)} \in \mathbb{N}^n$ and $u^{(j)} \in \mathbb{N}^n$ after applying Lemma 16 j times, i.e., $x^{(0)} = x$ and $u^{(0)} = u$. Let $k \in \mathbb{N}$ be smallest value such that $u^{(k)} = 0$.

▶ **Corollary 17.** *We have* $\left\|Ax^{(j)} - \frac{b}{2^j}\right\|_\infty \leq n\Delta \sum_{i=1}^{j} \frac{1}{2^{i-1}} \leq 2n\Delta$ *for* $j \leq k$.

Proof. By induction. $j = 1$ is Lemma 16. Let $j > 1$. Then, we have

$$\begin{aligned}
\left\|Ax^{(j)} - \frac{b}{2^j}\right\|_\infty &= \left\|Ax^{(j)} - \frac{Ax^{(j-1)}}{2} + \frac{Ax^{(j-1)}}{2} - \frac{b}{2^j}\right\|_\infty \\
&\leq \left\|Ax^{(j)} - \frac{Ax^{(j-1)}}{2}\right\|_\infty + \frac{1}{2}\left\|Ax^{(j-1)} - \frac{b}{2^{j-1}}\right\|_\infty \\
&\leq n\Delta + \frac{n\Delta}{2}\sum_{i=1}^{j-1}\frac{1}{2^{i-1}} = n\Delta + n\Delta\sum_{i=2}^{j}\frac{1}{2^{i-1}} = n\Delta\sum_{i=1}^{j}\frac{1}{2^{i-1}}.
\end{aligned}$$

◀

With this we can give the algorithm and proof Theorem 8.

▶ **Theorem 8.** *There is an algorithm that can solve ILP in time* $n^{d+1} \cdot \mathcal{O}(\Delta)^d \cdot \log(\|u - \ell\|_\infty)$.

Proof. We find an optimal solution to the bounded ILP by finding the longest path in a graph that is constructed based on Lemma 16. We define the graph as follows.

$$V \coloneqq \left\{(b', j, i) : j \in [k]_0, \left\|b' - \frac{b}{2^j}\right\|_\infty \leq 2(2n - i)\Delta, i \in [n]_0\right\}$$

The intuition here is that we have a layer for every application of Lemma 16. We can jump from one layer to the next by doubling the corresponding right side (the first component) and length of the path to the current vertex. Then, to reconstruct the solution prior to an application of Lemma 16 we may need to increase variables by up to two. For this we have the third component. While going to a vertex with $i \in [n]$ in the last component we may use the ith column up to the number of times it was removed to get the new upper bound. This is also an upper bound for the number of times it was removed from a solution as stated in Lemma 16. More precisely, we associate the ILP

$$\begin{aligned} \max\, & c^\mathsf{T} x \\ & Ax = b' \\ & x_\ell \leq u_\ell^{(j)} && \text{(for } \ell \in [i]\text{)} \\ & x_\ell \leq 2u_\ell^{(j+1)} && \text{(for } \ell > i\text{)} \\ & x \in \mathbb{N}^n \end{aligned}$$

with a vertex $(b', j, i) \in V$ and refer to it by $\mathrm{ILP}_{(b',j,i)}$. A path from $(0, k-1, 0)$ to (b', j, i) describes a solution to $\mathrm{ILP}_{(b',j,i)}$ where the value is equal to the length of the path.

Next we give the precise definitions for the edges in the graph. For a vertex with $j < k$ and $i < n$ we define the following edges

$$(b', j, i) \xrightarrow{xc_{i+1}} (b' + xA_{i+1}, j, i+1)$$

with $x \in [u_{i+1}^{(j)} - 2u_{i+1}^{(j+1)}]_0 \subseteq [2]_0$ and for vertices with $j > 0$ and $i = n$ we define the edge

$$(b', j, n) \xrightarrow{\text{double the length}} (2b', j-1, 0).$$

Because of the changes in the second and third component the graph is acyclic, and thus the longest path can be calculated in linear time by utilizing a topological order of the graph. Note that the second type of edge does not have a classic length, but this doubling of the current length works with the aforementioned algorithm. The out degree of every vertex is bounded by 3 and thus the size of graph is linear in the number vertices which is $n^{d+1} \cdot \mathcal{O}(\Delta)^d \cdot \log(\|u\|_\infty)$. Now we find the longest path from $(0, k-1, 0)$ to $(b, 0, n)$. The only solution to $\mathrm{ILP}_{(0,k-1,0)}$ is $\vec{\mathbf{0}}$. The structure of the graph is depicted in Figure 1.

Let $x \in \mathbb{N}^n$ be a solution to the ILP. Further, let $x^{(j)} \in \mathbb{N}^n$ be the solutions obtained by iterating Lemma 16. Define

$$x^{(j,i)} := \begin{cases} 2x^{(j+1)} & \text{if } i = 0, \\ x^{(j,i-1)} + \left(x_i^{(j)} - 2x_i^{(j+1)}\right) e_i & \text{otherwise} \end{cases}$$

for $j \in [k-1]_0$ and $i \in [n]_0$ where e_i is the ith unit vector. We have $\left\| Ax^{(j,i)} - \frac{b}{2^j} \right\|_\infty \leq 2(2n-i)\Delta$ by Lemma 16 and Corollary 17. Thus,

$$\begin{aligned} & (0 = Ax^{(k-1,0)}, k-1, 0), (Ax^{(k-1,1)}, k-1, 1), \ldots, \\ & \quad (Ax^{(k-1,n)}, k-1, n), (Ax^{(k-2,0)}, k-2, 0), \ldots, (b = Ax^{(0,n)}, 0, n) \end{aligned}$$

is a path in the graph with length $c^\mathsf{T} x$. Therefore, ILP has a solution if and only if there is a path in the graph from $(0, k-1, 0)$ to $(b, 0, n)$. Also the value of the optimal solution to the ILP is equal to the length of the longest path in the graph. ◀

5 Conclusion

First, we have proven that the relationship between Knapsack and Convolution in the one-dimensional case also prevails in higher dimensional variants. A natural follow-up question is how many more problems can be shown to be equivalent to d-dimensional convolution.

We developed a parameterized algorithm for multidimensional knapsack that generalized techniques for one-dimensional knapsack. Further, we used these techniques to obtain a faster algorithm for Integer Linear Programming with upper bounds.

Finally, we gave an improved algorithm for Integer Linear Programming with upper bounds that avoids the quadratic dependency on the dimension by increasing the dependency on the number of variables.

References

1 Kyriakos Axiotis and Christos Tzamos. Capacitated Dynamic Programming: Faster Knapsack and Graph Algorithms. In *46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132, pages 19:1–19:13, Dagstuhl, Germany, 2019. doi:10.4230/LIPICS.ICALP.2019.19.

2 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.

3 David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, and Perouz Taslakian. Necklaces, convolutions, and x + y. In *Algorithms – ESA 2006*, pages 160–171, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. doi:10.1007/11841036_17.

4 Karl Bringmann. *A Near-Linear Pseudopolynomial Time Algorithm for Subset Sum*, pages 1073–1084. Society for Industrial and Applied Mathematics, 2017. doi:10.1137/1.9781611974782.69.

5 Marcel Celaya, Stefan Kuhlmann, Joseph Paat, and Robert Weismantel. Improving the cook et al. proximity bound given integral valued constraints. In *Integer Programming and Combinatorial Optimization - 23rd International Conference, IPCO 2022, Eindhoven, The Netherlands, June 27-29, 2022, Proceedings*, volume 13265 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2022. doi:10.1007/978-3-031-06901-7_7.

6 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3sum via additive combinatorics. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 31–40, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2746539.2746568.

7 Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. Faster algorithms for bounded knapsack and bounded subset sum via fine-grained proximity results. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 4828–4848. SIAM, 2024. doi:10.1137/1.9781611977912.171.

8 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 1529–1542, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520057.

9 Marek Cygan, Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. On problems equivalent to $(min, +)$-convolution. *ACM Trans. Algorithms*, 15(1), January 2019. doi:10.1145/3293465.

10 Ilan Doron-Arad, Ariel Kulik, and Pasin Manurangsi. Fine grained lower bounds for multidimensional knapsack, 2024. doi:10.48550/arXiv.2407.10146.

11 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the steinitz lemma. *ACM Trans. Algorithms*, 16(1), November 2019. doi:10.1145/3340322.

12 Kilian Grage, Klaus Jansen, and Björn Schumacher. Convolution and knapsack in higher dimensions, 2024. arXiv:2403.16117.

13 Dmitry V. Gribanov, Ivan A. Shumilov, and Dmitriy S. Malyshev. Structured $(\min, +)$-convolution and its applications for the shortest/closest vector and nonlinear knapsack problems. *Optim. Lett.*, 18(1):73–103, 2024. doi:10.1007/S11590-023-02017-5.

14 Ce Jin. 0-1 knapsack in nearly quadratic time. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 271–282. ACM, 2024. doi:10.1145/3618260.3649618.

15 L. Kronecker. Grundzüge einer arithmetischen theorie der algebraische grössen. *Journal für die reine und angewandte Mathematik*, 1882(92):1–122, 1882. doi:10.1515/crll.1882.92.1.

16 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.ICALP.2017.21.

17 Jon Lee, Joseph Paat, Ingo Stallknecht, and Luze Xu. Improving proximity bounds using sparsity. In *Combinatorial Optimization - 6th International Symposium, ISCO 2020, Montreal, QC, Canada, May 4-6, 2020, Revised Selected Papers*, volume 12176 of *Lecture Notes in Computer Science*, pages 115–127. Springer, 2020. doi:10.1007/978-3-030-53262-8_10.

18 Victor Y. Pan. Simple multivariate polynomial multiplication. *J. Symb. Comput.*, 18(3):183–186, 1994. doi:10.1006/JSCO.1994.1042.

19 Adam Polak, Lars Rohwedder, and Karol Węgrzycki. Knapsack and Subset Sum with Small Items. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 106:1–106:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.106.

20 Lars Rohwedder and Karol Węgrzycki. Fine-Grained Equivalence for Problems Related to Integer Linear Programming. In *16th Innovations in Theoretical Computer Science Conference (ITCS 2025)*, volume 325 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 83:1–83:18, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2025.83.

21 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 664–673, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591811.

A Calculate d-dimensional Convolution with 1-dimensional Convolution

We can interpret $(\max, +)$-convolution as multiplication of polynomials over the $(\max, +)$-semiring, also called the Arctic semiring. d-dimensional convolution is multiplication of polynomials with d variables $x_1, \ldots, x_d$. More precisely, let A, B be two d-dimensional arrays of size L. Then, the corresponding polynomial to A is given by

$$p(A) := \sum_{0 \leq v < L} A[v] \prod_{i=1}^{d} x_i^{v_i}.$$

The Kronecker map [15] as described in [18] maps x_{k+1} to $x^{D(1)\cdots D(k)}$ for every $k \in [d-1]_0$ where $D(i) = 2L_i - 1$ for every $i \in [d]$. This map is generalized to polynomials accordingly and allows us to obtain an one-dimensional array this way by adding dummy values. An example for this construction is depicted in Figure 2. Let $\tilde{p}(A)$ and $\tilde{p}(B)$ be the results of this transformation.

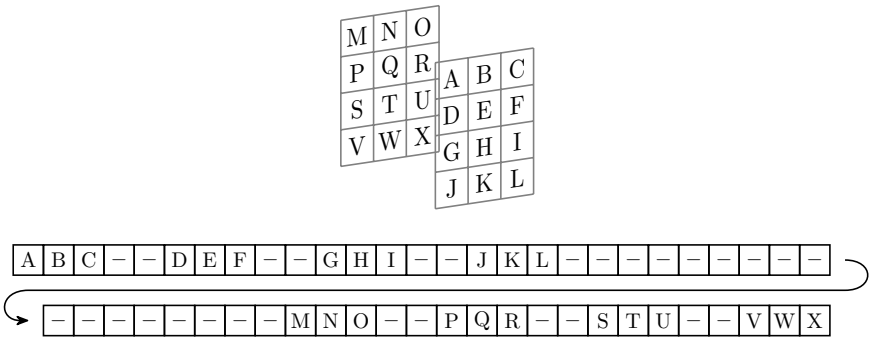


Figure 2 Example for the transformation in 3 dimensions with $L = (3, 4, 2)^\mathsf{T}$.

Now, we can calculate $\tilde{p}(A) \cdot \tilde{p}(B)$ with an one-dimensional $(\max, +)$-convolution algorithm and transform the result back according to the transformations above to obtain an d-dimensional array. The result is correct since the polynomial multiplication is correct due to the added padding in the definition of $D(i)$ [18].

To show Lemma 5 is remains to show that $\deg(\tilde{p}(A)) + 1 \leq 2^d \prod(L)$ since $\deg(\tilde{p}(A))$ is the length of the resulting arrays for the one-dimensional convolution. We have

$$
\begin{aligned}
2\deg(\tilde{p}(A)) &= 2\deg\Big(A[L - \vec{\mathbf{1}}] \prod_{i=1}^{d} (x^{D(1)\cdots D(i-1)})^{L_i - 1}\Big) \quad (L - \vec{\mathbf{1}} \text{ is the maximum index.}) \\
&= 2\sum_{\ell=1}^{d} (L_\ell - 1) \prod_{m=1}^{\ell-1} D(m) \\
&\leq \sum_{\ell=1}^{d} \Big(\prod_{m=1}^{\ell-1} 2L_m\Big) 2L_\ell \\
&= 2L_1 + 2L_1 \sum_{\ell=2}^{d} \Big(\prod_{m=1}^{\ell-1} 2L_m\Big) 2L_\ell \\
&= \sum_{\ell=1}^{d} 2^\ell \prod_{m=1}^{\ell} L_m \qquad \text{(With induction)} \\
&= \Big(\sum_{\ell=1}^{d} 2^\ell\Big) \prod(L) \\
&\leq 2^{d+1} \prod(L) \\
\implies \deg(\tilde{p}(A)) &\leq 2^d \prod(L)
\end{aligned}
$$

In conclusion we have show the following lemma.

▶ **Lemma 5.** *Given an algorithm for 1-dimensional convolution with running time $T(n)$ and two d-dimensional arrays A, B with size L, we can calculate $A \oplus B$ in time $T(2^d \prod(L))$.*

Repairing Schedules by Removing Waiting Times: A Parameterized Complexity Analysis

Niels Grüttemeier
Fraunhofer IOSB-INA, Lemgo, Germany

Klaus Heeger
Fraunhofer IOSB-INA, Lemgo, Germany

Abstract

We consider the problem of repairing production schedules in a job-shop setting by reducing pre-planned waiting times. Herein, a schedule of all jobs is given. To compensate unforeseen disturbances, this schedule contains waiting times between the execution of two consecutive tasks of a job. Further, we assume that the schedule temporarily overloads some machines, e.g. due to reduced machine capacities because of worker sickness or (partially) broken machines. We study the problem of removing as few waiting times as possible in order to eliminate the machine overloads. After formalizing this problem, we perform an extensive analysis of its parameterized complexity with respect to several natural parameters, resulting in a detailed picture of the problem's complexity.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Problems, reductions and completeness; Mathematics of computing → Combinatorics

Keywords and phrases Job shop, parallel machines, reactive scheduling

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.31

Funding *Niels Grüttemeier*: Supported by the project *Datenfabrik.NRW*, a project by *KI.NRW*, funded by the Ministry for Economics, Innovation, Digitalization and Energy of the State of North Rhine-Westphalia (MWIDE).
Klaus Heeger: Supported by the German Federal Ministry for the Environment, Nature Conservation, Nuclear Safety and Consumer Protection (BMUV) under the project *Smart-E-Factory*.

1 Introduction

Scheduling is a crucial task in the industrial production of goods, helping to achieve key goals such as low production costs or on-time deliveries. Therefore, scheduling is an established and very important area in the intersection of theoretical computer science, operations research, and combinatorial optimization. In classical scheduling problems, one is usually given a set of jobs (and their characteristics) to be processed in the future, and the task is to compute an "optimal" (with respect to some specified criterium) schedule of the jobs. However, many real-world scenarios are not adequatly covered by this approach, for instance when a large number of unexpected obstacles (e.g., machines breaking down, workers calling in sick, arrival of urgent customer orders, etc.) may compromise the initial plan. Recomputing a new schedule from scratch after each such obstacle is hardly an option in practice, as reassignments of jobs and operations to different resources require cancellations of expensive set-ups and lead to shop floor nervousness [21].

Instead of rescheduling, there are also other ways to handle unexpected obstacles in production planning. One such approach is scheduling under uncertainties, containing several different models of uncertainties in scheduling. One of the most common such models is stochastic scheduling, where the values of job parameters (most commonly of the processing time) are independent random variables instead of numbers, see e.g. [8, 19]. Another model is multi-scenario scheduling, where a finite number of different scenarios needs to be considered

© Niels Grüttemeier and Klaus Heeger;
licensed under Creative Commons License CC-BY 4.0

t	1	2	3	4	5	6	7	8
A	3	2	2	2	2	2	0	1
B	1	1	2	0	2	2	1	1
1					A	x	x	B
2					A	B		
3			A	x	A	B		
4	A	x	x	A	B			
5	A	A	x	A	B			

→

t	1	2	3	4	5	6	7	8
A	3	2	2	2	2	2	0	1
B	1	1	2	0	2	2	1	1
1					A	x	x	B
2					A	B		
3			A	A	B			
4	A	A	B					
5	A	A	x	A	B			

Figure 1 An example of an instance of WTR (left) together with a repaired schedule (right). There are time steps $t = 1, \ldots, 8$ and jobs $1, \ldots, 5$ that are processed on two machines A and B. The upper part shows the machine capacities at the time steps, the lower part shows the jobs aligned at their starting time steps. At time step $t = 5$, the capacity constraints are violated, since three jobs are processed on machine A. Removing the grey marked symbols x in Job 3 and Job 4 converts the schedule into a schedule satisfying all capacity constraints.

(see [18] for a survey). Another approach is online scheduling, where the machine breakdowns appear one after another and one has to update the schedule immediately, see e.g. [2, 10]. Another approach is to *repair* a schedule. In this approach, one starts with a given schedule and applies small changes to it, adapting the schedule to the new situation. This is done by heuristically applying generic operations on the given schedule [20, 21] or by applying data-driven techniques from the field of machine learning [14, 16, 22]. Relatively new approaches combine scheduling under uncertainties with repairing policies [13]. For a very recent survey of scheduling problems in general, we refer to Agnetis et al. [1].

While the vast majority of research on repairing schedules focusses on heuristic approaches and their experimental evaluation, this work aims to lay a foundation for the theoretical study of these problems. In order to do so, we first formalize our scheduling problem as a precise mathematical problem which we call Waiting Time Removal (WTR) using notations from stringology. The idea of WTR is that in a case where unexpected resource breakdowns occur, updated information about resource availability and job durations is perfectly known and one has to adapt the old solution to the new situation. WTR is motivated by real-world production planning scenarios in German mid-sized companies. In this work, we study the computational complexity of WTR, focussing on the parameterized complexity.

An example of an instance of WTR together with a repaired schedule is illustrated in Figure 1. Intuitively, our scheduling problem WTR models the following scenario: there are multiple types of different machines (e.g. machine types A and B). Similar to a job shop, each job has to be processed on machines of different types in a job-specific order for a specific amount of time. However, in contrast to the standard job shop model, there may be waiting times planned between the steps, serving as a "buffer" to cover unexpected delays. To model this, each job is represented by a string enlisting the required machines (repeating a machine if it needs to be processed for multiple time units) and the planned waiting times. As an example, the string $AAxB$ corresponds to a job that first needs to be processed on machine A for two time units, then has one time unit waiting time (encoded by the letter "x"), and finally needs to be processed on machine B for one time unit. At each time step, a different number of machines of each type is available (e.g. at time $t = 1$, three machines of type A might be available, meaning that up to three different jobs can be processed on machines of type A); this number will be called the *capacity* of the machine at that time step. The capacities model e.g. the breakdown of machines or workers required to operate the machine calling in sick. Additionally, each job is assigned with a starting time. The goal is to remove a small number of waiting times from the jobs such that the resulting

Table 1 Our results. Each cell corresponds to parameterization by the sum of the two corresponding values. The diagonal cells correspond to parameterization by n, k, $|\Sigma|$, or T alone.

	n	k	$\lvert\Sigma\rvert$	T
n	XP (Thm. 4) W[1]-h (Thm. 3)			
k	FPT (Cor. 5) No Poly Ker. (Cor. 10)	XP (Obs. 2) W[2]-h (Cor. 9)		
$\lvert\Sigma\rvert$	XP (Thm. 4) W[1]-h (Thm. 3)	XP (Obs. 2) W[2]-h (Cor. 9)	Para NP-h (Thm. 3)	
T	FPT (Obs. 1) Poly Kernel (Obs. 1)	XP (Obs. 2) W[1]-h (Thm. 11)	FPT (Thm. 7) No Poly Ker. (Thm. 8)	Para NP-h (Thm. 6)

schedule satisfies the capacity constraints posed on the machine types in the all steps. The idea behind keeping the number of waiting time removals small is that maintaining many waiting times increases the robustness against future disruptions.

Related Work. To the best of our knowledge, our introduced problem WTR has not yet been studied.

One framework that is conceptually related to the idea of repairing given schedules is the approach of *parameterized local search*: The idea is to apply a limited number of changes on a given schedule to improve a target function value. Balzereit et al. [3] study parameterized local search for a single machine scheduling problem. While their approach mostly serves as a subroutine in a hill-climbing setting, it can also be used to repair schedules.

Another framework related to repairing schedules is *reoptimization* [4, 17]: The idea of reoptimization is to compute approximate solutions of perturbed instances from given optimal solutions of an original instance. There are constant factor approximations for reoptimization settings of multi-machine makespan minimization under the constraint that some jobs are not allowed to be scheduled parallel [17]. Boria and Della Croce [4] provide constant factor approximizations for multiple min-sum scheduling problems with release dates.

Our Results. We initiate the theoretical study of WTR. As WTR is NP-hard even if the number of allowed waiting time removals is unbounded (Theorem 3), the problem is inapproximable, motivating our parameterized complexity analysis. Considering the most natural parameterizations (by the number n of jobs, the number k of deletions, the time horizon T, and the alphabet size $|\Sigma|$ and combinations thereof), we derive a complete picture of the parameterized complexity of WTR. An overview of the results is shown in Table 1.

Notably, while most of our hardness results hold already for alphabets of constant size, the paraNP-hardness of WTR parameterized by T (Theorem 6) is contrasted with the fixed-parameter tractability with respect to $T + |\Sigma|$ (Theorem 7). A further notable result is that parameterizing with the number n of jobs is not sufficient to ensure fixed-parameter tractability (Theorem 3), but only results in an XP-algorithm (Theorem 4).

2 Preliminaries

We use standard notations from parameterized complexity [5].

For integers $a \leq b$, we let $[a, b] := \{i \in \mathbb{N} \mid a \leq i \leq b\}$. We let $\circ$ denote the concatenation of strings, and we let $|w|$ denote the length of a string w. Furthermore, for $i \in \mathbb{N}$, we let $w[i]$ be the ith symbol of w or the empty word if $i \notin [1, |w|]$.

Let Σ be a finite alphabet, let $x \notin \Sigma$, and let $w \in (\Sigma \cup \{x\})^*$ be a string with $m \in \mathbb{N}$ occurrences of the symbol x. For $R \subseteq [1, m]$, we let $w - R$ denote the string after removing the ith occurrences of x for each $i \in R$ from w. As an example, consider $axbbxcx - \{1, 3\} = abbxc$, where we remove the first and the third occurrence of x while keeping the second occurrence of x. Throughout this work, we call such set R an *x-removal set*.

Problem Definition. Recall that we aim to study a problem where the goal is to adjust a given plan by removing waiting times. In the following, we formalize this idea.

Let Σ be a finite alphabet, let $x \notin \Sigma$, and let $T \in \mathbb{N}$ be a number of time steps. A *T-step capacity bound for* Σ is a mapping $c : \Sigma \times [1, T] \to \mathbb{N}_0$. Intuitively, $c(\sigma, t)$ corresponds to the capacity of symbol σ at time step t. A *job* (w, s) is a tuple consisting of a string $w \in (\Sigma \cup \{x\})^*$ and a starting time $s \in \mathbb{N}$ that satisfies $s + |w| - 1 \leq T$. Intuitively, the inequality guarantees that every job is completed by time step T. Given a set of jobs $\mathcal{J} := \{(w_i, s_i) \mid i \in [1, n]\}$, a symbol $\sigma \in \Sigma$ and a time step $t \in [1, T]$, we define the *load of symbol σ at time step t* as

$$\lambda_\sigma^t := |\{(w, s) \in J \mid w[t - s + 1] = \sigma\}|.$$

A set $\mathcal{J}$ of jobs *satisfies the capacity constraints* c if $\lambda_\sigma^t \leq c(\sigma, t)$ for every combination of $\sigma \in \Sigma$ and $t \in [1, T]$. For a job J and some $t \in [s, s + |J| - 1]$, we say that the $(t - s + 1)$th letter of J *is processed* at time t.

Given a set of jobs $\mathcal{J} := \{(w_i, s_i) \mid i \in [1, n]\}$, we let $\#x_i$ denote the number of occurrences of the symbol x in the string w_i for $i \in [1, n]$. The problem WTR is defined as follows.

Waiting Time Removal (WTR)
Input: A number of time steps T, a set $\mathcal{J} := \{(w_i, s_i) \mid i \in [1, n]\}$ of jobs, a T-step capacity bound c, and an integer k.
Question: Does there exist a family of sets $R_1, \ldots, R_n$ with $R_i \subseteq [1, \#x_i]$ for each $i \in [1, n]$ and $\sum_{i=1}^{n} |R_i| \leq k$ such that the job set $\mathcal{J}' := \{(w_i - R_i, s_i) \mid i \in [1, n]\}$ satisfies the capacity constraints c?

For notational convenience, we define for a job $J = (w, s)$ its length $|J| := |w|$, its starting time $s(J) := s$, and $J - R := (w - R, s)$ for $R \subseteq [1, \#x_i]$.

Note that $T \cdot n$ essentially bounds the size of an instance $(T, \mathcal{J}, c, t)$ of WTR: Each w_i has length at most T and thus, $|\mathcal{J}|$ can be encoded by encoding at most $T \cdot n$ symbols. Moreover, the load of a symbol can never exceed n and thus, c can be encoded using $\mathcal{O}(T^2 \cdot n \log(n))$ bits. Therefore, we can observe the following.

▶ **Observation 1.** *WTR admits a polynomial kernel for $T + n$.*

Furthermore, note that WTR can be solved by the following simple brut-force algorithm: Enumerate all size-k subsets of occurrences of the symbol x in the input instance, and check whether removing precisely these occurrences provides a modified set $\mathcal{J}'$ that satisfies the capacity constraints c. This simple brute-force strategy implies the following.

▶ **Observation 2.** *WTR can be solved in $|I|^{k+\mathcal{O}(1)}$ time, where $|I|$ denotes the input size.*

Graph Theory. We use standard notation from graph theory, see e.g. [12]. More precisely, all appearing graphs are undirected unless stated otherwise. A graph $G = (V, E)$ is *k-partite* if its vertex set can be partitioned into k independent sets, i.e., $V = V_1 \cup \ldots \cup V_k$ with $V_i \cap V_j = \emptyset$ for each $i \neq j$ and for each set V_i, there is no edge whose endpoints are both contained in V_i. For a graph $G = (V, E)$ and $V_1, V_2 \subseteq V$, we denote by $E[V_1, V_2]$ the edges with one endpoint in V_1 and the other endpoint in V_2. For a vertex $v \in V$, we denote by $\delta(v)$ the set of edges incident to v.

3 Parameterization by the Number of Jobs

We first study the parameterization by the number n of jobs. While the total number of jobs naturally appears to be a large parameter in real-world applications, we start by showing that – somewhat surprisingly – WTR is unlikely to be fixed-parameter tractable for the parameter n.

▶ **Theorem 3.** *For an instance of WTR, deciding the existence of a deletion family (of arbitrary size) is NP-hard and W[1]-hard parameterized by the number of jobs, even if $|\Sigma| = 2$.*

Proof. We reduce from Multicolored Independent Set which is NP-hard [11] and W[1]-hard parameterized by solution size ℓ [15]. In Multicolored Independent Set, one is given an ℓ-partite graph $G = (V_1 \cup \ldots \cup V_\ell, E)$ with $|V_1| = |V_2| = \ldots = |V_\ell|$, and the task is to find a multicolored independent set, i.e., a set of ℓ pairwise non-adjacent vertices containing exactly one vertex from each V_i. Let $G = (V_1 \cup \ldots \cup V_\ell, E)$ be an instance of Multicolored Independent Set. We fix an arbitrary ordering $e_1, \ldots, e_m$ of the edges of G as well as of the vertices of each color class $V_i = \{v_i^1, \ldots, v_i^{|V_1|}\}$. For each color class V_i and each edge $e_p \in E$, we define the $|V_1|$-letter substring w_i^p to be

$$w_i^p := \begin{cases} 0^{j-1}10^{|V_1|-j} & \text{if } e_p \in \delta(v_i^j) \text{ for some } j \in [1, |V_1|] \\ 0^{|V_1|} & \text{otherwise.} \end{cases}$$

Using these substrings, we create, for each $i \in [1, \ell]$, one job $J_i = (w_i, 1)$ where

$$w_i := x^{|V_1|-1} \circ \bigcirc_{p=1}^{m} w_i^p \,.$$

We define the $(m \cdot |V_1| + |V_1| - 1)$-capacity bound c: For every $p \in [1, m]$, letter 1 has capcity one at time $p \cdot |V_1|$. All other capacities are infinite. See Figure 2 for an example. The reduction clearly runs in polynomial time and the number of jobs equals ℓ.

We continue by showing correctness. First note that for any set $R_i \subseteq [1, |V_1| - 1]$, job $J_i - R_i$ contributes to the load of letter 1 at time $p \cdot |V_1|$ if and only if $v_i^{|R_i|+1}$ is adjacent to e_p. Given a multicolored independent set $I = \{v_1^{j_1}, \ldots, v_\ell^{j_\ell}\}$, we construct a deletion set $\mathcal{R} = \{R_1, \ldots, R_n\}$ as follows: For each job J_i, we let $R_i = [1, j_i - 1]$. To see that this $\{(w_i - R_i, 1) \mid i \in [1, n]\}$ satisfies the capacity constraints, note that only letter 1 has a finite capacity (at times $t = p \cdot |V_1|$ for $p \in [1, m]$), so it suffices to show that the load of letter 1 does not exceed its capacity. Assume towards a contradiction that there is some time t where the load of letter 1 exceeds its capacitiy, i.e., $t = p \cdot |V_1|$ for some $p \in [1, m]$ and there are two jobs $J_i - R_i$ and $J_{i'} - R_{i'}$ both contributing to the load of 1 at time t. This implies that e_p is incident to both $v_i^{j_i}$ and $v_{i'}^{j_{i'}}$, a contradiction to I being an independent set.

We conclude with the reverse direction, so let $\mathcal{R} = \{R_1, \ldots, R_n\}$ be a deletion family satisfying the capacity constraints. Let $j_i := |R_i| \in [1, |V_1| - 1]$. We claim that $I = \{v_1^{j_1}, \ldots, v_\ell^{j_\ell}\}$ is an independent set. Assume towards a contradiction that $e_p = \{v_i^{j_i}, v_{i'}^{j_{i'}}\}$ for some $p \in [m]$ and $i, i' \in [1, \ell]$. Then at time $p \cdot |V_1|$, both $J_i - R_i$ and $J_{i'} - R_{i'}$ contribute to the load of letter 1, a contradiction to the feasibility of the schedule. ◀

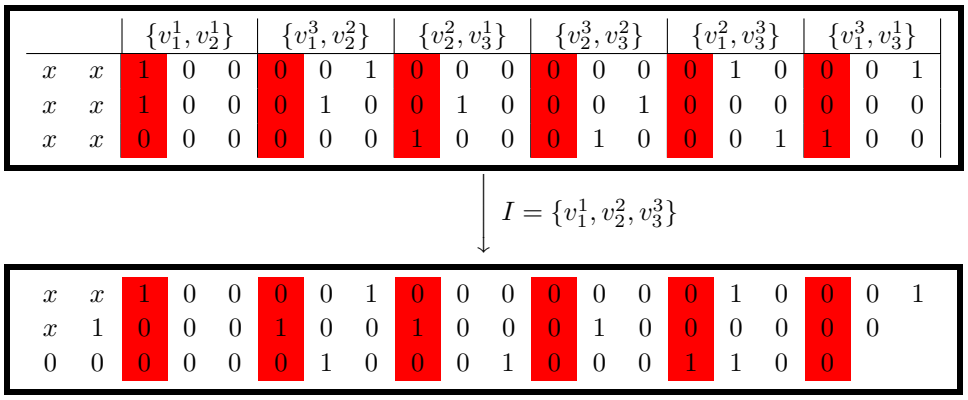


Figure 2 The upper part shows an example for the reduction for $\ell = 3$ and the graph containing edges $\{v_1^1, v_2^1\}$, $\{v_1^3, v_2^2\}$, $\{v_2^2, v_3^1\}$, $\{v_2^3, v_3^2\}$, $\{v_1^2, v_3^3\}$, and $\{v_1^3, v_3^1\}$. Red columns correspond to time steps where the capacity of letter 1 equals one while all other time steps have unboundend capacity. The vertical lines separate the different "blocks" of $|V_1|$ letters corresponding to an edge. The lower part depicts a solution corresponding to the independent set $\{v_1^1, v_2^2, v_3^3\}$.

While WTR is presumably not fixed-parameter tractable for n, we show that – on the positive side – WTR can be solved in polynomial time for constant values of n by providing the following XP algorithm.

▶ **Theorem 4.** *WTR can be solved in $O(n \cdot T^{2n+1})$ time.*

Proof. We give a dynamic program solving the problem. The dynamic programming table τ has entries of the form $\tau[(t, p_1, \ldots, p_n)]$ for every suitable tuple $(t, p_1, \ldots, p_n)$, where we call $(t, p_1, \ldots, p_n)$ with $t \in [0, T+1]$ and $p_i \in [0, |J_i|+1]$ for each $i \in [1, n]$ *suitable* if, for every $i \in [n]$, it holds that $p_i = 0$ if and only if $t < s(J_i)$. Each entry stores the minimum size of a deletion set such that the load of any symbol for any time $t' \leq t$ does not exceed its capacity and at time t (where each letter has capacity 0 for $t' \in \{0, T+1\}$), the p_ith letter of J_i is processed. Herein, $p_i = 0$ encodes that no letter of J_i is processed until time t and $p_i = |J_i| + 1$ encodes that all letters from J_i have already been processed until time t. The table is computed as follows: We initialize the table by setting

$$\tau[(0, p_1, \ldots, p_n)] = \begin{cases} 0 & \text{if } p_1 = \ldots = p_n = 0 \\ \infty & \text{otherwise.} \end{cases}$$

For the update step, we introduce some additional notation. We say that a suitable tuple $(t-1, p_1', \ldots, p_n')$ *is compatible with* a suitable tuple $(t, p_1, \ldots, p_n)$ if, for every $i \in [n]$,

- $p_i' \leq p_i$,
- if $p_i \notin \{0, |J_i|+1\}$, then $p_i' < p_i$ and $w_i[p_i'+1] = w_i[p_i'+2] = \ldots = w_i[p_i-1] = x$, and
- for each $\sigma \in \Sigma$, we have $|\{i \mid s_i[p_i] = \sigma\}| \leq \lambda_\sigma^t$.

Intuitively, this means that if the p_i'th letter of job J_i is processed at time $t-1$, then the p_ith letter can be processed at time t (after deleting all x in between; the second item ensures that only x are in between the p_i'th and p_ith letter). We update the table as follows:

$$\tau[(t, p_1, \ldots, p_n)] = \min_{(t-1, p_1', \ldots, p_n') \text{ compatible with } (t, p_1, \ldots, p_n)} \tau[(t-1, p_1', \ldots, p_n')] + \sum_{i : p_i \neq p_i'} (p_i - p_i' - 1)\,.$$

An optimal solution can be read from $\tau[(T+1, |J_1|+1, \ldots, |J_n|+1)]$ using traceback. More formally, we have a yes-instance of WTR if and only if $\tau[(T+1, |J_1|+1, \ldots, |J_n|+1)] \leq k$.

The dynamic programming table has $O(T \cdot \Pi_{i=1}^{n}(|J_i|+2)) = O(T^{n+1})$ entries, each of which can be computed in $O(n \cdot T^n)$ time. Thus, the dynamic program runs in $O(n \cdot T^{2n+1})$ time.

It remains to show correctness. First, we show by induction on t that if $\tau[(t, p_1, \ldots, p_n)] = q < \infty$, then there is a deletion family of size q such that no load exceeds its capacity at any time $t' \leq t$ and at time t, the p_ith letter of J_i is scheduled. This clearly holds for $t = 0$. For $t \geq 1$, let $a := (t-1, p'_1, \ldots, p'_n)$ such that $\tau[(t, p_1, \ldots, p_n)] = \tau[a] + \sum_{i:p_i \neq p'_i}(p_i - p'_i - 1)$. This entry $\tau[a]$ corresponds to a deletion family $\mathcal{R}'$. By induction, for jobs $\{J_i - R'_i\}$, no load is exceeded for any time $t' < t$. By the second condition, the $p'_i + 1, \ldots, p_i - 1$th letters of J_i are all x, say the r_ith to r'_ith occurrences of x. Setting $\mathcal{R} = \{R'_i \cup [r_i, r'_i] \mid i \in [n]\}$ for each $i \in [n]$ results in a deletion family of size $\tau[a] + \sum_{i:p_i \neq p'_i}(p_i - p'_i - 1) = q$. By the third condition, no load is exceeded at time t. By induction, no load is exceeded at any time $t' \leq t-1$ for $\mathcal{R}'$; consequently, also for $\mathcal{R}$ no load is exceeded at any time $t' \leq t-1$ for $\mathcal{R}$. Thus, $\mathcal{R}$ is a deletion family of size q with the desired properties.

Next, we show that if there is a deletion family $\mathcal{R}$ of size q such that at time t, for each $i \in [1, n]$ the p_ith letter of J_i is scheduled, and no load is exceeded for any $t' \leq t$, then $\tau[(t, p_1, \ldots, p_n)] \leq q$. We again do so by induction on t. For $t = 0$, the statement is obvious, so consider $t > 0$. The deletion family $\mathcal{R}$ is also a deletion family where at time $t-1$, the p'_ith letter of J_i is scheduled for some $p'_1, \ldots, p'_n$. By the second condition, the $p'_i + 1, \ldots, p_i - 1$th letters of J_i are all x, say the r_ith to r'_ith occurrences of x. Setting $R'_i := R_i \setminus [r_i, r'_i]$ for $i \in [1, n]$ therefore results in a deletion set $\mathcal{R}' = \{R'_1, \ldots, R'_n\}$ of size $q - \sum_{i:p_i \neq p'_i}(p_i - p'_i - 1)$. By induction, we have $\sum_{i=1}^{n} |R'_i| \geq \tau[(t-1, p'_1, \ldots, p'_n)]$. Thus, we have

$$q = \sum_{i=1}^{n} |R'_i| + \sum_{i:p_i \neq p'_i} (p_i - p'_i - 1) \geq \tau[(t-1, p'_1, \ldots, p'_n)] + \sum_{i:p_i \neq p'_i} (p_i - p'_i - 1)$$
$$\geq \tau[(t, p_1, \ldots, p_n)] .$$

The correctness follows. ◀

Adding a simple pruning rule to the dynamic program from Theorem 4 yields fixed-parameter tractability by the combined parameter $n + k$:

▶ **Corollary 5.** *WTR parameterized by $n + k$ can be solved in $O(T \cdot n^{2k+1})$ time.*

Proof. For every job $J_i = (w_i, s_i)$, the deletion set R_i has size at most k. Thus, at each time t, at most k different letters from w_i can be scheduled, implying that at most k different values for p_i need to be considered. Consequently, the number of DP entries to consider is $O(T \cdot n^k)$, and each of these entries can be computed in $O(n \cdot n^k)$ time. Therefore, WTR can be solved in $O(T \cdot n^{2k+1})$ time. ◀

4 Parameterization by the Number of Time Steps

We next study parameterization by the number T of time steps. We start by showing that WTR cannot be solved in polynomial time for constant values of T unless P = NP. Motivated by this strong hardness result, we then study parameterizations by the combined parameters of $T + |\Sigma|$ and $T + k$.

▶ **Theorem 6.** *WTR is NP-hard even if T is constant.*

Proof. We provide a reduction from Independent Set. In Independent Set, one is given a graph $G = (V, E)$ together with an integer ℓ. The question is whether there exists a set $I \subseteq V$ of pairwise non-adjacend vertices with $|I| \geq \ell$. Independent Set is well-known to be NP-hard even if G has maximum degree 3 [7].

Let $(G = (V, E), \ell)$ be an instance of Independent Set, where G has maximum degree 3. We describe how to construnct an equivalent instance of WTR in polynomial time. To this end, we assume that G has a proper edge coloring with four colors: Due to Vizing's Theorem [23], we can partition the edge set E into four sets E_1, E_2, E_3, and E_4 such that for each $i \in [1, 4]$, no two edges in E_i share one endpoint. Since the proof of Vizing's Theorem provides an algorithm computing the partition in polynomial time, we may use the sets $E_1, \ldots, E_4$ for our construction.

To construct an instance $(T, \mathcal{J}, c, k)$ of WTR, we first set $k := \ell$ and we set $T := 10$. Furthermore, we define the alphabet $\Sigma := \{0\} \cup E$. Next, given some $v \in V$ and some $i \in [1, 4]$, we let $w_v^i := 00$ if v has no incident edge in E_i, and we let $w_v^i := 0e$ otherwise, where e is the unique incident edge in E_i. We then define $w_v := x \circ w_v^1 \circ w_v^2 \circ w_v^3 \circ w_v^4 \circ 0$ and set $\mathcal{J} := \{(w_v, 1) \mid v \in V\}$. Note that for every $v \in V$, we have $|w_v| = 10$.

We complete the construction by defining the capacity c. We set $c(0, 10) := |V| - \ell$. Furthermore, for every $i \in \{2, 4, 6, 8\}$ and $e \in E$ we define $c(e, i) := 1$. All remaining capacities are set to ∞.

Before we prove the correctness, we provide some intuition. Since $c(0, 10) = |V| - \ell$ and every w_v ends with the symbol 0 at time step 10, we need to remove the unique occurrence of x in ℓ strings, which then corresponds to the choice of vertices in a solution of Independent Set. The capacities $c(e, i) := 1$ for $i \in \{2, 4, 6, 8\}$ then guarantee that no pair of chosen vertices are incident with the same edge.

We conclude the proof by showing that (G, ℓ) is a yes-instance of Independent Set if and only if $(T, \mathcal{J}, c, k)$ is a yes-instance of WTR.

Let (G, ℓ) be a yes-instance of Independent Set. Then, there is a set $I \subseteq V$ containing ℓ vertices that are pairwise non-adjacent. We construct a solution of the WTR instance as follows: for each $v \in I$, we remove the unique occurrence of x in w_v and let $\tilde{w}_v$ denote the resulting string. Note that $\tilde{w}_v[t] = w_v[t + 1]$ for $t \in [1, 9]$. Note that we removed $|I| = \ell = k$ occurrences of x. We show that the modified job set satisfies the capacity constraints c: First, observe that $\lambda_0^{10} = |V| - |I| = V - \ell = c(0, 10)$. Second, let $i \in \{2, 4, 6, 8\}$ and assume towards a contradiction that $c(e, i) \geq 2$ for some symbol $e \in \Sigma$. Then, there exist $\tilde{w}_v$ and $\tilde{w}_u$ with $\tilde{w}_v[i] = \tilde{w}_u[i] = e$. Then, we have $w_v[i+1] = w_u[i+1] = e$ and thus, by the construction of $\mathcal{J}$, the vertices v and u are both incident with the same edge e in G. This contradicts the fact that I consists of pairwise non-adjacent vertices. Consequently, we have $c(e, i) \leq 1$ for every $i \in \{2, 4, 6, 8\}$ and $e \in E$. Since all other capacities are ∞, we conclude that the modified job set satisfies the capacity constraints c and thus, $(T, \mathcal{J}, c, k)$ is a yes-instance of WTR.

Conversely, let $(T, \mathcal{J}, c, k)$ be a yes-instance of WTR. Then, there is a subset of jobs $\mathcal{J}' \subseteq \mathcal{J}$ with $|\mathcal{J}'| \leq k = \ell$ such that removing the unique occurrence of x in the corresponding strings results in a plan that satisfies the capacity constraints c. Observe that $|\mathcal{J}'| = \ell$ since otherwise $\lambda_0^{10} = |V| - |\mathcal{J}'| > |V| - \ell = c(0, 10)$. Let $w_{v_1}, \ldots, w_{v_\ell}$ be the strings of the jobs in $\mathcal{J}'$. Then, $I := \{v_1, \ldots, v_\ell\}$ is a set of vertices of G with $|I| = \ell$. It reamins to show that the vertices in I are pairwise non-adjacent. Assume towards a contradiction that two vertices v_1 and v_2 of I are connected by an edge e. Recall that the edge set E is partitioned into the sets $E_1, \ldots, E_4$. Without loss of generality we assume $e \in E_1$. Then, by the construction of the jobs we have $w_{v_1}[3] = w_{v_2}[3] = e$. Removing the single occurrence of x in w_{v_1} and w_{v_2}

provides strings $\tilde{w}_{v_1}$ and $\tilde{w}_{v_2}$ with $\tilde{w}_{v_1}[2] = \tilde{w}_{v_2}[2] = e$. Since $c(e, 2) = 1$, this contradicts the fact that the modified job set satisfies the capacity constraints c. Consequently, all vertices in I are pairwise non-adjacent and thus, (G, ℓ) is a yes-instance of Independent Set. ◀

Note that the alphabet size of the instance constructed in the proof of Theorem 6 is roughly the size of the input graph of the Independent Set instance. Recall that in our application the symbols in Σ correspond to distinct machine types on which the jobs need to be processed. In a real-world production setting it is reasonable that there are only few distinct machines types. Motivated by this observation, we study the combined parameter $T + |\Sigma|$ and show that WTR is FPT for this parameterization.

▶ **Theorem 7.** *WTR parameterized by $T + |\Sigma|$ is FPT.*

Proof. Let $(T, \mathcal{J}, c, k)$ be an instance of WTR. We show fixed-parameter tractability by reformulating WTR as an integer linear program (ILP) where the number of variables is bounded by a function only depending on T and $|\Sigma|$. It is well-known that ILP parameterized by the number of variables is in FPT [9]. To this end, we introduce the notion of job types. Recall that a job $(w, s) \in \mathcal{J}$ consists of a string $w \in (\Sigma \cup \{x\})^*$ and a starting time $s \in [1, T]$. We say that two jobs *have the same type* $\tau = (w_\tau, s_\tau)$ if they have the same string w_τ and the same starting time s_τ. We denote the number of jobs of type τ by n_τ. For each job type τ, we consider possible x-removal sets $R \subseteq [1, \#w_\tau]$. We provide the ILP formulation by specifying the variables, the constraints, and the target function.

Variables. For each combination of a job type τ and a possible removal set $R \subseteq [1, \#w_\tau]$, we introduce an integer variable $Y_{(\tau,R)}$. The intuitive idea is that the value of $Y_{(\tau,R)}$ equals the number of jobs with type τ whose internal x-deletions correspond to the (possibly empty) removal set R. More precisely, a solution set $\mathcal{J}'$ contains exactly $Y_{(\tau,R)}$ jobs of the form $(w_\tau - R, s_\tau)$.

Constraints. For each type τ, we add the following constraint, ensuring that all jobs of type τ are scheduled:

$$\sum_{R \subseteq [1, \#w_\tau]} Y_{(\tau,R)} = n_\tau \,.$$

For each combination of $\sigma \in \Sigma$ and $t \in [1, T]$, we introduce the following constraint:

$$\sum_{\text{Type } \tau = (w_\tau, s_\tau)} \; \sum_{\substack{R \subseteq [1, \#w_\tau] \\ (w_\tau - R)[t - s_\tau + 1] = \sigma}} Y_{(\tau,R)} \leq c(\sigma, t) \,.$$

The left-hand side corresponds to the load of symbol σ at time step t. Thus, these constraints guarantee that the solution satisfies the capacity constraints c: Intuitively, each string w_τ has been modified by a (possibly empty) x-removal set R. For a given type τ, the inner sum counts all modified strings of type τ that have symbol σ at time step t. Since we sum up these values over all possible types, the left-hand side corresponds to the load of symbol σ at time step t.

Target Function. The target function has the form

$$\sum_{\text{Type } \tau} \; \sum_{R \subseteq [1, \#w_\tau]} |R| \cdot Y_{(\tau,R)}.$$

Recall that intuitively each value of $Y_{(\tau,R)}$ equals the number of jobs with type τ whose internal x-deletions correspond to the (possibly empty) removal set R. Thus, for each combination of a type τ and a removal set R, we delete exactly $Y_{(\tau,R)} \cdot |R|$ symbols. Since we sum up over all τ and R, the target function models the total number of x-deletions.

Number of Variables. Note that the number of strings in $(\Sigma \cup \{x\})^*$ in an instance with maximum time step T is bounded by $(|\Sigma|+1)^T$, and that we have at most T starting positions. Consequently, there are at most $T \cdot (|\Sigma|+1)^T$ possible job types $\tau = (w_\tau, s_\tau)$. Furthermore, since $|w_\tau| \leq T$ for all τ, there are at most 2^T possible x-removal sets $R \subseteq [1, \#w_\tau]$. Thus, the number of variables $Y_{(\tau,R)}$ is bounded by $T \cdot (|\Sigma|+1)^T \cdot 2^T$. By a classical result by Lenstra [9], an ILP can be solved FPT-time with respect to the number of variables. This implies that the above ILP can be solved in FPT-time with respect to $T + |\Sigma|$. ◀

We complement the FPT result for $T + |\Sigma|$ by showing that WTR is unlikely to admit a problem kernel of polynomial size for this parameterization.

▶ **Theorem 8.** *WTR parameterized by $T+|\Sigma|$ does not admit a polynomial kernel unless* NP $\subseteq$ coNP/poly.

Proof. We prove the statement by providing a polynomial parameter transformation from Set Cover. In Set Cover, one is given a universe of elements U, a familiy $\mathcal{F} \subseteq 2^U$ of subsets of U, and an integer ℓ. The question is whether there exists a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ such that $|\mathcal{F}'| \leq \ell$ and $\bigcup_{F \in \mathcal{F}'} F = U$. Set Cover parameterized by $|U|$ does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly [6].

Let $(U, \mathcal{F}, \ell)$ be an instance of Set Cover and fix an arbitrary ordering $U := \{u_1, \ldots, u_{|U|}\}$ of the elements in the universe. We describe how to construct an equivalent instance $(T, \mathcal{J}, c, k)$ of WTR for the alphabet $\Sigma = \{0, 1\}$. We set $k := \ell$ and we set $T := 2 \cdot |U| + 1$. Note that T is polynomial in $|U|$. Next, we define the job set $\mathcal{J}$. Given some $F \in \mathcal{F}$ and some $u \in U$, we let $w_u^F := 01$ if $u \in F$ and $w_u^F := 00$ if $u \notin F$. We define the job set $\mathcal{J} := \{(w^F, 1) \mid F \in \mathcal{F}\}$, where

$$w_F := x \circ \bigcirc_{i=1}^{|U|} w_{u_i}^F.$$

Note that $w_F[2i+1] = 1$ if and only if $u_i \in F$ and we have $w_F[2i] = 0$ for every $i \in [1, |U|]$.

We finish the construction by defining the $(2 \cdot |U| + 1)$-capacity bound c: For every $i \in [1, |U|]$, we set $c(1, 2i+1) := |\{F \in \mathcal{F} \mid u \in F\}| - 1$. All remaining $c(\sigma, t)$ are set to ∞.

We next show that $(U, \mathcal{F}, \ell)$ is a yes-instance of Set Cover if and only if $(T, \mathcal{J}, c, \ell)$ is a yes instance of WTR.

Let $(U, \mathcal{F}, \ell)$ be a yes-instance of Set Cover. Then, there exists a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ with $|\mathcal{F}'| \leq \ell$ such that for each $i \in [1, |U|]$, there is one $F \in \mathcal{F}'$ with $u_i \in F$. We construct a solution of the WTR instance as follows: for each $F \in \mathcal{F}'$, we remove the unique occurrence of x in w^F, and let $\tilde{w}^F$ denote the resulting string. Note that $\tilde{w}^F[t] = w^F[t+1]$ for every $t \in [1, T-1]$. We show that the modified job set satisfies the capacity constraints c. Since all other capacities are ∞, it suffices to show that for each $i \in [1, |U|]$ the load of symbol 1 at time step $2i+1$ is not larger than $c(1, 2i+1)$. Since $\tilde{w}^F[2i+1] = w^F[2i+2] = 0$ for all $F \in \mathcal{F}'$, we have $\lambda_1^{2i+1} = |\{F \in \mathcal{F} \setminus \mathcal{F}' \mid u_i \in F\}|$. Since each u_i is contained in at least one $F \in \mathcal{F}'$, we conclude that $\lambda_1^{2i+1} \leq |\{F \in \mathcal{F} \mid u_i \in F\}| - 1 = c(1, 2i+1)$. Therefore, $(T, \mathcal{J}, c, \ell)$ is a yes instance of WTR.

Conversely, let $(T, \mathcal{J}, c, k)$ be a yes-instance of WTR. Then, there is a subset $\mathcal{J}' \subseteq \mathcal{J}$ of jobs with $|\mathcal{J}'| \leq \ell$ such that removing the unique occurrence of x in the corresponding strings results in a plan that satisfies the capacity constraints c. Let $w^{F_1}, \ldots, w^{F_{|\mathcal{J}'|}}$ be the

strings of the jobs in $\mathcal{J}'$. Then, $\mathcal{F}' := \{F_1, \ldots, F_{|\mathcal{J}'|}\}$ is a subfamily of $\mathcal{F}$ of size at most ℓ. Assume towards a contradiction that there is some $u_i \in U$ with $u_i \notin \bigcup_{F \in \mathcal{F}'} F$. Then, for each $F \in \mathcal{F}$ with $u_i \in F$, the occurrence of x in w^F has not been removed. Consequently, the load of symbol 1 at time step $2i+1$ is $|\{F \in \mathcal{F} \mid u_i \in F\}| > c(1, 2i+1)$. This contradicts the fact that removing the x in the strings of $\mathcal{J}'$ results in a job set that satisfies the capacity constraints c. Therefore, $(U, \mathcal{F}, \ell)$ is a yes-instance of Set Cover. ◀

The following two observations can be made by a taking closer look at the instance constructed in the proof of Theorem 8: First, the parameter k equals the solution size of the Set Cover instance. Since Set Cover is known to be W[2]-hard when parameterized by solution size [5], this implies the following.

▶ **Corollary 9.** *WTR parameterized by k is W[2]-hard even if $|\Sigma| = 2$.*

Second, the parameter $n + k$ is upper-bounded by $2 \cdot |\mathcal{F}|$, where $\mathcal{F}$ is the set familiy in the Set Cover instance. Since Set Cover does not admit a polynomial kernel for $|\mathcal{F}|$ [6], this implies the following.

▶ **Corollary 10.** *WTR does not admit a polynomial kernel for $n+k$ unless* NP $\subseteq$ coNP/poly *even if $|\Sigma| = 2$.*

Next, consider the parameterization by $T + k$. Recall that WTR is XP for k by Observation 2 and thus, WTR is XP for $T + k$. We complement this result by showing W[1]-hardness.

▶ **Theorem 11.** *WTR parameterized by $T + k$ is W[1]-hard.*

Proof. We reduce from Multicolored Clique parameterized by solution size ℓ. In Multicolored Clique, one is given an ℓ-partite graph $G = (V_1 \cup \ldots \cup V_\ell, E)$ with $|V_1| = |V_2| = \ldots = |V_\ell|$, and the task is to find a multicolored clique, i.e., a set of ℓ pairwise adjacent vertices containing exactly one vertex from each V_i. Multicolored Clique is W[1]-hard parameterized by ℓ, even if there is some $r \in \mathbb{N}$ so that each vertex has exactly r neighbors in every other color class [5]. Let $G = (V_1 \cup \ldots \cup V_\ell, E)$ be an instance of Multicolored Clique where each vertex v has exactly r neighbors in every other color class. We let $f : [1, \binom{\ell}{2}] \to \binom{[1,\ell]}{2}$ be an arbitrary bijection to fix an ordering of the set $\binom{[1,\ell]}{2} := \{X \subseteq [1, \ell] \mid |X| = 2\}$ containing all two-element subsets of $[1, \ell]$. We use the alphabet $\Sigma = V \cup \{0, 1\} \cup \{z_i \mid i \in [\ell]\}$. We add the following jobs:

- for each $v \in V_i$, we add one job $J_v = (w_v, 1)$ where w_v starts with xz_i. Afterwards, for each $q \in [1, \binom{\ell}{2}]$, we append the following four letters:
  - if $i \notin f(q)$, then we append 0000,
  - if $f(q) = \{i, i'\}$ for some $i' > i$, then we append $0v00$, and
  - if $f(q) = \{i', i\}$ for some $i' < i$, then we append $000v$.
- for each edge $e = \{u, v\} \in E$ with $u \in V_i$ and $v \in V_{i'}$ for some $i < i'$, we add one job $J_e = (xu0v, 4 \cdot (f(\{i, i'\}) - 1) + 2)$.

The capacities are as follows:

- for each $i \in [1, \ell]$, letter z_i has capacity 1 at time 1 and capacity $|V_1| - 1$ at time 2,
- for each $v \in V$, letter v has capacity r at every time step, and
- all other capacities are set to be ∞.

Finally, we set $k := \ell + \binom{\ell}{2}$. Note that $T = 4 \cdot \binom{\ell}{2} + 2$. As the reduction can clearly be computed in polynomial time, it remains to show correctness.

Before we show the correctness, we provide some intuition. The capacities of z_i at time 1 ensures that at most one deletion occurs in jobs $\{J_v \mid v \in V_i\}$, while the capacities of z_i at time 2 ensure that at least one deletion occurs in $\{J_v \mid v \in V_i\}$. Therefore, exactly one deletion occurs in $\{J_v \mid v \in V_i\}$ for every $i \in [1, \ell]$. These ℓ vertices with a deletion correspond to a multicolored clique: Since there exists some unique $v_i \in V_i$ such that the x in J_{v_i} is deleted, this implies that all occurrences of v_i in J_{v_i} move one time step earlier. Thereby, for each $q \in [1, \binom{\ell}{2}]$ with $i \in f(q)$, letter v_i from J_{v_i} "collides" with letter v_i in J_e the r edges e incident to v_i in $E[V_i, V_{i'}]$ where $i' = f(q) \setminus \{i\}$. Because the capacity of v_i is r at each time step, this implies that a deletion has to occur in J_e for some edge $e \in E[V_i, V_{i'}]$ incident to v_i. As the budget allows for $k - \ell = \binom{\ell}{2}$further deletions, this implies that there is exactly one edge in $E[V_i, V_{i'}]$ with a deletion for $\{i, i'\} \in \binom{[1,\ell]}{2}$. Since this edge needs to be incident to both v_i and $v_{i'}$, it follows that $\{v_i \mid i \in [1, \ell]\}$ is a multicolored clique.

We first show that if G is a yes-instance to Multicolored Clique, then $(T, \mathcal{J}, c, k)$ is yes-instance of WTR. Given a clique $\{v_1, \ldots, v_\ell\}$ with $v_i \in V_i$, we construct a solution to the WTR instance as follows: We remove the unique occurrence of x in job J_{v_i} for each $i \in [1, \ell]$ and in job $J_{\{v_{i_1}, v_{i_2}\}}$ for $i_1 < i_2 \in [1, \ell]$. We call the resulting strings $\tilde{w}_v$ for $v \in V$ and $\tilde{w}_e$ for $e \in E$. As we removed k occcurences of x in total, it remains to show that the capacity constraints are satisfied. For letter z_i, we have a load of 1 at time 1 (from job J_{v_i}) and a load of $|V_1| - 1$ at time to (from jobs J_v for $V \in V_i \setminus \{v_i\}$). Thus, the capacities of z_i are satisfied. For $v \in V \setminus \{v_1, \ldots, v_\ell\}$, the maximum load at any time is $\deg(v) = r$, so its capacity is satisfied. We continue with letter v_i for $i \in [1, \ell]$. At time $4 \cdot (q-1) + 3$, the load of v_i is either 0 (if $i \notin f(q)$ or $f(q) = \{i, i'\}$ for some $i' < i$) or r (if $f(q) = \{i, i'\}$ for some $i' > i$; in this case, job J_{v_i} and $r - 1$ edge jobs (all edges from $E[V_i, V_{i'}] \cap \delta(v_i)$ except for $\{v_i, v_{i'}\}$). In both cases, the capacity constraint is satisfied. Symetrically, at time $4 \cdot (q-1) + 5$, the load is either 0 (if $i \notin f(q)$ or $f(q) = \{i, i'\}$ for some $i' > i$) or r (if $f(q) = \{i, i'\}$ for some $i' < i$; in this case, job J_{v_i} and $r - 1$ edge jobs (all edges from $E[V_i, V_{i'}] \cap \delta(v_i)$ except for $\{v_i, v_{i'}\}$). In both cases, the capacity constraint is satisfied.

Lastly, we show that if $(T, \mathcal{J}, c, k)$ is a yes-instance to WTR, then G is a yes-instance to Multicolored Clique. Since $(T, \mathcal{J}, c, k)$ is a yes-instance, there is a subset $\mathcal{J}' \subseteq \mathcal{J}$ of jobs with $|\mathcal{J}'| \leq \ell$ such that removing the unique occurrence of x in the corresponding strings results in a plan that satisfies the capacity constraints c. As the capacity of z_i for $i \in [\ell]$ at time step 2 is $|V_1| - 1$, it follows that for each $i \in [1, \ell]$, there exists some $v_i \in V_i$ with $J_{v_i} \in \mathcal{J}'$. As $c(z_i, 1) = 1$, this v_i is unique.

We next show that $\{v_1, \ldots, v_\ell\}$ forms a multicolored clique. Consider any $q \in [1, \binom{\ell}{2}]$ and let $f(q) = \{i_1, i_2\}$ for some $i_1 < i_2$. At time $4 \cdot (q-1) + 3$, job $J_{v_{i_1}}$ contributes one to the load of v_{i_1}. For each edge $e \in E[V_{i_1}, V_{i_2}] \cap \delta(v_{i_1})$, job J_e contributes 1 to the load of v_{i_1} at time $4 \cdot (q-1) + 3$ if and only if $J_e \notin \mathcal{J}'$. As $|E[V_{i_1}, V_{i_2}] \cap \delta(v_{i_1})| = r = c(v_i, 4 \cdot (q-1) + 3)$, this implies that there is at least one edge $e \in E[V_{i_1}, V_{i_2}] \cap \delta(v_{i_1})$ with $J_e \in \mathcal{J}'$. Considering the letter v_{i_2} and time $4 \cdot (q-1) + 5$, symmetric arguments show that there is at least one edge $e \in E[V_{i_1}, V_{i_2}] \cap \delta(v_{i_2})$ with $J_e \in \mathcal{J}'$. As there are ℓ deletions in jobs J_v for $v \in V$ and $k - \ell = \binom{\ell}{2}$, this implies that for each $i_1 < i_2 \in [\ell]$, there is a unique $e \in E[V_i, V_{i'}] \cap \delta(v_{i_1}) \cap \delta(v_{i_2})$ with $J_e \in \mathcal{J}'$, and this edge is incident to both v_{i_1} and v_{i_2}, i.e., $e = \{v_{i_1}, v_{i_2}\}$. This implies that $\{v_1, \ldots, v_\ell\}$ is a multicolored clique. ◀

5 Conclusion

In this work, we initiated theoretical studies of repairing production schedules by the deletion of waiting times in a job shop setting by introducing a mathematical formulation of the problem. While our results mostly highlight the hardness of this problem, there are also few tractability results, for example when there are only few different machine types and a short time horizon is considered. We conclude with several proposals for further research questions:

- Instead of removing waiting times, there are also other natural operations to modify a schedule, for example swapping jobs. What is the complexity of the resulting problem?
- Our scheduling model is quite general, being based on job shop scheduling. Does basing the problem on simpler scheduling models, e.g. flow shops, result in a more tractable problem?
- Given that eliminating all capacity violations turned out to be quite hard, one might try to not directly search for a "perfect" schedule, but instead only a "better" one (i.e., one with fewer capacity violations). Can such a schedule be found using the framework of parameterized local search?
- While we study how to remove pre-planned waiting times, it might be interesting to study how these waiting times can be established in the first place. How can we create schedules that are robust in the sense that they can be repaired by removing waiting times?

References

1 Alessandro Agnetis, Jean-Charles Billaut, Michael Pinedo, and Dvir Shabtay. Fifty years of research in scheduling – Theory and applications. *European Journal of Operational Research*, 2025. doi:10.1016/j.ejor.2025.01.034.

2 Susanne Albers and Günter Schmidt. Scheduling with unexpected machine breakdowns. *Discret. Appl. Math.*, 110(2-3):85–99, 2001. doi:10.1016/S0166-218X(00)00266-3.

3 Kaja Balzereit, Niels Grüttemeier, Nils Morawietz, Dennis Reinhardt, Stefan Windmann, and Petra Wolf. Scalable neighborhood local search for single-machine scheduling with family setup times. *CoRR*, abs/2409.00771, 2024. doi:10.48550/arXiv.2409.00771.

4 Nicolas Boria and Federico Della Croce. Reoptimization in machine scheduling. *Theor. Comput. Sci.*, 540:13–26, 2014. doi:10.1016/J.TCS.2014.04.004.

5 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

6 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and ids. *ACM Trans. Algorithms*, 11(2):13:1–13:20, 2014. doi:10.1145/2650261.

7 M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.

8 Kevin D Glazebrook. Scheduling tasks with exponential service times on parallel processors. *J. Appl. Probab.*, 16(3):685–689, 1979.

9 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/MOOR.8.4.538.

10 Bala Kalyanasundaram and Kirk Pruhs. Fault-tolerant scheduling. *SIAM J. Comput.*, 34(3):697–719, 2005. doi:10.1137/S0097539794261799.

11 Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Problems*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.

12 Bernhard Korte and Jens Vygen. *Combinatorial Optimization*. Springer, 6th edition, 2018.

13 Patricio Lamas, Marcos Goycoolea, Bernardo K. Pagnoncelli, and Alexandra M. Newman. A target-time-windows technique for project scheduling under uncertainty. *Eur. J. Oper. Res.*, 314(2):792–806, 2024. doi:10.1016/J.EJOR.2023.10.027.

14 Arthur Müller and Lukas Vollenkemper. Reinforcement learning as an improvement heuristic for real-world production scheduling. In *Proceedings of the 23rd IEEE International Conference on Machine Learning and Applications (ICMLA '21)*. IEEE, 2024. doi:10.1109/ICMLA52953.2021.00085.

15 Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. Comput. Syst. Sci.*, 67(4):757–771, 2003. doi:10.1016/S0022-0000(03)00078-3.

16 George A. Rovithakis, Stelios E. Perrakis, and Manolis A. Christodoulou. Application of a neural-network scheduler on a real manufacturing system. *IEEE Trans. Control. Syst. Technol.*, 9(2):261–270, 2001. doi:10.1109/87.911378.

17 Markus W. Schäffter. Scheduling with forbidden sets. *Discret. Appl. Math.*, 72(1-2):155–166, 1997. doi:10.1016/S0166-218X(96)00042-X.

18 Dvir Shabtay and Miri Gilenson. A state-of-the-art survey on multi-scenario scheduling. *Eur. J. Oper. Res.*, 310(1):3–23, 2023. doi:10.1016/J.EJOR.2022.11.014.

19 Martin Skutella, Maxim Sviridenko, and Marc Uetz. Unrelated machine scheduling with stochastic processing times. *Math. Oper. Res.*, 41(3):851–864, 2016. doi:10.1287/MOOR.2015.0757.

20 Stephen F Smith. Reactive scheduling systems. *Intelligent scheduling systems*, pages 155–192, 1995.

21 V Subramaniam, AS Raheja, and K Rama Bhupal Reddy. Reactive repair tool for job shop schedules. *Int. J. Prod. Res.*, 43(1):1–23, 2005.

22 E. Szelke and G. Markus. A blackboard based perspective of reactive scheduling. In *Artificial Intelligence in Reactive Scheduling*, pages 60–77. Springer US, 1995. doi:10.1007/978-0-387-34928-2_6.

23 Vadim G Vizing. On an estimate of the chromatic class of a p-graph. *Diskret. analiz.*, 3:25–30, 1964.

Fantastic Flips and Where to Find Them: A General Framework for Parameterized Local Search on Partitioning Problems

Niels Grüttemeier
Fraunhofer IOSB-INA, Lemgo, Germany

Nils Morawietz
Institute of Computer Science, Friedrich Schiller University Jena, Germany
LaBRI, Université de Bordeaux, Talence, France

Frank Sommer
Institute of Logic and Computation, TU Wien, Austria

Abstract

Parameterized local search combines classic local search heuristics with the paradigm of parameterized algorithmics. While most local search algorithms aim to improve given solutions by performing one single operation on a given solution, the parameterized approach aims to improve a solution by performing k simultaneous operations. Herein, k is a parameter called *search radius* for which the value can be chosen by a user. One major goal in the field of parameterized local search is to outline the trade-off between the size of k and the running time of the local search step.

In this work, we introduce an abstract framework that generalizes natural parameterized local search approaches for a large class of partitioning problems: Given n items that are partitioned into b bins and a target function that evaluates the quality of the current partition, one asks whether it is possible to improve the solution by removing up to k items from their current bins and reassigning them to other bins. Among others, our framework applies for the local search versions of problems like Cluster Editing, Vector Bin Packing, and Nash Social Welfare. Motivated by a real-world application of the problem Vector Bin Packing, we introduce a parameter called *number of types* $\tau \leq n$ and show that all problems fitting in our framework can be solved in $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time, where $|I|$ denotes the total input size. In case of Cluster Editing, the parameter τ generalizes the well-known parameter neighborhood diversity of the input graph.

We complement these algorithms by showing that for all considered problems, an algorithm significantly improving over our algorithm with running time $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ would contradict the Exponential Time Hypothesis. Additionally, we show that even on very restricted instances, all considered problems are W[1]-hard when parameterized by the search radius k alone. In case of the local search version of Vector Bin Packing, we provide an even stronger W[1]-hardness result.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Computing methodologies → Discrete space search

Keywords and phrases Flip-Neighborhood, Cluster Editing, Vector Bin Packing, Vertex Cover, NP-hard problem, Max c-Cut

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.32

Related Version *Continuously Updated Version*: `https://arxiv.org/abs/2506.24001`

Funding *Niels Grüttemeier*: Supported by the project *Datenfabrik.NRW*, a project by *KI.NRW*, funded by the Ministry for Economics, Innovation, Digitalization and Energy of the State of North Rhine-Westphalia (MWIDE).
Nils Morawietz: Partially supported by the French ANR, project ANR-22-CE48-0001 (TEMPOGRAL).
Frank Sommer: Supported by the Alexander von Humboldt Foundation.

Acknowledgements We would like to thank Lea Hoeing, Kai Lebbing, Christiane Markuse-Schneider, and Lukas Ptock (Schmitz Cargobull AG) for showing us the interesting application of Vector Bin Packing.

© Niels Grüttemeier, Nils Morawietz, and Frank Sommer;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

The principle of local search is among the most important heuristic approaches in combinatorial optimization and it is highly relevant to find good solutions to NP-hard problems in practice [19]. The idea is to apply small modifications on a given starting solution to obtain a new solution with a better target value than the starting solution. Local search has been studied extensively and it has been proven to be highly efficient [2, 19, 26, 1]. Furthermore, it is easy to understand and it is also a good plugin to improve already competitive solutions provided by other metaheuristics [31, 10].

Consider a classic partitioning problem as Multi Knapsack, where the goal is to assign items (each with a weight) to multiple knapsacks (each with a weight capacity and specific values for the items) in a way that all capacity constraints are satisfied and the total value is maximal. One of the most natural ways to apply small modifications in a local search scenario is an *item flip*, where one removes a single item from its current knapsack and inserts it into another knapsack. This natural idea of flipping the assignment of single items to improve a solution is studied for knapsack problems [4] and for other partitioning problems [8, 27]. A general drawback in performing these single item flips – and also in local search in general – is the chance of getting stuck in poor local optimal solutions. Thus, to obtain a robust local search application, a strategy to prevent getting stuck in these poor local optima is required.

In this work, we consider the approach of *parameterized local search* [29, 7] to decrease the chance of getting stuck in poor local optima reached by item flips. Instead of searching for an improving solution by *one* single operation (here: a single item flip), the user may set a search radius k to extend the search space for possible improvements that can be reached with up to k simultaneous operations. Thus, the idea is to make the search space larger so that getting stuck in poor local optima becomes less likely. Parameterized local search combines local search with the paradigm of parameterized algorithmics [3] and aims to outline the trade-off between the size of the search radius k and the running time of the search step. Parameterized local search has been studied extensively in the algorithmic community: for vertex deletion and partitioning problem in graphs [7, 13, 16, 18, 5, 21, 11, 10], for problems on strings and phylogenetic networks [17, 23], and many other problems [29, 32, 9, 15]. One major goal in parameterized local search is to show that finding an improvement within search radius k is fixed parameter tractable (FPT) for k. That is, finding an improving solution can be done in time $g(k) \cdot |I|^{\mathcal{O}(1)}$, where $|I|$ is the total encoding length of the input instance and g is some computational function only depending on k. Note that an algorithm with such a running time nicely outlines the trade-off between radius size and running time, as the superpolynomial part only depends on k while $|I|$ only contributes as a polynomial factor to the running time. Unfortunately, most parameterized local search problems are W[1]-hard for k [29, 7, 17, 30] and therefore, an algorithm with running time $g(k) \cdot |I|^{\mathcal{O}(1)}$ presumably does not exist.

Motivated by the negative results for the parameter k, one often studies the combination of k and some structural parameter τ to obtain algorithms with running $g(k, \tau) \cdot |I|^{\mathcal{O}(1)}$. This approach has been successful both from a theoretical and experimental perspective [18, 21, 12, 10]. In this work, we follow this direction by studying parameterization by k and an additional parameter τ that we call the *number of types*. We consider the local search versions of a large class of well-known combinatorial problems including Max c-Cut, Multi Knapsack, Cluster Editing and Vector Bin Packing, where the search space is defined by performing k flips. Recall that one flip intuitively removes one item from its assigned set and inserts in into some other set. The interpretation of flips and of our parameter τ always depends on the concrete problem and will be explained for each problem individually.

Our approach of exploiting the parameter τ is motivated by a real-world production planning application at the company Schmitz Cargobull AG. Planning the production at Schmitz Cargobull AG corresponds to solving an instance of Vector Bin Packing [28]. In Vector Bin Packing, one is given a large collection of vectors $\mathcal{S} \subseteq \mathbb{N}^d$ together with a vector $w \in \mathbb{N}^d$ and an integer b. The question is, whether there exists a partition of $\mathcal{S}$ into b parts $S_1, \ldots, S_b$, such that $\sum_{v \in S_i} v \leq w$ for all $i \in [1, b]$. In the application at Schmitz Cargobull AG, we have $\mathcal{S} \subseteq \{0,1\}^d$ and the vectors $v \in \mathcal{S}$ correspond to customer orders where the entries of v specify whether a specific option is chosen ($\widehat{=}$ the entry has value 1) or not ($\widehat{=}$ the entry has value 0). The entries of w correspond to the production capacities available for the corresponding product option at one day of production. Instead of asking for a partition where each resulting vector set satisfies $\sum_{v \in S_i} v \leq w$, one asks for a partition minimizing the *total overload*. More precisely, the *overload* of a single set S_i is defined as $\sum_{j=1}^{d} \max\left(0, \left(\sum_{v \in S_i} v_j\right) - w_j\right)$, and the *total overload* is the sum of all overloads of all sets $S_1, \ldots, S_b$. Note that the total overload is 0 if and only if the given instance is a yes-instance of the decision version of Vector Bin Packing. While Schmitz Cargobull AG usually receives a large number of customer orders, relatively many of these orders request the exact same product option combinations. Consequently, the number of distinct vectors in $\mathcal{S}$ is much smaller than $|\mathcal{S}|$. Therefore, our research is motivated by setting τ to be the number of distinct vectors in an instance of Vector Bin Packing and study parameterized local search for the combination of k and τ where the target is to minimize the total overload.

From a more abstract point of view, Vector Bin Packing is a problem where one assigns a collection of *items* (vectors) to a collection of *bins* (sets of the resulting partition) in a way that a target function (total overload) is minimized. Furthermore, if two elements from $\mathcal{S}$ have the same vector, these elements have the same effect on the target function. In other words, there are only τ distinct ways in which a specific item might have an influence on the target function when assigned to a specific bin. This more abstract view leads to a framework providing running times $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ and $k^{\mathcal{O}(\tau)} \cdot |I|^{\mathcal{O}(1)}$ for the parameterized local search versions for a wide range of well-known combinatorial problems that behave in the same way as Vector Bin Packing. The $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ running-time is particularly motivated since the value of k is a small constant chosen by the user [24].

Recall that the concrete interpretation of the parameter τ, of the items, and of the bins always depends on the concrete problem and will be explained for each problem individually. Besides the number of distinct vectors, τ can be interpreted as the neighborhood diversity of an input graph in case of Max c-Cut or Cluster Editing, or as the number of distinct value-weight combinations in case of Multi Knapsack.

Our Contributions. In the first part (see Section 3), we introduce the LS Generalized Bin Problem, which is a general framework capturing many parameterized local search problems where the search radius is defined by a number of item flips. Moreover, we introduce the parameter *number of types* τ as an abstract concept for LS Generalized Bin Problem capturing well-known parameters such as the neighborhood diversity of a graph. We describe general algorithms for the abstract LS Generalized Bin Problem leading to running times of $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ and $k^{\mathcal{O}(\tau)} \cdot |I|^{\mathcal{O}(1)}$ for the local search versions of many problems that fit into our framework (see Theorem 3.5).

In the second part (see Section 4) we provide simple example applications of the introduced framework leading to new results for the parameterized local search versions of classic combinatorial optimization problems, graph problems, and one problem from computational social choice (Nash Social Welfare). The formal problem definitions can be found in

Table 1 The corresponding local search problems for the flip distance of these problems can be solved in $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time and in $k^{\mathcal{O}(\tau)} \cdot |I|^{\mathcal{O}(1)}$ time for the respective parameter τ as we show in Section 4. In case of Vertex Cover, we have exactly two bins: the resulting vertex cover (vc) and the remaining independent set (is). In this case, the flip distance corresponds to the cardinality of the symmetric difference between the current vertex cover and the improving vertex cover.

Problem	Items	Bins	Parameter τ	Section
Max c-Cut	vertices	color classes	neighborhood diversity	4.1
Cluster Editing	vertices	clusters	neighborhood diversity	4.2
Vector Bin Packing	vectors	bins	number of distinct vectors	4.3
Vertex Cover	vertices	(vc, is)	neighborhood diversity	4.4
Nash Social Welfare	items	agents	number of distinct items	4.5
Multi Knapsack	items	knapsacks	number of distinct items	4.5

Section 4. All results in this part mainly rely on simply reformulating the concrete problem as LS Generalized Bin Problem. An overview of the studied problems is given in Table 1. To the best of our knowledge, this is the first work studying parameterized local search for Nash Social Welfare, Vector Bin Packing, and Multi Knapsack.

In Section 5, we complement our results by studying parameterization by the search radius k alone. We show that the parameterized local search versions of Nash Social Welfare and Multi Knapsack are W[1]-hard when parameterized by k. Furthermore, we provide a strong hardness result for Vector Bin Packing showing W[1]-hardness for $k + q$, where q denotes the maximum number of non-zero-entries over all vectors from the input. The parameter q is particularly motivated by the real-world application from the company Schmitz Cargobull AG, where q is even smaller than τ. We finally show that all of our algorithms with running time $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ are tight in the sense of the *Exponential Time Hypothesis (ETH)* [20].

2 Preliminaries

For details on parameterized complexity, we refer to the standard monographs [3, 6].

For integers a and b with $a \leq b$, we define $[a, b] := \{i \in \mathbb{N} \mid a \leq i \leq b\}$. Given a set X and some integer b, we call a mapping $f : X \to [1, b]$ a *b-partition of X* or a *b-coloring of X*. For every $i \in [1, b]$, we let $f^{-1}(i) := \{x \in X \mid f(x) = i\}$. The *flip* between two b-partitions f and f' is defined as $D_{\text{flip}}(f, f') := \{x \in X \mid f(x) \neq f'(x)\}$. The *flip distance* between f and f' is then defined as $d_{\text{flip}}(f, f') := |D_{\text{flip}}(f, f')|$. For a set X, we let 2^X denote the *power set* of X.

For a graph $G = (V, E)$, by $n := |V|$ we denote the *number of vertices* and by $m := |E|$ we denote the *number of edges*. By $N(u) := \{w \in V \mid \{u, w\} \in E\}$ and by $N(S) := (\bigcup_{u \in S} N(u)) \setminus S$ we denote the *open neighborhood* of u and S, respectively. Two vertices u and w have the *same neighborhood class* if $N(u) \setminus \{w\} = N(w) \setminus \{u\}$. By $\text{nd}(G)$ we denote the number of neighborhood classes of G. A vertex set S is a *vertex cover* for G if each edge of E has at least one endpoint in S. A vertex set S is an *independent set* of G, if no edge has of E has both endpoints in S.

Proofs of statements marked with ($\star$) are deferred to the full version.

3 Generalized Parameterized Local Search for Partitioning Problems

We present a framework that captures many computational problems such as Vector Bin Packing, Cluster Editing, and Nash Social Welfare. On a high level, all these problems have in common, that one aims to partition some set X (e.g. a set of vectors, a set of vertices, or a set of items) into multiple "bins". A target function assigns a value to the resulting partition. The goal is to find a partition that minimizes (or maximizes) the target function. This section is structured as follows. We first introduce a "Generalized Bin Problem" that generalizes the computational problems studied in this work. Afterwards, we introduce a parameter called "types". Finally, we provide the algorithmic results for this parameter.

3.1 The Generalized Bin Problem

Intuitively, we aim to partition a given set X into a given number of "bins" $b \in \mathbb{N}$, and a target value specifies how good this b-partition of X is. This target value is determined by an "individual bin evaluation", which is a collection of local target values of each single bin. These values are then combined to obtain the target value for the whole b-partition of X.

▶ **Definition 3.1.** *Let X be a set, let $b \in \mathbb{N}$, and let* $\inf \in \{\infty, -\infty\}$. *An* individual bin evaluation (IBE) *is a b-tuple $(\varphi_i)_{i\in[1,b]}$ of functions $\varphi_i : 2^X \to \mathbb{Z} \cup \{\inf\}$. An IBE defines a* target value $\mathrm{val}(f)$ *for every b-partition f by*

$$\mathrm{val}(f) := \left(\bigoplus_{i=1}^{b} \varphi_i(f^{-1}(i)) \right),$$

where $\oplus$ is a commutative and associative binary operation $\oplus : \mathbb{Z}\cup\{\inf\}\times\mathbb{Z}\cup\{\inf\} \to \mathbb{Z}\cup\{\inf\}$ satisfying $a \oplus \inf = \inf \oplus a = \inf$ for all $a \in \mathbb{Z} \cup \{\inf\}$.

While our general framework works for arbitrary commutative and associative operations $\oplus$, this work only considers concrete problems where $\oplus$ is either the summation or the multiplication of integer numbers. The value $\inf \in \{\infty, -\infty\}$ corresponds to infeasible assignments of bins, for example, violating capacity constraints of a knapsack. In case of a minimization problem we consider IBE with $\inf = \infty$ and in case of a maximization problem we have $\inf = -\infty$. In the remainder of this section, all problem statements and algorithms are given for the case where one aims to minimize the target function. Maximization problems are defined analogously. With Definition 3.1 at hand, we define the following general computational problem for every fixed commutative and associative operation $\oplus$.

Generalized Bin Problem
Input: An integer b, a set X, and an IBE $(\varphi_i)_{i\in[1,b]}$.
Goal: Find a b-partition f that minimizes $\mathrm{val}(f)$.

Note that we did not yet specify how the IBE from the input is given. As this depends on the concrete problems, this will be discussed for each problem individually. Throughout this section, we analyze the algorithms running times with respect to the parameter Φ denoting the running time needed for one evaluation of a value $\varphi_i(X')$ with $X' \subseteq X$.

Recall that our aim is to study parameterized local search for the flip neighborhood. More precisely, we aim to find a b-partition f' that has a better target value than some given b-partition f, while $d_{\mathrm{flip}}(f, f') \leq k$ for a given k. The corresponding computational problem is defined as follows.

LS Generalized Bin Problem
Input: An integer b, a set X, and an IBE $(\varphi_i)_{i\in[1,b]}$, a b-partition $f : X \to [1, b]$, and an integer k.
Question: Is there a b-partition f' with $d_{\text{flip}}(f, f') \leq k$ such that $\text{val}(f') < \text{val}(f)$?

3.2 Types in Generalized Bins

We next define a parameter called "number of types" τ. The idea is, that in a polynomial-time preprocessing step, the set X is partitioned into classes of elements $(X_1, \ldots, X_\tau)$ in a way that all elements in each X_i have the exact same impact on the target value for every possible bin assignment. The intuitive idea of "same impact" is formalized as follows.

▶ **Definition 3.2.** *Let X be a set, let $b \in \mathbb{N}$, and let $(\varphi_i)_{i\in[1,b]}$ be an IBE for X and b. Two (not necessarily distinct) elements $x \in X$ and $y \in X$ are* target equivalent *($x \sim y$), if for every $i \in [1, b]$ and for every $A \subseteq X$ with $\{x, y\} \cap A = \{x\}$ we have $\varphi_i((A \setminus \{x\}) \cup \{y\}) = \varphi_i(A)$.*

▶ **Proposition 3.3 ($\star$).** *The relation $\sim$ is an equivalence relation on X.*

The main idea of our algorithm is that target equivalent elements can be treated equally as they have the same influence on the target value. When considering concrete problems, we always use a simple pairwise relation between the elements leading to a partition of X into classes of pairwise target equivalent elements. These classes do not necessarily need to be maximal under this constraint. Thus, it suffices to consider the following relaxation of the equivalence classes.

▶ **Definition 3.4.** *Let X be a set, let $b \in \mathbb{N}$, and let $(\varphi_i)_{i\in[1,b]}$ be an IBE for X and b. A tuple $(X_1, \ldots, X_\tau)$ of disjoint sets with $\bigcup_{j=1}^{\tau} X_j = X$ is called a* type partition *of X if the elements of each X_j are pairwise target equivalent. For every $j \in [1, \tau]$, we say that the elements of X_j* have type j.

Obviously, the equivalence classes for $\sim$ always form a type partition with the minimum number of sets. Throughout this work, we assume that each instance $I := (b, X, (\varphi_i)_i, f, k)$ of LS Generalized Bin Problem is associated with a specified type partition, and the parameter *number of types* $\tau := \tau(I)$ is defined as the number of sets of the type partition associated with I.

3.3 Algorithmic Results for LS Generalized Bin Problem

Our goal is to study LS Generalized Bin Problem parameterized by $k + \tau$. Applying this on concrete problems then leads to FPT algorithms for a great range of parameterized local search versions of well-known computational problems. The interpretation of the parameter τ always depends on the concrete problem. We provide the following algorithmic results. Recall that Φ denotes the running time needed for one evaluation of a value $\varphi_i(X')$ with $X' \subseteq X$.

▶ **Theorem 3.5.** *LS Generalized Bin Problem can be solved*
a) *in $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$ time, and*
b) *in $k^{\mathcal{O}(\tau)} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$ time*
if a type partition $(X_1, \ldots, X_\tau)$ is additionally given as part of the input.

To present the algorithms behind Theorem 3.5, we introduce the notion of *type specification* and *type specification operations*. Recall that elements of the same type behave in the same way when assigned to a bin. Thus, when evaluating the target function, one may consider

the types of the assigned elements instead of the concrete elements. A type specification is a vector $\vec{p}$ that intuitively corresponds to a collection of elements in X containing exactly p_j elements from the class X_j.

▶ **Definition 3.6.** *Let b be an integer, let X be a set, and let $(\varphi_i)_{i\in[1,b]}$ be an IBE. Moreover, let $(X_1, \ldots, X_\tau)$ be a type partition. A* type specification *is a vector $\vec{p} = (p_1, \ldots, p_\tau) \in {\mathbb{N}_0}^\tau$ with $p_j \in [0, |X_j|]$ for each $j \in [1, \tau]$.*

Since elements of the same type have the same impact on the target function, we can address the change of target values $\varphi_i(X')$ by just specifying the types of elements added to and removed from X' with $X' \subseteq X$. To ensure that after adding and removing elements of specific types from a set X' corresponds to an actual subset of X, we introduce the notion of subtractive and additive compatibility.

▶ **Definition 3.7.** *Let b be an integer, let X be a set, and let $(\varphi_i)_{i\in[1,b]}$ be an IBE. Moreover, let $(X_1, \ldots, X_\tau)$ be a type partition. We say that a type specification $\vec{p}$ is*
a) subtractive compatible *with a set $X' \subseteq X$ if for every $j \in [1, \tau]$, we have $|X_j \cap X'| \geq p_j$.*
b) additive compatible *with a set $X' \subseteq X$ if for every $j \in [1, \tau]$, we have $|X_j \cap (X \setminus X')| \geq p_j$.*
We use $(\vec{p}, \vec{q}) \propto X'$ to denote that $\vec{p}$ is subtractive compatible with X' and $\vec{q}$ is additive compatible with X'. Given $\vec{p}$ and $\vec{q}$ with $(\vec{p}, \vec{q}) \propto X'$, the type vector operation *for each $i \in [1, b]$ is defined as*

$$\varphi_i((X' \setminus \vec{p}) \cup \vec{q}) := \varphi_i((X' \setminus X_{\vec{p}}) \cup X_{\vec{q}}),$$

where $X_{\vec{p}} \subseteq X$ is some set containing exactly p_j arbitrary elements from $X_j \cap X'$ for every $j \in [1, \tau]$, and $X_{\vec{q}} \subseteq X$ is some set containing exactly q_j arbitrary elements from $X_j \cap (X \setminus X')$ for every $j \in [1, \tau]$.

The notion of target equivalence (Definition 3.2) together with the notion of subtractive and additive compatibility guarantees the following.

▶ **Proposition 3.8.** *The type vector operation is well-defined.*

Proof. Since $\vec{p}$ is subtractive compatible with X', there are at least p_j elements in $X_j \cap X'$ and thus, the set $X_{\vec{p}} \subseteq X$ exists. Analogously, the set $X_{\vec{q}}$ exists due to the additive compatibility of $\vec{q}$ with X'. Consequently, $(X' \setminus X_{\vec{p}}) \cup X_{\vec{q}} \subseteq X$ and therefore, $\varphi_i((X' \setminus X_{\vec{p}}) \cup X_{\vec{q}})$ is defined.

It remains to show that the value $\varphi_i((X' \setminus X_{\vec{p}}) \cup X_{\vec{q}})$ does not depend on the choice of elements in $X_{\vec{p}}$ and $X_{\vec{q}}$. First, let $x \in X_{\vec{p}}$ and let $y \notin X_{\vec{p}}$ with $y \sim x$. Then,

$$\varphi_i(X' \setminus (X_{\vec{p}} \setminus \{x\} \cup \{y\}) \cup X_{\vec{q}}) = \varphi_i((X' \setminus X_{\vec{p}} \cup X_{\vec{q}}) \setminus \{y\} \cup \{x\}) = \varphi_i(X' \setminus X_{\vec{p}} \cup X_{\vec{q}}),$$

by Definition 3.2. Analogously, let $x \in X_{\vec{q}}$ and let $y \notin X_{\vec{q}}$ with $x \sim y$ we have

$$\varphi_i(X' \setminus X_{\vec{p}} \cup (X_{\vec{q}} \setminus \{x\} \cup \{y\})) = \varphi_i((X' \setminus X_{\vec{p}} \cup X_{\vec{q}}) \setminus \{x\} \cup \{y\}) = \varphi_i(X' \setminus X_{\vec{p}} \cup X_{\vec{q}}).$$

Therefore, the type vector operation is well-defined. ◀

With the notion of type specifications at hand, we next present the algorithmic results. Intuitively, the algorithms behind Theorem 3.5 work as follows: One considers every possibility of how many elements of which type belong to the (at most) k elements in $D_{\text{flip}}(f, f')$. For each such choice, one computes the best improving flip corresponding to the choice. Note that the information how many elements of which type are flipped, can be encoded as a type specification $\vec{\delta}$ in the way that all δ_j correspond to the number of flipped elements of type j.

We first describe the subroutine computing the improving b-partition f' corresponding to a given type specification $\vec{\delta}$. Afterwards, we describe how to efficiently iterate over all possible $\vec{\delta}$ to obtain the running times stated in Theorem 3.5.

We use the notation $\vec{p} \leq \vec{q}$ to indicate that $p_i \leq q_i$ for all vector entries, and we let $\vec{p} - \vec{q}$ denote the component-wise difference between $\vec{p}$ and $\vec{q}$.

▶ **Lemma 3.9.** *Let b be an integer, let X be a set, let $(\varphi_i)_{i\in[1,b]}$ be an IBE, and let $f : X \to [1, b]$ be a b-partition. Moreover, let $(X_1, \ldots, X_\tau)$ be a type partition and let $\vec{\delta}$ be a type specification and we let $k := \sum_{j=1}^{\tau} \delta_j$. A b-partition $f' : X \to [1, b]$ with*

$$(|X_1 \cap D_{\text{flip}}(f, f')|, \ldots, |X_\tau \cap D_{\text{flip}}(f, f')|) \leq \vec{\delta}$$

that minimizes $\text{val}(f')$ *among all such b-partitions can be computed in*
a) $2^{\mathcal{O}(k)} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$ *time, or in*
b) $(\lceil \frac{k}{\tau} \rceil + 1)^{4\tau} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$ *time.*

Proof. We first provide an algorithm based on dynamic programming and afterwards we discuss the running times *a*) and *b*).

Intuition. Before we formally describe the dynamic programming algorithm, we provide some intuition: Every $x \in D_{\text{flip}}(f, f')$ gets removed from its bin and is inserted into a new bin. We fix an arbitrary ordering of the bins and compute solutions for prefixes of this ordering in a bottom-up manner. For every bin, there is a set of removed elements R and a set of inserted elements I. Since the target value depends on the types of the elements rather than the concrete sets R and I, we may only consider how many elements of which type are removed from a bin and inserted into a bin. Therefore, we compute the partial solutions for given $\vec{p} \leq \vec{\delta}$ and $\vec{q} \leq \vec{\delta}$, where $\vec{p}$ corresponds to the removed types, and $\vec{q}$ corresponds to the inserted types. After the computation, we consider the solution, where $\vec{p} = \vec{q} = \vec{\delta}$, that is, the solution where the exact same types were removed and inserted. Going back from abstract types to concrete elements then yields the resulting b-partition f'.

Dynamic programming algorithm. The dynamic programming table has entries of the form $T[\vec{p}, \vec{q}, \ell]$ where $\ell \in [1, b]$, $\vec{q} \leq \vec{\delta}$, and $\vec{p} \leq \vec{\delta}$. One such table entry corresponds to the minimal value of

$$\bigoplus_{i=1}^{\ell} \varphi_i((f^{-1}(i) \setminus R_i) \cup I_i)$$

under every choice of sets $R_i \subseteq f^{-1}(i)$ and $I_i \subseteq X \setminus f^{-1}(i)$ for $i \in [1, \ell]$ that satisfy

$$(\sum_{i=1}^{\ell} |R_i \cap X_1|, \ldots, \sum_{i=1}^{\ell} |R_i \cap X_\tau|) = \vec{p} \quad \text{and} \quad (\sum_{i=1}^{\ell} |I_i \cap X_1|, \ldots, \sum_{i=1}^{\ell} |I_i \cap X_\tau|) = \vec{q}.$$

The table is filled for increasing values of ℓ. As base case, we set $T[\vec{p}, \vec{q}, 1] := \varphi_1((f^{-1}(1) \setminus \vec{p}) \cup \vec{q})$ if $(\vec{p}, \vec{q}) \propto f^{-1}(1)$. Recall that $(\vec{p'}, \vec{q'}) \propto f^{-1}(1)$ is true if and only if $\vec{p'}$ is subtractive compatible with $f^{-1}(1)$ and $\vec{q'}$ is additive compatible with $f^{-1}(1)$. If $\vec{p}$ or $\vec{q}$ is incompatible, we set $T[\vec{p}, \vec{q}, 1] := \infty$. The recurrence to compute an entry with $\ell > 1$ is

$$T[\vec{p}, \vec{q}, \ell] := \min_{\substack{\vec{p'} \leq \vec{p},\ \vec{q'} \leq \vec{q} \\ (\vec{p'}, \vec{q'}) \propto f^{-1}(\ell)}} T[\vec{p} - \vec{p'}, \vec{q} - \vec{q'}, \ell - 1] \oplus \varphi_\ell((f^{-1}(\ell) \setminus \vec{p'}) \cup \vec{q'}),$$

Note that $(\vec{0}, \vec{0}) \propto f^{-1}(\ell)$ is always true, and therefore, such a minimum always exists. Herein, $\vec{0}$ denotes the τ-dimensional vector where each entry is 0.

▷ Claim 3.10. The recurrence is correct.

Proof. Throughout the proof of this claim, we call the properties posed on the sets $(R_1, \ldots, R_\ell)$ and $(I_1, \ldots, I_\ell)$ in the definition of the table entries the *desired properties for $\vec{p}$, $\vec{q}$, and ℓ*.

We show that the recurrence is correct via induction over ℓ. If $\ell = 1$, we have $T[\vec{p}, \vec{q}, 1] := \varphi_1((f^{-1}(1) \setminus \vec{p}) \cup \vec{q})$. Thus, by Definition 3.7, there are sets $R \subseteq f^{-1}(1)$ and $I \subseteq X \setminus f^{-1}(1)$ where R is a set containing exactly p_j elements from $X_j \cap f^{-1}(1)$ for every $j \in [1, \tau]$, and I is a set containing exactly q_j elements from $X_j \cap (X \setminus f^{-1}(1))$ for every $j \in [1, \tau]$ and we have $T[\vec{p}, \vec{q}, 1] = \varphi_1((f^{-1}(1) \setminus R) \cup I)$. Since the value is invariant under the concrete choices of elements in R and I due to Proposition 3.8, the value $\varphi_1((f^{-1}(1) \setminus R) \cup I)$ is minimal among all such R and I. Thus, the Recurrence holds for the base case $\ell = 1$.

Next, let $\ell > 1$ and assume that the recurrence holds for $\ell - 1$. Let $(R_1, \ldots, R_\ell)$ and $(I_1, \ldots, I_\ell)$ be sets with the desired properties for $\vec{p}$, $\vec{q}$, and ℓ. We show

$$\bigoplus_{i=1}^{\ell} \varphi_i((f^{-1}(i) \setminus R_i) \cup I_i) = T[\vec{p}, \vec{q}, \ell].$$

($\geq$) Since $\oplus$ is associative and commutative, we have

$$\bigoplus_{i=1}^{\ell} \varphi_i((f^{-1}(i) \setminus R_i) \cup I_i) = \underbrace{\left(\bigoplus_{i=1}^{\ell-1} \varphi_i((f^{-1}(i) \setminus R_i) \cup I_i)\right)}_{=:Z} \oplus \varphi_\ell((f^{-1}(\ell) \setminus R_\ell) \cup I_\ell).$$

Note that the vectors $\vec{a} := (|R_\ell \cap X_1|, \ldots, |R_\ell \cap X_\tau|)$ and $\vec{b} := (|I_\ell \cap X_1|, \ldots, |I_\ell \cap X_\tau|)$ satisfy $\vec{a} \leq \vec{p}$ and $\vec{b} \leq \vec{q}$. Moreover, since $R_\ell \subseteq f^{-1}(\ell)$ and $I_\ell \subseteq X \setminus f^{-1}(\ell)$, it holds that $(\vec{a}, \vec{b}) \propto f^{-1}(\ell)$ is true. Thus, the minimum of the right-hand-side of the recurrence includes $\vec{a}$ and $\vec{b}$, and by the definition of type specification operations and Proposition 3.8, we have $\varphi(\ell)((f^{-1}(\ell) \setminus \vec{a}) \cup \vec{q}) = \varphi(\ell)((f^{-1}(\ell) \setminus R_\ell) \cup I_\ell)$. Since by the inductive hypothesis we have $T[\vec{p} - \vec{a}, \vec{q} - \vec{b}, \ell - 1] \leq Z$, we conclude $\bigoplus_{i=1}^{\ell} \varphi_i((f^{-1}(i) \setminus R_i) \cup I_i) \geq T[\vec{p}, \vec{q}, \ell]$.

($\leq$) Let $\vec{a}$ and $\vec{b}$ be the vectors minimizing the right-hand-side of the recurrence. By the definition of type specification operations, there are sets $\widetilde{R_\ell} \subseteq f^{-1}(\ell)$ and $\widetilde{I_\ell} \subseteq X \setminus f^{-1}(\ell)$ with $(|\widetilde{R_\ell} \cap X_1|, \ldots, |\widetilde{R_\ell} \cap X_\tau|) = \vec{a}$ and $(|\widetilde{I_\ell} \cap X_1|, \ldots, |\widetilde{I_\ell} \cap X_\tau|) = \vec{b}$. By inductive hypothesis, the value $T[\vec{p} - \vec{a}, \vec{q} - \vec{b}, \ell - 1]$ corresponds to the minimal value of $\bigoplus_{i=1}^{\ell-1} \varphi_i((f^{-1}(i) \setminus \widetilde{R_i}) \cup \widetilde{I_i})$ for sets $(\widetilde{R_1}, \ldots, \widetilde{R_{\ell-1}})$ and $(\widetilde{I_1}, \ldots, \widetilde{I_{\ell-1}})$ that satisfy the desired properties for $\vec{p} - \vec{a}$, $\vec{q} - \vec{b}$, and $\ell - 1$. Then, $T[\vec{p}, \vec{q}, \ell]$ corresponds to the value $\bigoplus_{i=1}^{\ell} \varphi_i((f^{-1}(i) \setminus \widetilde{R_i}) \cup \widetilde{I_i})$ for sets $(\widetilde{R_1}, \ldots, \widetilde{R_\ell})$ and $(\widetilde{I_1}, \ldots, \widetilde{I_\ell})$ that satisfy the desired properties for $\vec{p}$, $\vec{q}$, and ℓ. Thus, it follows that $\bigoplus_{i=1}^{\ell} \varphi_i((f^{-1}(i) \setminus R_i \cup I_i)) \leq T[\vec{p}, \vec{q}, \ell]$. ◁

Computation of a solution f'. We next describe how to compute the desired b-partition f' after filling the dynamic programming table T. The sets $(R_1, \ldots, R_b)$ and $(I_1, \ldots, I_b)$ corresponding to the table entry $T[\vec{\delta}, \vec{\delta}, b]$ can be found via trace back. We let $\mathcal{R} := \bigcup_{i=1}^{b} R_i$. Due to the property $R_i \subseteq f^{-1}(i)$ for all $i \in [1, b]$, all R_i are disjoint. Together with the properties on the vectors $\vec{p}$ and $\vec{q}$, this implies $|\mathcal{R} \cap X_j| = \sum_{i=1}^{b} |R_i \cap X_j| = \sum_{i=1}^{b} |I_i \cap X_j| = \delta_j$ for every $j \in [1, \tau]$. Consequently, for every $j \in [1, \tau]$, there is a mapping $\gamma_j : \mathcal{R} \cap X_j \to [1, b]$, that maps exactly $|I_i \cap X_j|$ elements of $\mathcal{R} \cap X_j$ to i for every $i \in [1, b]$. Intuitively, the mapping γ_j assigns a new bin to each item of type j that was removed from some bin.

The b-partition f' is defined via the mappings γ_j as follows: For every $x \in X \setminus \mathcal{R}$, we set $f'(x) := f(x)$. For every $x \in \mathcal{R}$, we set $f'(x) := \gamma_j(x)$ for the corresponding j with $x \in \mathcal{R} \cap X_j$.

By the definition of f', the b-partitions f and f' may only differ on $\mathcal{R}$ and therefore, $|\mathcal{R} \cap X_j| = \delta_j$ implies $(|X_1 \cap D_{\text{flip}}(f,f')|, \ldots, |X_\tau \cap D_{\text{flip}}(f,f')|) \leq \vec{\delta}$. It remains to show that $\text{val}(f')$ is minimal. By the construction of f' we have $f'^{-1}(i) = f^{-1}(i) \setminus R_i \cup I'_i$, where $I'_i := \bigcup_{j=1}^{\tau} \{x \in \mathcal{R} \cap X_j \mid \gamma_j(x) = i\}$. Then, by the construction of the γ_j we have

$$\begin{aligned}(|I'_i \cap X_1|, \ldots, |I'_i \cap X_\tau|) &= (|\{x \in \mathcal{R} \cap X_1 \mid \gamma_1(x) = i\}|, \ldots, |\{x \in \mathcal{R} \cap X_\tau \mid \gamma_\tau(x) = i\}|) \\ &= (|I_i \cap X_1|, \ldots, |I_i \cap X_\tau|).\end{aligned}$$

Thus, the sets $(f^{-1}(i) \setminus R_i) \cup I'_i$ and $(f^{-1}(i) \setminus R_i) \cup I_i$ contain the exact same number of elements from each X_j. Consequently, we have $\varphi_i((f^{-1}(i) \setminus R_i) \cup I'_i) = \varphi((f^{-1}(i) \setminus R_i) \cup I_i)$ for every $i \in [1,b]$ and thus, by the definition of types, the minimality of $T[\vec{\delta}, \vec{\delta}, b]$ implies the minimality of $\text{val}(f')$.

Running time. We finally provide two different ways to analyze the running time.

a) The vector $\vec{\delta}$ can be regarded as a set M containing $\sum_{j=1}^{\tau} \delta_j = k$ individual identifiers from which exactly δ_j are labeled with type j for each $j \in [1,\tau]$. With this view on $\vec{\delta}$, the vectors $\vec{p} \leq \vec{\delta}$ and $\vec{q} \leq \vec{\delta}$ correspond to subsets of M, so we have 2^k possible choices for $\vec{p}$ and 2^k possible choices for $\vec{q}$. Consequently, the size of the table T is $2^{2k} \cdot b$. To compute one entry, one needs to consider up to 2^{2k} choices of $\vec{p'}$ and $\vec{q'}$. For each such choice, $(\vec{p'}, \vec{q'}) \propto f^{-1}(\ell)$ can be checked in $|X|^{\mathcal{O}(1)}$ time. With the evaluation of the IBE, this leads to a total running time of $2^{4k} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$.

b) The number of possible vectors that are component-wise smaller than $\vec{\delta}$ is $\prod_{j=1}^{\tau}(\delta_j + 1)$, since each entry is an element of $[0, \delta_j]$. Since $\sum_{j=1}^{\tau} \delta_j = k$, the product of all $(\delta_j + 1)$ is maximal if all δ_j have roughly the same size $\frac{k}{\tau}$. Thus, we have $\prod_{j=1}^{\tau}(\delta_j + 1) \leq (\frac{k}{\tau} + 1)^\tau$. Consequently, the size of T is upper-bounded by $(\lceil \frac{k}{\tau} \rceil + 1)^{2\tau} \cdot b$. To compute one entry, one needs to consider up to $(\lceil \frac{k}{\tau} \rceil + 1)^{2\tau}$ choices of $\vec{p'}$ and $\vec{q'}$. For each such choice, $(\vec{p'}, \vec{q'}) \propto f^{-1}(\ell)$ can be checked in $|X|^{\mathcal{O}(1)}$ time. With the evaluation of the IBE, this leads to a total running time of $(\lceil \frac{k}{\tau} \rceil + 1)^{4\tau} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$. ◀

We next use Lemma 3.9 to prove Theorem 3.5.

Proof of Theorem 3.5. Recall that the idea is to consider every possible vector $\vec{\delta}$ specifying how many elements of which type belong to the elements of the flip. For each possible $\vec{\delta}$ we then use the algorithm behind Lemma 3.9. It remains to describe how to efficiently iterate over the possible choices of $\vec{\delta}$ to obtain the running times *a*) and *b*).

a) Let $\vec{e_1}, \ldots, \vec{e_\tau}$ be the τ-dimensional unit vectors. That is, only the jth entry of $\vec{e_j}$ equals 1 and all other entries equal 0. We enumerate all $\vec{\delta}$ with $\sum_{i=1}^{\tau} \delta_i \leq k$ by considering all τ^k possibilities to repetitively draw up to k-times from the set $\{\vec{e_1}, \ldots, \vec{e_\tau}\}$. For each such choice $\vec{\delta}$, we check whether $\delta_i \leq |X_i|$ for each $i \in [1,\tau]$ to ensure that $\vec{\delta}$ is a type specification. If this is the case, we apply the algorithm behind Lemma 3.9. With the running time from Lemma 3.9 *a*), this leads to a total running time of $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$.

b) Note that for a vector $\vec{\delta}$ with $\sum_{i=1}^{\tau} \delta_i = k$, we have $\delta_i \in [0,k]$ for every $i \in [1,\tau]$. Thus, in time $(k+1)^\tau \cdot k^{\mathcal{O}(1)}$ we can enumerate all possible $\vec{\delta}$. Analogously to *a*), we check whether $\delta_i \leq |X_i|$ for each $i \in [1,\tau]$. With the running time from Lemma 3.9 *b*), this leads to a total running time of $k^{\mathcal{O}(\tau)} \cdot b \cdot \Phi \cdot |X|^{\mathcal{O}(1)}$. ◀

4 Applications of the Framework

We next study parameterized local search versions of a wide range of well-known problems. For each of these problems, we consider parameterization by the search radius in combination with some structural parameter. All results rely on the algorithm behind Theorem 3.5.

Therefore, it suffices to express an instance I of a local search problem as a corresponding instance $J = (b, X, (\varphi_i)_{i\in[1,b]}, f, k)$ of LS Generalized Bin Problem. The proofs in this section are given in the following structure:

1. *GBP Construction:* Given the instance I of the local search problem, we specify the universe X, the number of bins b and the IBE $(\varphi_i)_{i\in[1,b]}$ of the instance J. This also includes a description of how to efficiently evaluate the IBE from the input instance I.
2. *Type Partition:* We express how a type partition for J is computed from I and describe how the targeted structural parameter corresponds to the number of types as defined in Section 3.
3. *Solution Correspondence:* Recall that an instance of a local search problem always contains a solution of the underlying problem. Let s be the solution given in the instance I. To show the solution correspondence, we describe how solutions of I can be transformed into solutions of J and vice versa, such that the following holds: A solution s' of I is better than the given solution s of I if and only if the corresponding b-partition $f_{s'}$ has strictly smaller (larger) target value than f_s with respect to the IBE $(\varphi_i)_{i\in[1,b]}$.

4.1 Max c-Cut

Let $c \in \mathbb{N}$ and let $G = (V, E)$ be an undirected graph. We say that an edge $e \in E$ is properly colored by a c-partition χ of V, if χ assigns distinct colors to the endpoints of e.

Max c-Cut
Input: An undirected graph $G = (V, E)$.
Goal: Find a c-partition χ of V that maximizes the number of properly colored edges under χ.

Note that the goal of Max c-Cut can also be equivalently redefined as: Find a c-partition χ of V that minimizes the number of edges that are not properly colored under χ, that is, a c-partition χ that minimizes $\text{faults}(\chi) := \sum_{i\in[1,c]} |E_G(\chi^{-1}(i))|$.

The input of the corresponding local search problem LS Max c-Cut additionally consists of a c-partition χ and some $k \in \mathbb{N}$, and one aims to find a c-partition χ' with $\text{faults}(\chi') < \text{faults}(\chi)$ and $d_{\text{flip}}(\chi, \chi') \leq k$.

LS Max c-Cut is W[1]-hard parameterized by k [10] and cannot be solved in $f(k) \cdot n^{o(k)}$ time unless the ETH is false [30]. Form a positive side, it was shown that the problem can be solved in $2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time on apex-minor-free graphs [7]. Moreover, on general graphs, the problem can be solved in $\Delta^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time [10], where Δ denotes the maximum degree of the input graph. This algorithms found successful application as a post-processing algorithm for a state-of-the-art heuristic for Max c-Cut [10].

We show the following.

▶ **Theorem 4.1.** *LS Max c-Cut can be solved in* $\text{nd}^k \cdot 2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ *and* $k^{\mathcal{O}(\text{nd})} \cdot n^{\mathcal{O}(1)}$ *time.*

Proof. The proof relies on the algorithm behind Theorem 3.5. We describe how to use this algorithm to solve an instance $I := (G = (V, E), \chi, k)$ of LS Max c-Cut.

1. GBP Construction. We describe how to obtain an instance $J := (b, X, (\varphi_i)_{i\in[1,b]}, f, k)$ of LS Generalized Bin Problem. Our universe X is exactly the vertex set V of G, and the number b of bins is exactly the number c of colors. We next define the IBE. For each $i \in [1, c]$ and each vertex set $S \subseteq V$, we define $\varphi_i(S) := |E_G(S)|$. Note that for each value $\varphi(S)$ can obviously be computed in $n^{\mathcal{O}(1)}$ time, and thus, the IBE can be evaluated in polynomial time from the input instance I. Our operation $\oplus$ is the sum of integer numbers.

2. Type Partition. Our type partition is the collection of neighborhood classes of G. Recall that this collection can be computed in $\mathcal{O}(n+m)$ time.

To show that this collection is in fact a type partition, we show that two vertices from the same neighborhood class are target equivalent according to Definition 3.2. To this end, let C be a neighborhood class of G, let u and v be vertices of C. We show that u and v are target equivalent. Let $S \subseteq V$ be a vertex set with $S \cap \{u, v\} = \{u\}$. We show that for each $i \in [1, b]$, $\varphi_i(S) = \varphi_i((S \setminus \{u\}) \cup \{v\})$. Let $S' := (S \setminus \{u\}) \cup \{v\}$. Since all IBEs $(\varphi_i)_{i \in [1,b]}$ are identically defined, it suffices to only consider $i = 1$. By the fact that C is a neighborhood class of G, u and v have the same neighbors in $S \setminus \{u\} = S' \setminus \{v\}$. Hence, $\varphi_1(S) = \varphi_1(S')$. This implies that u and v are target equivalent.

3. Solution Correspondence. Note that the set of solutions χ' of LS Max c-Cut is exactly the set of all c-partitions of V. Since $X = V$ and $b = c$, the solutions of I have a one-to-one correspondence to the solutions of J. Moreover, note that the definition of the IBE is based on the reformulation of the objective function of Max c-Cut. Since $\oplus$ is the sum of integer numbers, we have $\text{val}(\chi') = \text{faults}(\chi')$ for every c-partition χ'. Therefore, χ' is a better solution than χ for LS Max c-Cut if and only if χ' is a smaller target value than χ with respect to the IBE. ◀

4.2 Cluster Editing

A *cluster graph* is an undirected graph in which each connected component forms a clique. Let $G = (V, E)$ be an undirected graph. In Cluster Editing, one aims to apply a minimum number of edge modification (edge insertions and edge deletions) on G, such that the resulting graph is a cluster graph. We let $\binom{V}{2}$ denote the set of two-element subsets of vertices corresponding to edge modifications. Given a set $E' \subseteq \binom{V}{2}$, we let $E \triangle E' := (E \setminus E') \cup (E' \setminus E)$ denote the symmetric difference corresponding to the application of the graph modifications.

Cluster Editing
Input: An undirected graph $G = (V, E)$.
Goal: Find a set $E' \subseteq \binom{V}{2}$ such that $(V, E \triangle E')$ is a cluster graph and $|E'|$ is minimal under this property.

Note that the maximum number of clusters corresponds to the number of vertices n. We consider an equivalent definition of Cluster Editing where one asks for an n-labeling χ with partitioning V into n (possibly empty) classes $\chi^{-1}(1), \ldots, \chi^{-1}(n)$, such that

$$\text{score}(\chi) := \sum_{i=1}^{n} |E(\chi^{-1}(i))| - \left(\binom{|\chi^{-1}(i)|}{2} - |E(\chi^{-1}(i))|\right)$$

is maximal [30]. Intuitively, maximizing this score corresponds to maximizing the number of present edges in the connected components minus the required edge insertions such that these components form a cliques.

We study a parameterized local search version of Cluster Editing where one flip in the flip neighborhood corresponds to moving single vertices from their current cluster into another cluster. The input of the corresponding local search problem LS Cluster Editing consists of the input of Cluster Editing, together with an additional integer k and an n-labeling χ of V for some. The task is to compute a n-labeling χ' such that $\text{score}(\chi') > \text{score}(\chi)$ and $d_{\text{flip}}(\chi, \chi') \leq k$.

With respect to this neighborhood, LS Cluster Editing is known to be W[1]-hard parameterized by k and cannot be solved in $f(k) \cdot n^{o(k)}$ time unless the ETH is false [11]. Form a positive side, it was shown that the problem can be solved in $\Delta^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time [11], where Δ denotes the maximum degree of the input graph.

We now show the following.

▶ **Theorem 4.2.** *LS Cluster Editing can be solved in* $\mathrm{nd}^k \cdot 2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ *time and in* $k^{\mathcal{O}(\mathrm{nd})} \cdot n^{\mathcal{O}(1)}$ *time.*

Proof. The proof relies on the algorithm behind Theorem 3.5. We describe how to use this algorithm to solve an instance $I := (G = (V, E), k, \chi)$.

1. GBP Construction. We describe how I corresponds to an LS Generalized Bin Problem-instance $J := (b, X, (\varphi_i)_{i\in[1,b]}, f, k)$. Our universe X is exactly the vertex set V of G, and the number b of bins is exactly n. We next define the IBE. For each $i \in [1, n]$ and each vertex set $S \subseteq V$, we define $\varphi_i(S) := |E(S)| - (\binom{|S|}{2} - |E(S)|)$. Note that each value $\varphi(S)$ can obviously be computed in $n^{\mathcal{O}(1)}$ time from the input graph G. Our operation $\oplus$ is the sum of integer numbers.

2. Type Partition. Our type partition is the collection of neighborhood classes of G. Recall that this collection can be computed in $\mathcal{O}(n + m)$ time.

To show that this collection is in fact a type partition, we show that two vertices from the same neighborhood class of G are target equivalent according to Definition 3.2. To this end, let C be a neighborhood class of G and let u and v be vertices of c. Let $S \subseteq V$ be a vertex set with $S \cap \{u, v\} = \{u\}$. We obviously have $|S| = |(S \setminus \{u\}) \cup \{v\}|$. Moreover, since u and v have the exact same neighbors in G, we have $|E(S)| = |E((S \setminus \{u\}) \cup \{v\})|$. Then, by the definition of the IBE we have $\varphi_i(S) = \varphi_i((S \setminus \{u\}) \cup \{v\})$ for all $i \in [1, n]$.

3. Solution Correspondence. Note that the set of solutions χ' of LS Cluster Editing is exactly the set of all n-partitions of V. Since $X = V$ and $b = n$, the solutions of J have a one-to-one correspondence to the solutions of J. Moreover, note that the definition of the IBE is based on the reformulation of the objective function of Cluster Editing and $\oplus$ is the sum of integer numbers, we have $\mathrm{val}(\chi') = \mathrm{score}(\chi')$ for every n-partition χ'. Therefore, χ' is a better solution than χ for Cluster Editing if and only if χ' has a larger target value than χ with respect to the IBE. ◀

4.3 Vector Bin Packing

Let b, d be integers. In this section for $i \in [1, b]$, B_i is a *bin*. Each bin B_i is associated with a *weight vector* $\omega_i \in \mathbb{N}^d$. Let $\mathcal{S}$ be a set of vectors from $\mathbb{N}_0^d$. A b-partition χ of $\mathcal{S}$ is called a *bin assignment*.

Given a subset $S \subseteq \mathcal{S}$, we say that the *overload of bin B_i with respect to S* is $\mathsf{ol}(B_i, S) := \sum_{j=1}^{d} \max(0, (\sum_{v\in S} v_j) - w_j)$. Our target is to find an assignment χ minimizing the total overload, which is defined as $\sum_{i\in[1,b]} \mathsf{ol}(B_i, \chi^{-1}(i))$.

Vector Bin Packing
Input: Integers b, d, a set $\mathcal{S}$ of vectors from $\mathbb{N}_0^d$, a vector $\omega_i \in \mathbb{N}_0^d$ for each $i \in [1, b]$.
Goal: Find a bin assignment χ of $\mathcal{S}$ such that the total overload is minimized, that is, $\sum_{i\in[1,b]} \mathsf{ol}(B_i, \chi^{-1}(i))$ is minimized.

The input of the corresponding local search problem LS Vector Bin Packing additionally consists of a bin assignment χ and some $k \in \mathbb{N}$, and one aims to find a bin assignment χ' with $\sum_{i\in b} \mathsf{ol}(B_i, \chi'^{-1}(i)) < \sum_{i\in b} \mathsf{ol}(B_i, \chi^{-1}(i)))$ and $d_{\mathrm{flip}}(\chi, \chi') \leq k$.

We say that two vectors vec_1 and vec_2 are *identical* if they agree in all dimensions, that is, if $\mathsf{vec}_1(j) = \mathsf{vec}_2(j)$ for each dimensions $j \in [1, d]$ and two vectors are *distinct* otherwise.

To the best of our knowledge, parameterized local search for Vector Bin Packing has not yet been studied. We now show the following.

▶ **Theorem 4.3.** *Let τ denote the maximum number of pairwise distinct items in an instance of LS Vector Bin Packing. LS Vector Bin Packing can be solved in $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot (b + d + |\mathcal{S}|)^{\mathcal{O}(1)}$ time and in $k^{\mathcal{O}(\tau)} \cdot (b + d + |\mathcal{S}|)^{\mathcal{O}(1)}$ time.*

Proof. The proof relies on the algorithm behind Theorem 3.5. We describe how to use this algorithm to solve an instance

$$I := (b, d, \mathcal{S}, (\omega_i)_{i\in[1,b]}, \chi, k)$$

of LS Vector Bin Packing

1. GBP construction. Initially, we describe how I corresponds to an instance $J := (b, X, (\varphi_i)_{i\in[1,b]}, f, k)$ of LS Generalized Bin Problem. Our universe X is exactly the set $\mathcal{S}$ and the number b is the number of bins in I. It remains to define the IBE. For $i \in [1, b]$, we define $\varphi_i(X') := \mathsf{ol}(B_i, X')$. That is, the ith IBE corresponds to the overload of B_i with respect to the vectors X'. Our operation $\oplus$ is the sum of integer numbers.

2. Type Partition. Let $(X_1, \ldots, X_\tau)$ be a partition of $\mathcal{S}$ where each X_j is an inclusion-wise maximal set of pairwise identical elements. Note tat this partition can clearly be computed in $|\mathcal{S}|^{\mathcal{O}(1)}$ time.

To show that this collection is in fact a type partition, we show that two elements from one set X_j are target equivalent according to Definition 3.2. Let vec_1 and vec_2 be two vectors of the same set X_j. Let $S \subseteq \mathcal{S}$ be a set with $S \cap \{\mathsf{vec}_1, \mathsf{vec}_2\} = \{\mathsf{vec}_1\}$. Let $S' := (S \setminus \{\mathsf{vec}_1\}) \cup \{\mathsf{vec}_2\}$. We show that for each $i \in [1, b]$, $\varphi_i(S) = \varphi_i(S')$. Without loss of generality, we consider the IBE φ_i. Since C is a type class containing vec_1 and vec_2, we have $S \setminus \{\mathsf{vec}_1\} = S' \setminus \{\mathsf{vec}_2\}$. Consequently, $\varphi_i(S \cup \{\mathsf{vec}_2\}) = \varphi_i(S' \cup \{\mathsf{vec}_1\})$ and thus vec_1 and vec_2 are target equivalent.

3. Solution Correspondence. Since $\oplus$ is the sum of integer numbers and $\varphi_i(X') = \mathsf{ol}(B_i, X')$ for all $i \in [1, b]$, we have $\mathrm{val}(\chi) = \sum_{i\in b} \mathsf{ol}(B_i, \chi^{-1}(i))$ for every bin assignment χ. ◀

In Theorem 5.3 we complement the above algorithm by showing that LS Vector Bin Packing is W[1]-hard parameterized by k and cannot be solved in $f(k) \cdot n^{o(k)}$ time unless the ETH fails.

4.4 Vertex Deletion Distance to Specific Graph Properties

We next consider a general class of graph problems. Given a graph, one aims to delete a minimum number of vertices such that the remaining graph satisfies a specific property (for example: being bipartite). For any fixed graph-property Π that can be verified in polynomial time, we define the following problem.

Π Vertex Deletion
Input: A graph $G = (V, E)$.
Goal: Find a subset $S \subseteq V$ such that $G - S$ fulfills property Π and $\lvert S\rvert$ is minimal.

Parameterized local search was considered for Π Vertex Deletion for general graph properties Π [7] and in particular for the special case of Π being the family of all edgeless graphs, that is, for Vertex Cover [7, 13, 21, 24]. For the corresponding local search problems LS Π Vertex Deletion and LS Vertex Cover, the considered local neighborhood is the *k-swap neighborhood*, that is, the set of all solutions that have a symmetric difference with the current solution of size at most k. This neighborhood coincides with the k-flip neighborhood if a solution for Π Vertex Deletion and Vertex Cover is represented as a 2-coloring χ of the vertex set V, where $G[\chi^{-1}(1)]$ fulfills property Π and the goal is to minimize the number of vertices that receive color 2 under χ.

It was shown that LS Π Vertex Deletion is W[1]-hard when parameterized by k for all hereditary graph properties Π that contain all edgeless graphs but not all cliques or vice versa [7]. This holds in particular for the special case of LS Vertex Cover [7, 13, 24]. Furthermore, a closer inspection shows that LS Vertex Cover cannot be solved in $f(k) \cdot n^{o(k)}$ time unless the ETH fails [24, Theorem 3.7]. Since LS Vertex Cover is a special case of LS Π Vertex Deletion, we also obtain the same hardness results for the more general LS Π Vertex Deletion. From the positive side, LS Vertex Cover admits an FPT-algorithm for k, if the input graph has a bounded local treewidth. Moreover, algorithms for LS Vertex Cover are known that run in $f(k) \cdot \ell^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time, where ℓ is any of (i) the maximum degree Δ [21], (ii) the h-index of the input graph [24], or (iii) the treewidth of the input graph [24]. In particular, the performance of the algorithm combining k and Δ was shown to be very successful [21].

We show that our framework is applicable for an even more general problem LS Multi Component Π Deletion in which we aim to find a vertex set of minimal size to remove, so that the remaining vertices can be partitioned into c sets, that each fulfill property Π. Note that LS Π Vertex Deletion is the special case of $c = 1$.

▶ **Theorem 4.4** (⋆). *LS Multi Component Π Deletion can be solved in* $\mathrm{nd}^k \cdot 2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ *time and in* $k^{\mathcal{O}(\mathrm{nd})} \cdot n^{\mathcal{O}(1)}$ *time.*

4.5 Further Applications of the Framework

With similar applications the algorithm behind Theorem 3.5, we can obtain results for the local search versions of the well-known problems Nash Social Welfare and Multi Knapsack. Here, we only present the problem definitions and the respective results. References to the related literature and the proofs of our theorems for both problems are deferred to the full version.

Nash Social Welfare. Let n, m be integers. In this section, $\mathcal{A}$ is a set of n agents and $\mathcal{S}$ is a set of m items. Furthermore, we have n utility functions $(u_i : \mathcal{S} \to \mathbb{N})_{i\in[1,n]}$. An n-partition χ of $\mathcal{S}$ is called an *allocation*. For an allocation χ, the function $\mathrm{Nash}(\chi) := \prod_{i\in[1,n]} \left(\sum_{s\in\chi^{-1}(i)} u_i(s)\right)$ is called the *Nash score* of χ.

Nash Social Welfare
Input: A set $\mathcal{A}$ of agents, a set $\mathcal{S}$ of items, and a set $(u_i)_{i\in[1,n]}$ of utilities.
Goal: Find an allocation χ that maximizes $\mathrm{Nash}(\chi)$.

The input of the corresponding local search problem LS Nash Social Welfare additionally consists of an allocation χ and some $k \in \mathbb{N}$ and one asks for an allocation χ' with $\mathrm{Nash}(\chi') > \mathrm{Nash}(\chi)$ and $d_{\mathrm{flip}}(\chi, \chi') \leq k$.

We say that two items j_1 and j_2 are *identical* if each agents values j_1 and j_2 similarly, that is, if $u_i(j_1) = u_i(j_2)$ for each $i \in [n]$. Two items are *distinct* if they are not identical.

▶ **Theorem 4.5** (⋆). *Let τ denote the maximum number of pairwise distinct items in an instance of LS Nash Social Welfare. LS Nash Social Welfare can be solved in $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot (n+m)^{\mathcal{O}(1)}$ time and in $k^{\mathcal{O}(\tau)} \cdot (n+m)^{\mathcal{O}(1)}$ time.*

Multi Knapsack. Let $m \in \mathbb{N}$ be a number of knapsacks, with their capacities $W_1, \ldots, W_m \in \mathbb{N}$ and let $[1, n]$ be a set of items with values $v_{(\ell,i)} \in \mathbb{N}$ and weights $w_{(\ell,i)} \in \mathbb{N}$ for all $\ell \in [1, n]$ and $i \in [1, m]$. We say that an $(m+1)$-partition χ of $[1, n]$ *fits into the knapsacks*, if $\sum_{i \in \chi^{-1}(\ell)} w_{(\ell,i)} \leq W_i$ for every $i \in [1, m]$. We define the *score of* χ as $\text{score}(\chi) := \sum_{i=1}^{m} \sum_{\ell \in \chi^{-1}(i)} v_{(\ell,i)}$. Intuitively, for $i \in [1, m]$, the set $\chi^{-1}(i)$ corresponds to the chosen items for the ith knapsack and $\chi^{-1}(m+1)$ corresponds to the set of non-chosen items.

Multi Knapsack
Input: Knapsacks with capacities $W_1, \ldots, W_m \in \mathbb{N}$, items $[1, n]$ with values $v_{(\ell,i)} \in \mathbb{N}$ and weights $w_{(\ell,i)} \in \mathbb{N}$ for all $\ell \in [1, n]$ and $i \in [1, m]$.
Goal: Find an $(m+1)$-partition χ of $[1, n]$ that fits into the knapsacks, such that $\text{score}(\chi)$ is maximal.

The input of the corresponding local search problem LS Multi Knapsack additionally consists of an m-partition χ that fits into the knapsacks and some $k \in \mathbb{N}$ and one asks for an improving $(m+1)$-partition χ' with $d_{\text{flip}}(\chi, \chi') \leq k$ that fits into the knapsacks.

We say that two items $\ell \in [1, n]$ and $\ell' \in [1, n]$ are *identical* if $v_{(\ell,i)} = v_{(\ell',i)}$ and $w_{(\ell,i)} = w_{(\ell',i)}$ for all $i \in [1, m]$. Two elements are distinct if they are not identical.

To the best of our knowledge, parameterized local search for Multi Knapsack has not yet been studied. We now show the following.

▶ **Theorem 4.6** (⋆). *Let τ denote the maximum number of pairwise distinct items in an instance of LS Multi Knapsack. LS Multi Knapsack can be solved in $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot (n+m)^{\mathcal{O}(1)}$ time and in $k^{\mathcal{O}(\tau)} \cdot (n+m)^{\mathcal{O}(1)}$ time.*

5 Intractability Results for the Considered Local Search Problems

In this section, we present (parameterized) intractability results for the considered local search problems in this work and tight running-time lower bounds with respect to the $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$-time algorithms derived in Section 4. As already discussed in the previous sections, some of the considered problems in this work are known to be W[1]-hard when parameterized by k and cannot be solved in $f(k) \cdot |I|^{o(k)}$ time, unless the ETH fails. This is the case for LS Max c-Cut [30], LS Cluster Editing [11], LS Π Vertex Deletion [7, 13, 24]. In this section we thus only show that these intractability results also hold for the remaining local search problems considered in this work, that is, for LS Multi Knapsack, LS Nash Social Welfare, and LS Vector Bin Packing.

We start by presenting our (parameterized) intractability results and running-time lower bounds for the local search problems LS Multi Knapsack and LS Nash Social Welfare.

▶ **Theorem 5.1** (⋆). *LS Multi Knapsack is* W[1]*-hard with respect to k and cannot be solved in $n^{o(k)}$ time, unless the ETH fails, where n denotes the number of items in the input instance. This holds even if there is only a single knapsack.*

▶ **Theorem 5.2** (⋆). *LS Nash Social Welfare is* W[1]*-hard with respect to k and cannot be solved in $n^{o(k)}$ time, unless the ETH fails. This holds even if there are only two agents and both have the same evaluation for each item.*

Now, we provide matching hardness results for LS Vector Bin Packing. More precisely, we provide two hardness results even if each entry in each vector is only 0 or 1. First, we show that LS Vector Bin Packing is W[1]-hard with respect to k and that an algorithm with running time $f(k) \cdot |I|^{o(k)}$ violates the ETH. Consequently, our $\tau^k \cdot 2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ time algorithm presented in Theorem 4.3 is tight if the ETH is true. Second, we show W[1]-hardness for LS Vector Bin Packing parameterized by $k + q$. Here, $q := \max_{v \in \mathcal{S}} \|v\|_1$ is the maximal sum of entries over all vectors. Recall that q is usually smaller than τ in our real-world application.

▶ **Theorem 5.3.** *LS Vector Bin Packing is* W[1]*-hard with respect to k and cannot be solved in $f(k) \cdot |I|^{o(k)}$ time, unless the ETH fails. This holds even if $b = 2$, each entry in each vector is only 0 or 1, and each entry of the vector ω for each bin is 1.*

Proof. We present a simple reduction from LS Max c-Cut which provides the desired intractability results. Let $c \geq 2$ and let $I := (G = (V, E), \chi : V \to [1, c], k)$ be an instance of LS Max c-Cut where χ is locally optimal if and only if χ is globally optimal. Even under these restrictions, LS Max c-Cut is W[1]-hard with respect to k and cannot be solved in $f(k) \cdot n^{o(k)}$ time, unless the ETH fails [10]. Let $m := |E|$ and let $E := \{e_1, \ldots, e_m\}$. We obtain an equivalent instance $I' := (c, m, \mathcal{S}, (\omega_i)_{i \in [1,c]}, \psi, k')$ of LS Vector Bin Packing as follows: For each bin $j \in [1, c]$, we define the vector ω_j as the vector of length m that has a 1 in each dimension. For each vertex $v \in V$, we create a vector x_v that has a 1 at dimension $i \in [1, m]$ if and only if vertex v is incident with edge e_i. Let $\mathcal{S}$ be the set of these vectors and let ψ be the coloring obtained from χ by assigning for each vertex $v \in V$, color $\chi(v)$ to vector x_v, that is, $\chi(v) = \psi(x_v)$. Finally, we set $k' := k$.

For the sake of simplicity, we may identify each vector x_v by its corresponding vertex v. Similarly, we may consider c-partitions of V instead of c-partitions of $\mathcal{S}$ as solutions for I' based on the obvious bijection between vertices and vectors. Next, we show that I is a yes-instance of LS Max c-Cut if and only if I' is a yes-instance of LS Vector Bin Packing. To this end, we first analyze the objective functions of both instances with respect to corresponding solutions.

Let χ' be a c-coloring of V and let ψ' be the corresponding c-coloring of $\mathcal{S}$. Let $e_i := \{u, v\}$ be an edge of G having endpoints of distinct color under χ'. Then, no bin produces an overload in dimension i because only the vectors x_u and x_v have a 1 at dimension i and both vectors receive distinct colors under ψ'. Similarly, let $e_i := \{u, v\}$ be an edge of G having endpoints of the same color $\alpha \in [1, c]$ under χ'. Then, no bin except α produces an overload at dimension i and bin α produces an overhead of exactly 1 in dimension i, because only the vectors x_u and x_v have a 1 at dimension i and both vectors receive color α under ψ'. Consequently, the total overload of ψ' over all dimensions and all bins equals $|E|$ minus the number of properly colored edges of G under χ'. In other words, χ' is a better solution for I than χ if and only if ψ' is a better solution for I' than ψ. Since $d_{\text{flip}}(\chi, \chi') = d_{\text{flip}}(\psi, \psi')$, this implies that I is a yes-instance of LS Max c-Cut if and only if I' is a yes-instance of LS Vector Bin Packing. Furthermore, since χ is a locally optimal solution if and only if χ is a globally optimal solution, ψ is a locally optimal solution if and only if ψ is a globally optimal solution.

Recall that LS Max c-Cut is W[1]-hard with respect to k and cannot be solved in $f(k) \cdot n^{o(k)}$ time, unless the ETH fails [10]. Since $|\mathcal{S}| = n$, each vector has $m \leq n^2$ dimensions, and $k' = k$, this implies that LS Vector Bin Packing is W[1]-hard with respect to k' and cannot be solved in $f(k') \cdot |I'|^{o(k')}$ time, unless the ETH fails. ◀

Now, we present our second hardness result for the parameter k plus q, the maximal sum of entries over all vectors.

▶ **Theorem 5.4** ($\star$). *LS Vector Bin Packing is* W[1]*-hard with respect to* $k + q$*, even if each entry in each vector is only 0 or 1, and each entry of the vector* ω *for each bin is 1.*

6 Discussion

There are several ways of extending our work: In all of our applications of the framework we extensively used the expressive power of the IBEs and the flexibility of the bins. So far, we have not exploited the power of the types. For future work it is thus interesting to find examples where the number τ of types is smaller than the neighborhood diversity or the number of distinct vectors. Also, it is interesting to study the parameterized complexity of the non-local search versions of the considered problems parameterized by τ alone. For example, Bin Packing admits an FPT-algorithm when parameterized by τ alone [22]. It appears to be reasonable that the algorithm can be modified to obtain fixed-parameter tractability for Vector Bin Packing parameterized by τ as well. Furthermore, Vertex Cover admits an FPT-algorithm when parameterized by nd [25] and Max c-Cut admits an FPT-algorithm when parameterized by nd $+ c$ [14]. However, it is not known whether such algorithms are possible for all (classic) variants of our study; for example for Nash Social Welfare such an algorithm is not known.

References

1 Thomas Bläsius, Philipp Fischbeck, Lars Gottesbüren, Michael Hamann, Tobias Heuer, Jonas Spinner, Christopher Weyand, and Marcus Wilhelm. PACE Solver Description: KaPoCE: A Heuristic Cluster Editing Algorithm. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation (IPEC '21)*, volume 214 of *LIPIcs*, pages 31:1–31:4. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.IPEC.2021.31.

2 Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. NuMVC: An Efficient Local Search Algorithm for Minimum Vertex Cover. *Journal of Artificial Intelligence Research*, 46:687–716, 2013. doi:10.1613/JAIR.3907.

3 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.

4 Juan A. Díaz and Elena Fernández. A tabu search heuristic for the generalized assignment problem. *European Journal of Operational Research*, 132(1):22–38, 2001. doi:10.1016/S0377-2217(00)00108-9.

5 Martin Dörnfelder, Jiong Guo, Christian Komusiewicz, and Mathias Weller. On the parameterized complexity of consensus clustering. *Theoretical Computer Science*, 542:71–82, 2014. doi:10.1016/J.TCS.2014.05.002.

6 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

7 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, and Yngve Villanger. Local search: Is brute-force avoidable? *Journal of Computer and System Sciences*, 78(3):707–719, 2012. doi:10.1016/J.JCSS.2011.10.003.

8 Paola Festa, Panos M Pardalos, Mauricio GC Resende, and Celso C Ribeiro. Randomized heuristics for the max-cut problem. *Optimization methods and software*, 17(6):1033–1058, 2002. doi:10.1080/1055678021000090033.

9 Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On structural parameterizations of the bounded-degree vertex deletion problem. *Algorithmica*, 83(1):297–336, 2021. doi:10.1007/S00453-020-00758-8.

10 Jaroslav Garvardt, Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz. Parameterized Local Search for Max c-Cut. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI '23)*, pages 5586–5594. ijcai.org, 2023. doi:10.24963/IJCAI.2023/620.

11 Jaroslav Garvardt, Nils Morawietz, André Nichterlein, and Mathias Weller. Graph clustering problems under the lens of parameterized local search. In *Proceedings of the 18th International Symposium on Parameterized and Exact Computation (IPEC '23)*, volume 285 of *LIPIcs*, pages 20:1–20:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.IPEC.2023.20.

12 Serge Gaspers, Joachim Gudmundsson, Mitchell Jones, Julián Mestre, and Stefan Rümmele. Turbocharging treewidth heuristics. *Algorithmica*, 81(2):439–475, 2019. doi:10.1007/S00453-018-0499-1.

13 Serge Gaspers, Eun Jung Kim, Sebastian Ordyniak, Saket Saurabh, and Stefan Szeider. Don't be strict in local search! In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI '12)*. AAAI Press, 2012. doi:10.1609/AAAI.V26I1.8128.

14 Tomas Gavenciak, Martin Koutecký, and Dusan Knop. Integer programming in parameterized complexity: Five miniatures. *Discrete Optimization*, 44(Part):100596, 2022. doi:10.1016/J.DISOPT.2020.100596.

15 Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz. Efficient Bayesian network structure learning via parameterized local search on topological orderings. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 12328–12335. AAAI Press, 2021. Full version available at https://doi.org/10.48550/arXiv.2204.02902. doi:10.1609/AAAI.V35I14.17463.

16 Jiong Guo, Sepp Hartung, Rolf Niedermeier, and Ondrej Suchý. The parameterized complexity of local search for TSP, more refined. *Algorithmica*, 67(1):89–110, 2013. doi:10.1007/S00453-012-9685-8.

17 Jiong Guo, Danny Hermelin, and Christian Komusiewicz. Local search for string problems: Brute-force is essentially optimal. *Theoretical Computer Science*, 525:30–41, 2014. doi:10.1016/J.TCS.2013.05.006.

18 Sepp Hartung and Rolf Niedermeier. Incremental list coloring of graphs, parameterized by conservation. *Theoretical Computer Science*, 494:86–98, 2013. doi:10.1016/J.TCS.2012.12.049.

19 Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004.

20 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.

21 Maximilian Katzmann and Christian Komusiewicz. Systematic exploration of larger local search neighborhoods for the minimum vertex cover problem. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI '17)*, pages 846–852. AAAI Press, 2017. doi:10.1609/AAAI.V31I1.10659.

22 Dusan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. Parameterized complexity of configuration integer programs. *Operations Research Letters*, 49(6):908–913, 2021. doi:10.1016/J.ORL.2021.11.005.

23 Christian Komusiewicz, Simone Linz, Nils Morawietz, and Jannik Schestag. On the complexity of parameterized local search for the maximum parsimony problem. In *Proceedings of the 34th Annual Symposium on Combinatorial Pattern Matching (CPM '23)*, volume 259 of *LIPIcs*, pages 18:1–18:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CPM.2023.18.

24 Christian Komusiewicz and Nils Morawietz. Parameterized Local Search for Vertex Cover: When Only the Search Radius Is Crucial. In *Proceedings of the 17th International Symposium on Parameterized and Exact Computation (IPEC '22)*, volume 249 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.IPEC.2022.20.

25 Martin Kouteckỳ. Solving hard problems on neighborhood diversity. Master's thesis, Charles University in Prague, 2013. URL: https://koutecky.name/mgr/mgr.pdf.

26 Ruizhi Li, Shuli Hu, Shaowei Cai, Jian Gao, Yiyuan Wang, and Minghao Yin. NuMWVC: A novel local search for minimum weighted vertex cover problem. *Journal of the Operational Research Society*, 71(9):1498–1509, 2020. doi:10.1080/01605682.2019.1621218.

27 Andrea Lodi, Silvano Martello, and Daniele Vigo. Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research*, 112(1):158–166, 1999. doi:10.1016/S0377-2217(97)00388-3.

28 Christiane Markuse-Schneider. Personal communication. Schmitz Cargobull production site in Vreden, Germany, 2023.

29 Dániel Marx. Searching the k-change neighborhood for TSP is W[1]-hard. *Oper. Res. Lett.*, 36(1):31–36, 2008. doi:10.1016/J.ORL.2007.02.008.

30 Nils Morawietz. *On the complexity of local search problems with scalable neighborhoods*. PhD thesis, Friedrich-Schiller-Universität Jena, 2024. Dissertation. URL: https://www.db-thueringen.de/receive/dbt_mods_00064137.

31 Quan Ouyang and Hong Yun Xu. Genetic algorithm for single machine scheduling problem with setup times. *Applied Mechanics and Materials*, 457:1678–1681, 2014.

32 Stefan Szeider. The parameterized complexity of k-flip local search for SAT and MAX SAT. *Discrete Optimization*, 8(1):139–145, 2011. doi:10.1016/J.DISOPT.2010.07.003.

Spanner for the $0/1/\infty$ Weighted Region Problem

Joachim Gudmundsson
The University of Sydney, Australia

Zijin Huang
The University of Sydney, Australia

André van Renssen
The University of Sydney, Australia

Sampson Wong
University of Copenhagen, Denmark

Abstract

We consider the problem of computing an approximate weighted shortest path in a weighted planar subdivision, with weights assigned from the set $\{0, 1, \infty\}$. The subdivision includes zero-cost regions (0-regions) with weight 0 and obstacles with weight ∞, all embedded in a plane with weight 1. In a polygonal domain, where the 0-regions and obstacles are non-overlapping polygons (not necessarily convex) with in total N vertices, we present an algorithm that computes a $(1+\varepsilon)$-approximate spanner of the input vertices in expected $\widetilde{O}(N/\varepsilon^3)$ time[1], for $0 < \varepsilon < 1$. Using our spanner, we can compute a $(1+\varepsilon)$-approximate weighted shortest path between any two points (not necessarily vertices) in $\widetilde{O}(N/\varepsilon^3)$ time. Furthermore, we prove that our results more generally apply to non-polygonal convex regions. Using this generalisation, one can approximate the weak partial Fréchet similarity [7] between two polygonal curves in expected $\widetilde{O}(n^2/\varepsilon^2)$ time, where n is the total number of vertices of the input curves.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases weighted region problem, approximate shortest path, spanner

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.33

Related Version *Full Version*: `https://arxiv.org/abs/2407.01951`

Funding *Joachim Gudmundsson*: This research was partially funded by the Australian Government through the Australian Research Council (project number DP240101353).
André van Renssen: This research was partially funded by the Australian Government through the Australian Research Council (project number DP240101353).

1 Introduction

The Weighted Region Problem (WRP) is a generalization of the shortest path problem, considering a planar subdivision E where each face has a non-negative weight associated with it. A path σ in E can be partitioned into a set of subpaths $\{\sigma_1, ..., \sigma_k\}$ based on its intersection with faces in E, where a subpath σ_i starts at the point s_i and ends at the point t_i. Both s_i and t_i must lie on the boundary of the same face F_i. The weight of the subpath σ_i is the Euclidean length of σ_i times the weight assigned to F_i. The total weight of a path is the sum of the weights of its subpaths. The goal of the WRP is to find the weighted shortest path from a source point s to a target point t. When the weights are in the set $\{0, 1, \infty\}$, this problem is referred to as the $0/1/\infty$ Weighted Region Problem [15].

[1] $\widetilde{O}(\cdot)$ notation ignores poly-logarithmic dependencies on N and $1/\varepsilon$.

© Joachim Gudmundsson, Zijin Huang, André van Renssen, and Sampson Wong;
licensed under Creative Commons License CC-BY 4.0

Researchers have conjectured that the WRP is difficult to solve [15], and recent studies confirm this conjecture – the Weighted Region Problem is unsolvable in the algebraic computation model over the rational numbers. De Carufel et al. [14] demonstrated that the WRP cannot be solved exactly even with only three different weights. De Berg et al. [12] confirmed its unsolvability with just two different weights. Mitchell and Papadimitriou [17] illustrated that in two dimensions, a weighted shortest path can intersect at least $\Omega(n^2)$ boundaries even when the regions are convex.

Due to the difficulty of solving the WRP exactly, approximation algorithms have been considered. A common approach is to discretize the problem space, either by assuming the space is a tessellation of convex polygons with exactly one associated weight [5], or by placing Steiner (sample) points on the boundaries of the regions [1, 2, 9, 16, 18]. In these approaches, the number of sample points depends not only on the complexity of the regions but also on geometric parameters such as the maximum integer coordinate of any vertex and the ratio r_w of the maximum weight over the minimum weight. As r_w increases, so does the number of required sample points. As a result, the weights are required to be strictly positive.

1.1 Related work

Our work is closely related to the data structure and algorithm by Gewali et al. [15] to solve the $0/1/\infty$ weighted region problem. Their algorithm takes a polygonal domain with N vertices as input and constructs a *critical graph* $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*)$ (a type of visibility graph) with $O(N^2)$ edges. Dijkstra's shortest path algorithm can be used on $\mathcal{G}^*$ to compute a weighted shortest path between any pair of vertices in $O(N^2)$ time.

In $0/1/\infty$ weighted regions, a weighted shortest path P^* avoids obstacles and traverses the 0-regions freely, while minimizing its length in the plane (1-region). Consider two (closed) regions A and B, each either a 0-region or an obstacle. The key observation in [15] is that an edge in P^* connecting A and B must be *locally optimal* (see Fact 16). For example, an edge (a, b) connecting two convex 0-regions A and B must be perpendicular to the tangent touching $a \in A$ and the tangent touching $b \in B$. Gewali et al. [15] showed that G^* contains all such locally optimal edges in G^*, which implies that G^* must contain the optimal path between any pair of vertices in G^*.

1.2 Our Contribution

In this paper, we build on the work by Gewali et al. [15] with a focus on the 0-regions as they are not handled well by existing approximation schemes (using sample points or tessellation). In Section 2, we consider the 0/1 weighted region problem where the 0-regions are convex but not necessarily polygonal.

▶ **Problem 1.** *In the planar subdivision induced by the plane with weight* 1 *and a set* $\mathcal{Z}$ *of non-overlapping convex zero-cost regions (0-regions) with weight* 0*, given an approximation error* $0 < \varepsilon < 1$*, find a* $(1+\varepsilon)$*-approximate weighted shortest path from an arbitrary point* s *to an arbitrary point* t*.*

The high-level idea is that, in order to obtain $(1+\varepsilon)$-approximate shortest paths, we place $O(1/\varepsilon)$ sample points on the boundary of each 0-region; the number of sample points is independent of other parameters. Using these sample points, we construct a Θ-graph and $O(1/\varepsilon)$ trapezoidal maps, which are part of our data structure $\mathcal{B}$. The trapezoidal maps ensure the existence of good paths between 0-regions that are close[2] to each other, while

[2] A precise definition is provided in Lemma 8 and 10, Section 2.

the Θ-graph ensures the same for 0-regions that are far from each other. To the best of our knowledge, our algorithm is the first near-linear time $(1+\varepsilon)$-approximation algorithm that finds an approximated weighted shortest path in a 0/1 weighted region. Note that both Theorem 1 and 3 apply to polygonal domains with non-convex regions, since they can be triangulated into a linear number of triangles.

▶ **Theorem 1.** *Consider a planar subdivision induced by a plane with weight 1, containing a set $\mathcal{Z}$ of non-overlapping convex 0-regions with weight 0. Let $|\mathcal{Z}| = n$ and N denote the total number of vertices in $\mathcal{Z}$. For any approximation factor $0 < \varepsilon < 1$, a data structure $\mathcal{B}$ can be constructed over $\mathcal{Z}$ in $O(N + (n/\varepsilon^2)(\log(n/\varepsilon) + \log N))$ expected time, with a total size of $O(N + n/\varepsilon^2)$. When queried with points s and t, $\mathcal{B}$ can return a weighted path P from s to t in $O(N + n/\varepsilon^2 + (n/\varepsilon)\log(n/\varepsilon) + (\log N)/\varepsilon)$ time, satisfying $\mathbf{w}(P) \leq (1+\varepsilon) \cdot \mathbf{w}(P^*)$, where P^* is the optimal weighted shortest path from s to t.*

To use our algorithm on an application, we prove the above theorem in a more general setting, where the 0-regions are non-polygonal. In Section 3, we use our algorithm to approximate the partial weak Fréchet similarity of two polygonal curves. This problem was first studied by Buchin et al. [7], and they presented a cubic time algorithm. De Carufel et al. [13] later transformed the problem into a weighted shortest path problem amidst 0/1-regions. Using Theorem 1, our algorithm is the first near-quadratic time $(\sqrt{2}+\varepsilon)$-approximation algorithm for computing the partial weak Fréchet similarity between a pair of polygonal curves.

Buchin et al. [8] showed that there is no strongly subquadratic time algorithm for approximating the weak Fréchet distance within a factor less than 3 unless the strong exponential-time hypothesis fails. Approximating the partial weak Fréchet similarity is at least as hard as approximating the weak Fréchet distance. As a result, it is unlikely that a subquadratic time algorithm exists.

▶ **Theorem 2.** *One can approximate the partial weak Fréchet similarity of two curves within a factor of $(\sqrt{2}+\varepsilon)$ in $O((n^2/\varepsilon^2)\log(n/\varepsilon))$ expected time.*

In Section 4, we generalise our data structure to also allow convex obstacles that cannot be traversed, i.e., obstacles of weight ∞. Let $\mathbf{d}(a,b)$ denote the weight of the weighted shortest path from a to b amidst $0/1/\infty$-weighted region. By introducing additional sample points, we show that if we need to take a detour from a sample point a to a sample point b, there exists a set D (a detour) of edges in $\mathcal{B}$ such that the total length of D approximates the distance $\mathbf{d}(a,b)$ within a factor of $1+\varepsilon$.

In the special case that the 0-regions and obstacles are polygonal, $\mathcal{B}$ is a $(1+\varepsilon)$-spanner of the input vertices. To the best of our knowledge, our algorithm is the first near-linear time $(1+\varepsilon)$-approximation algorithm for the weighted shortest path in a $0/1/\infty$ weighted region.

▶ **Theorem 3.** *Consider a planar subdivision induced by a plane with a weight of 1, consisting of two sets of convex and non-overlapping regions: 0-regions $\mathcal{Z}$ with a weight of 0, and obstacles $\mathcal{O}$ with a weight of ∞. Let $n = |\mathcal{Z}| + |\mathcal{O}|$ and let N denote be the total number of vertices in $\mathcal{Z} \cup \mathcal{O}$. For any approximation factor $0 < \varepsilon < 1$, a data structure $\mathcal{B}$ can be constructed over $\mathcal{Z} \cup \mathcal{O}$ in $O(N + (n/\varepsilon^3)(\log(n/\varepsilon) + \log N))$ expected time, with a total size of $O(N + n/\varepsilon^3)$. When queried with arbitrary points s and t, $\mathcal{B}$ returns a path P from s to t in $O(N + n/\varepsilon^3 + (n/\varepsilon^2)\log(n/\varepsilon) + (\log N)/\varepsilon)$ time, ensuring that $\mathbf{w}(P) \leq (1+\varepsilon) \cdot \mathbf{w}(P^*)$, where P^* is the optimal weighted shortest path from s to t.*

2 Shortest path amidst 0-regions

The exact version of Problem 1 has a brute-force $\Omega(n^2)$ algorithm, by computing the distance between every pair of 0-regions. To compute an approximate solution, the goal is to construct an undirected weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with a near-linear number of edges, such that there exists a path P between two 0-regions in $\mathcal{G}$ with $\mathbf{w}(P) \leq (1+\varepsilon) \cdot \mathbf{w}(P^*)$, for a fixed parameter $\varepsilon > 0$, where P^* is the weighted shortest path.

To this end, we will use two data structures: trapezoidal maps and Θ-graphs. Both data structures are used to determine which pairs of 0-regions are connected. The trapezoidal maps will ensure that there exist good paths between 0-regions that are close to each other, while the Θ-graph ensures the same for 0-regions that are far from each other.

2.1 Construction of the data structure

In order to define our data structure, we first define a set of directions. Let $\theta < \pi/6$ be a fixed positive real number. Let $r(k\theta)$ be the direction with a counter-clockwise angle of $k\theta$ with the positive x-axis. Let $r(p, k\theta)$ be the ray originating from the point p with a counter-clockwise angle of $k\theta$ with the positive x-axis. To simplify the discussion, we will assume that $(\pi/2)/\theta \in \mathbb{Z}$ to guarantee that if $r(k\theta)$ exists, then so does $r(k\theta + \pi/2)$.

For a 0-region A, we define a set $\mathcal{SP}(A)$ of sample points on the boundary of A. Let $sp(A, k\theta)$ be a sample point on the boundary ∂A of A such that $sp(A, k\theta)$ is extreme in the direction $r(k\theta)$ (see Figure 1). When the geometric region A is clear from context, we write $sp(k\theta)$ instead of $sp(A, k\theta)$.

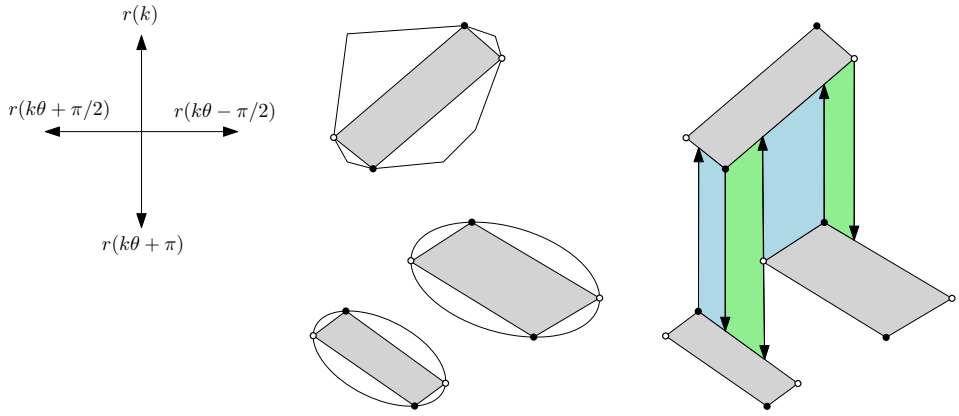


Figure 1 The sample points that are extreme in the directions of $r(k\theta)$ and $r(k\theta + \pi)$ are marked with black dots. The sample points that are extreme in the directions of $r(k\theta \pm \pi/2)$ are marked with circles. Using these sample points, we generate a simplified polygon and construct $\mathcal{M}(k)$; the blue and green regions are examples of faces in $\mathcal{M}(k)$.

Let $\partial A(a, a')$ define the subset of ∂A traversed from point a to point a' in counter-clockwise order, where $a, a' \in \partial A$. We say a line l *overlaps* A if l and A intersect at more than one point. We say two regions, A and B, are non-overlapping if their interiors do not intersect.

A point $p \in \partial A$ can be an extreme point for more than one direction, in which case we call p a vertex of A. There may be more than one extreme point on A for a single direction. If p is the extreme point on A for consecutive directions $\{r(k\theta), r((k+1)\theta), ..., r((k+m)\theta)\}$, we say $p = sp(k\theta)$, $p = sp((k+1)\theta)$, ..., and $p = sp((k+m)\theta)$ simultaneously. If there is

more than one extreme point for a single direction $r(k\theta)$, these extreme points must lie on some segment $ab \subseteq \partial A$, and we say both a and b are $sp(k\theta)$. The sample points on the boundary of a convex region can be computed by traversing the boundary.

▶ **Observation 4.** *Given n convex regions with N vertices in total, there are $O(n/\theta)$ sample points, and it takes $O(N + n/\theta)$ time to compute them.*

Using the sample points on a 0-region A, we can generate a simplified 0-region (a convex polygon) $\texttt{simpl}(A)$ by connecting adjacent sample points of every 0-region (see Figure 1). Using the set $\texttt{simpl}(\mathcal{Z})$ of simplified 0-regions, we will generate a set of trapezoidal maps, and we say $\texttt{simpl}(A)$ and $\texttt{simpl}(B)$ are adjacent if they are both adjacent to the same face in a trapezoidal map. We construct the query data structure $\mathcal{B}$ using Algorithm 1. We analyse $\mathcal{B}$ in the full version. Let $\|AB\|$ denote the Euclidean distance between two geometric objects A and B.

Algorithm 1 Construct $\mathcal{B}$ with 0-regions.

This algorithm takes as input a set $\mathcal{Z}$ of non-overlapping and convex 0-regions, and constructs a data structure $\mathcal{B}$. The undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is initially empty.

1. Compute the sample points $\mathcal{SP}(\mathcal{Z})$, and add $\mathcal{SP}(\mathcal{Z})$ to $\mathcal{V}$.
2. Pick an arbitrary sample point as the anchor $ak(A)$ for every $A \in \mathcal{Z}$. For each $p \in \mathcal{SP}(A)$, add $e = (p, ak(A))$ to $\mathcal{E}$, and set $\mathbf{w}(e) = 0$.
3. For each direction $r(k\theta)$, generate a trapezoidal map $\mathcal{M}(k)$ using $\texttt{simpl}(\mathcal{Z})$, and do the following for each $\mathcal{M}(k)$ (see [11, Theorem 6.3 and 6.8] for trapezoidal map construction).
   - For each face $F \in \mathcal{M}(k)$ adjacent to A and B, $A \neq B$, add $e = (ak(A), ak(B))$ to $\mathcal{E}$, and set $\mathbf{w}(e) = \|AB\|$.
4. With $\mathcal{V}$ as the input, generate a Θ-graph $\mathcal{G}_\Theta = (\mathcal{V}, \mathcal{E}_\Theta)$.
   - For each edge $(p, q) \in \mathcal{E}_\Theta$, if p and q do not belong to the same 0-region, add $e = (p, q)$ to $\mathcal{E}$, and set $\mathbf{w}(e) = \|pq\|$.
5. Return $\mathcal{B} = \{\mathcal{M}(k) \mid \forall k \in [0, 2\pi/\theta), k \in \mathbb{Z}\} \cup \{\mathcal{G}, \mathcal{G}_\Theta\}$ as the data structure.

▶ **Lemma 5.** *Given an approximation factor $0 < \varepsilon < 1$, and n non-overlapping convex 0-regions with total complexity N, one can build the data structure $\mathcal{B}$ in $O(N+(n/\varepsilon^2)(\log(n/\varepsilon)+\log N))$ time, and the total size of $\mathcal{B}$ is $O(N + n/\varepsilon^2)$.*

2.2 Trapezoidal map

Before arguing that there exists a good path using the edges constructed, we start with an observation about the pair of points realising the shortest distance between two 0-regions. For a convex region A and a point $p \in \partial A$, there exists at least one *supporting line* $l = l_t(A, p)$ going through p such that A lies entirely in one of the two halfplanes determined by l [19]. Let $p \in \partial A$, and $q \in \partial B$. We can observe that if pq realises $\|AB\|$, then pq must be perpendicular to a pair of supporting lines $l_t(A, p)$ and $l_t(B, q)$.

▶ **Observation 6.** *Let A and B be two convex regions. Let pq be the line segment realising $\|AB\|$, where $p \in A$ and $q \in B$. The segment pq must be perpendicular to a pair of supporting lines $l_t(A, p)$ and $l_t(B, q)$.*

We will show that if pq realises the distance between two 0-regions, we can transform pq into another segment pq' such that pq' is parallel to some direction $r(k\theta)$, and $\|pq'\|$ approximates $\|pq\|$. To do this, we first need a fact (see the full version for a proof).

▶ **Lemma 7.** *Given a segment pq, let α (resp. β) be the acute angle between pq and $r(p,(k+1)\theta)$ (resp. $r(p,k\theta)$), where $\alpha+\beta=\theta<\pi/6$. Let q' be the intersection of $r(p,(k+1)\theta)$ and $r(q,k\theta+\pi/2)$. We have that $\|pq'\|=(\cos(\beta)/\cos(\theta))\cdot\|pq\|$, and $\|qq'\|=(\sin(\alpha)/\cos(\theta))\cdot\|pq\|$.*

Let pq realise $\|AB\|$. We consider the scenario when $\|pq\|$ is relatively small compared to the horizontal span of (say) B. Using the above lemma, we will show that we have constructed a set of edges in $\mathcal{E}$ connecting two sample points $a\in A$ and $b\in B$, such that the total weight of these edges approximates $\|pq\|$.

▶ **Lemma 8.** *Let pq realise $\mathbf{d}(A,B)$, where $p\in A$ and $q\in B$. Let $p\in\partial A(a,a')$, and $q\in\partial B(b',b)$, where points a and a' (resp. b and b') are adjacent sample points on 0-region A (resp. B). If $\max\{\|pa'\|,\|qb\|\}\geq(\sin(\alpha)/\cos(\theta))\cdot\|pq\|$ or $\max\{\|pa\|,\|qb'\|\}\geq(\sin(\beta)/\cos(\theta))\cdot\|pq\|$, then there exists a path $P\subseteq\mathcal{E}$ from A to B such that $\mathbf{w}(P)\leq(\cos(\beta)/\cos(\theta))\cdot\|pq\|$.*

Proof. Without loss of generality, assume that $\|qb\|\geq(\sin(\alpha)/\cos(\theta))\cdot\|pq\|$. Lemma 7 implies that there exists a point $q'\in\partial B(b,q)$ such that pq' is in some direction $r(k\theta)$, and $\|pq'\|\leq(\cos(\beta)/\cos(\theta))\cdot\|pq\|$. Because

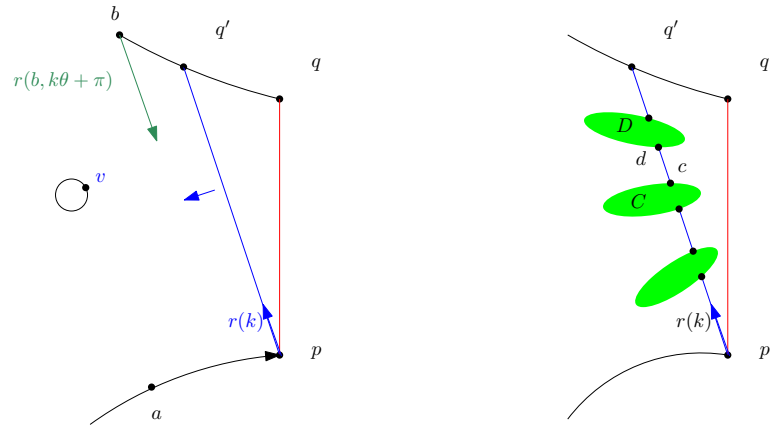


Figure 2 In the left figure, if pq' does not overlap a 0-region, we slide pq' until it touches a sample point. In the right figure, we slide each inter-region segment (cd as an example) the same way.

Observe that pq cannot overlap any 0-region; otherwise, pq does not realise $\mathbf{d}(A,B)$. If pq' does not overlap any 0-region (see Figure 2, left), we fix the orientation of pq', and move p along $\partial A(p,a)$ and q along $\partial B(b,q)$, until pq' touches a sample point.

If pq' touches a (resp. b), then $r(a,k)$ (resp. $r(b,k\theta+\pi)$) hits B (resp. A). If pq' touches a sample point $v\notin A\cup B$, v must be extreme in the direction $r(k\theta-\pi/2)$. As a result, $r(v,k)$ hits B, and $r(v,k\theta+\pi)$ hits A. In either case, A and B are adjacent in some face of $\mathcal{M}(k)$, and the edge $e=(ak(A),ak(B))$ is in $\mathcal{E}$ by construction. As a result, $\mathbf{w}(e)=\|AB\|=\|pq\|$. This is also trivially true when p or q' is already a sample point.

Otherwise, the segment pq' overlaps a set E' of 0-regions, and there exists a path P from A to B through E' (see Figure 2, right). Since the 0-regions do not overlap, the boundaries of the 0-regions in E' partition pq' into a set of intra-region and inter-region segments. Let cd be one among the set S of inter-region segments, where c is on a 0-region C, and d is on a 0-region D. Using the same argument as above, one can slide cd until it touches a sample point, and edge $(ak(C),ak(D))\in\mathcal{E}$ exists by construction.

In total, traveling from A to B via the 0-regions E' must be less costly than $\|pq'\|$, since $\mathbf{w}(ak(C), ak(D)) = \|CD\| \leq \|cd\|$, and the intra-region segments have weight 0. Summing up the cost of P, we have that

$$\mathbf{w}(P) = \sum_{cd \in S} \mathbf{w}(ak(C), ak(D)) < \sum_{cd \in S} \|cd\| < \|pq'\| = \frac{\cos(\beta)}{\cos(\theta)} \cdot \|pq\|. \quad \blacktriangleleft$$

2.3 Θ-Graph

In Step 4 of Algorithm 1, we constructed a Θ-graph $\mathcal{G}_\Theta = (\mathcal{V}_\Theta, \mathcal{E}_\Theta)$ using the defined sample points. The vertices $\mathcal{V}_\Theta$ are simply all sample points. The edges in $\mathcal{E}_\Theta$ are constructed using the standard Θ-graph construction [11]. Recall that in Algorithm 1, for every edge $(p, q) \in \mathcal{E}_\Theta$, with $p \in A$, $q \in B$, and $A \neq B$, we add an edge (p, q) to $\mathcal{E}$, and set $\mathbf{w}(p, q) = \|pq\|$.

The Θ-graph constructs a set of "good" edges in $\mathcal{E}$ when the distances between p (resp. q) and its adjacent sample points are small compared to $\|pq\|$. In this case, we argue that there exists a pair of sample points $a \in A$ and $b \in B$, such that $\|ab\| \leq (1/\cos(\theta)) \cdot \|pq\|$. Similar to Lemma 7, we prove the following geometric property in the full version.

▶ **Lemma 9.** *Given a segment pq, let α (resp. β) be the acute angle between pq and $r(p, (k+1)\theta)$ (resp. $r(p, k\theta)$), where $\alpha+\beta = \theta < \pi/6$. Let q' be the intersection of $r(p, (k+1)\theta)$ and $r(q, k\theta+\pi/2)$, and let p' be the intersection of $r(q, k\theta+\pi)$ and $r(p, (k+1)\theta+\pi/2)$. Let c be the intersection of pq' and qp'. We have that $\|cp'\| + \|cq'\| = (\sin(\alpha)+\sin(\beta))/(\cos(\theta)\sin(\theta))$, and $\|p'q'\| = (1/\cos(\theta)) \cdot \|pq\|$.*

In the case that both p and q are close to their adjacent sample points, we prove that there exists a pair (a, b) of sample points such that $\|ab\|$ approximates $\|pq\|$ in the full version.

▶ **Lemma 10.** *Let pq realise $\mathbf{d}(A, B)$, where $p \in A$ and $q \in B$. Let $p \in \partial A(a', a)$, and $q \in \partial B(b, b')$, where points a and a' (resp. b and b') are adjacent sample points on A (resp. B). If $\max\{\|pa'\|, \|qb\|\} < (\sin(\alpha)/\cos(\theta)) \cdot \|pq\|$ and $\max\{\|pa\|, \|qb'\|\} < (\sin(\beta)/\cos(\theta)) \cdot \|pq\|$, then $\mathbf{d}(a, b) < (1/\cos(\theta)) \cdot \|pq\|$.*

2.4 The quality of the path

For now, assume that s and t lie in some 0-region. An optimal s-t path P^* consists of a set of segments, where the endpoints of each segment lie on the boundaries of the 0-regions. A segment pq either lies within a 0-region or connects two different 0-region. Since it costs nothing to follow an edge inside a 0-region, the weight of P is the total weight of those edges connecting different 0-regions.

Let pq realise the distance between 0-regions A and B, where p lies on ∂A between sample points a and a', and q lies on ∂B between sample points b and b'. In Lemma 8, we have shown that if $\max\{\|pa'\|, \|qb\|\} \geq (\sin(\alpha)/\cos(\theta)) \cdot \|pq\|$ or $\max\{\|pa\|, \|qb'\|\} \geq (\sin(\beta)/\cos(\theta)) \cdot \|pq\|$, there exists a path $P \subseteq \mathcal{E}$ from a sample point on A to a sample point on B of length at most $(\cos(\beta)/\cos(\theta)) \cdot \|pq\|$.

In Lemma 10, we have shown that if $\max\{\|pa'\|, \|qb\|\} < (\sin(\alpha)/\cos(\theta)) \cdot \|pq\|$ and $\max\{\|pa\|, \|qb'\|\} < (\sin(\beta)/\cos(\theta)) \cdot \|pq\|$, there exist sample points $a \in A$ and $b \in B$, such that $\mathbf{d}(a, b) < (1/\cos(\theta)) \cdot \|pq\|$. To obtain a path between a and b in this case, we rely on the Θ-graph. The tightest bounds on the length of this path are due to Bose et al. [4], who showed that the spanning ratio of a Θ-graph is at most $r_\theta = 1 + 2\sin(\theta/2)/(\cos(\theta/2) - \sin(\theta/2))$. Our final approximation ratio is $(1/\cos(\theta)) \cdot r_\theta$. Given an input approximation factor $1+\varepsilon$, we can compute a desired angle $\theta \in O(\varepsilon)$. The undirected graph $\mathcal{G} \in \mathcal{B}$ is therefore a $(1+\varepsilon)$-spanner of the sample points. See the full version for details.

▶ **Lemma 11.** *The graph $\mathcal{G} \in \mathcal{B}$ contains a path $P \subseteq \mathcal{E}$ from sample point a to sample point b such that $\mathbf{w}(P) \leq (1+\varepsilon) \cdot \mathbf{w}(P^*)$, where P^* is the optimal path from a to b.*

Given a pair of query points s and t, we add s and t to $\mathcal{B}$. We treat both s and t as 0-regions with no interior to enable previous lemmas, yielding the theorem below. See the full version for details.

▶ **Theorem 1.** *Consider a planar subdivision induced by a plane with weight 1, containing a set $\mathcal{Z}$ of non-overlapping convex 0-regions with weight 0. Let $|\mathcal{Z}| = n$ and N denote the total number of vertices in $\mathcal{Z}$. For any approximation factor $0 < \varepsilon < 1$, a data structure $\mathcal{B}$ can be constructed over $\mathcal{Z}$ in $O(N + (n/\varepsilon^2)(\log(n/\varepsilon) + \log N))$ expected time, with a total size of $O(N + n/\varepsilon^2)$. When queried with points s and t, $\mathcal{B}$ can return a weighted path P from s to t in $O(N + n/\varepsilon^2 + (n/\varepsilon)\log(n/\varepsilon) + (\log N)/\varepsilon)$ time, satisfying $\mathbf{w}(P) \leq (1+\varepsilon) \cdot \mathbf{w}(P^*)$, where P^* is the optimal weighted shortest path from s to t.*

3 Partial weak Fréchet similarity

This section highlights one application of our data structure: approximating the partial weak Fréchet similarity. The Fréchet distance is a popular measure of the similarity between two polygonal curves. An *orientation-preserving reparameterisation* is a continuous and bijective function $f : [0,1] \to [0,1]$ such that $f(0) = 0$, and $f(1) = 1$. The $\mathbf{width}_{f,g}(\pi, \sigma)$ between two curves π and σ with respect to the reparameterisations f and g, is defined as follows.

$$\mathbf{width}_{f,g}(\pi, \sigma) = \max_{t \in [0,1]} \|\pi(f(t)) - \sigma(g(t))\|$$

Consider the scenario where a person is walking his dog with a leash connecting them: the person needs to stay on π while walking according to f, and the dog needs to stay on σ while walking according to g. The maximum leash length is the width between π and σ with respect to the reparameterisations f and g. The standard Fréchet distance $\delta_F(\pi, \sigma)$ is the minimum leash length required over all possible walks (defined by reparameterisations f and g).

$$\delta_F(\pi, \sigma) = \inf_{f,g \in [0,1] \to [0,1]} \mathbf{width}_{f,g}(\pi, \sigma)$$

Problems relating to the Fréchet distance are commonly solved in a configuration space called the *freespace diagram*. The *free space* $\mathcal{F}_d(\pi, \sigma)$ with respect to the Fréchet distance d is the union of all pairs of points $x \in \pi$ and $y \in \sigma$ such that the distance between x and y is at most d. As opposed to the free space, we will call $[0, \|\pi\|] \times [0, \|\sigma\|] \setminus \mathcal{F}_d(\pi, \sigma)$ the *forbidden space*.

$$\mathcal{F}_d(\pi, \sigma) = \{(x, y) \in [0, \|\pi\|] \times [0, \|\sigma\|] \mid \|\pi(x) - \sigma(y)\| \leq d\}$$

The *freespace diagram* $\mathcal{D}_d(\pi, \sigma)$ is a data structure that stores the free space $\mathcal{F}_d(\pi, \sigma)$ in n^2 cells. Alt and Godau [3] showed that the intersection of the free space with each cell is the intersection of an ellipse and a rectangle. Therefore the free space in each cell is convex, and its boundary is of constant complexity. For two polygonal curves with complexity n, Alt and Godau [3] proved the following fact.

▶ **Fact 12.** *The freespace diagram contains at most n^2 cells, and it can be constructed in $O(n^2 \log n)$ time. The free space inside each cell is the intersection of an ellipse and a rectangle.*

It is well-known that if one can find an xy-monotone path in $\mathcal{D}_d(\pi, \sigma)$ from the bottom-left corner s to the top-right corner t via the free space, then $\delta_F(\pi, \sigma) \leq d$.

The notion of weak Fréchet distance relaxes the requirement of the reparameterisation f: it still needs to be continuous but not bijective. This means that the person and the dog can walk backward. To determine if the weak Fréchet distance $\delta_{wF}(\pi, \sigma)$ is at most d, we need to find only a (potentially not xy-monotone) path through the free space from s to t in $\mathcal{D}_d(\pi, \sigma)$. Buchin et al. [8] showed that there is no strongly subquadratic time algorithm for approximating the weak Fréchet distance within a factor less than 3 unless the strong exponential-time hypothesis fails.

Buchin et al. [7] proposed the *partial Fréchet similarity* (partial similarity in short) to deal with the Fréchet distance's sensitivity to outliers. Instead of determining whether a leash of length d is enough to complete the walk, partial similarity determines how much can be completed given a leash of length d. The partial similarity is the total length of the portion of two curves that are matched under the Fréchet distance d.

Let $\|xy\|_p$ be the distance between point x and point y under the L_p norm. Let $\|v\|$ be the L_2 norm of the vector v. Under the L_p metric, given the desired Fréchet distance d, the partial similarity $S_{f,g}(\pi, \sigma)$ of curves π and σ with respect to the reparameterisations f and g is formally defined as follows [7].

$$S_{f,g}(\pi, \sigma) = \int_{\|\pi(f(t))\sigma(g(t)\|_p \leq d} (\|\pi(f(t))'\| + \|\sigma(g(t))'\|)dt$$

Naturally, we want to compute a pair of reparameterisations f and g that maximise the partial similarity. To do this, Buchin et al. [7] proposed a cubic time algorithm under L_1. They showed that it is sufficient to find an xy-monotone and *rectilinear* path P from s to t such that P intersects as much free space as possible, where s (resp. t) is the bottom-left (resp. top-right) corner of the freespace diagram.

Under the weak Fréchet distance, the monotonicity requirement is removed. But since a path P can traverse back and forth in the freespace diagram, it is no longer meaningful for P to intersect as much free space as possible. We instead are interested in computing a path that intersects as little forbidden space as possible to minimise the portions of the two curves that are not matched within distance d. Therefore, solving the partial weak Fréchet distance problem under the L_2 metric is equivalent to finding a weighted rectilinear shortest path amidst a set of $O(n^2)$ non-overlapping and convex 0-regions embedded in the plane (the forbidden space) with weight 1. By Fact 12, a 0-region is the free space within a cell, which has constant complexity.

Amidst the 0-regions and measured in L_p metric, let OPT_{L_p} denote the weight of the weighted shortest path from s to t, and let ALG_{L_p} denote the weight of the weighted path computed by Theorem 1. Since $OPT_{L_1} \leq \sqrt{2} \cdot OPT_{L_2}$, we have

$$OPT_{L_2} \leq OPT_{L_1} \leq \sqrt{2} \cdot OPT_{L_2} \leq \sqrt{2} \cdot ALG_{L_2} \leq \sqrt{2} \cdot (1 + \varepsilon) OPT_{L_2},$$

which leads to the following theorem.

▶ **Theorem 2.** *One can approximate the partial weak Fréchet similarity of two curves within a factor of* $(\sqrt{2} + \varepsilon)$ *in* $O((n^2/\varepsilon^2) \log(n/\varepsilon))$ *expected time.*

4 Shortest path amidst 0-regions and obstacles

In this section, we generalise our data structure from Section 2 to allow convex obstacles that cannot be traversed, i.e., obstacles with weight ∞. Our problem is finding an approximate shortest path amidst 0-regions and obstacles. More concretely, we consider the following problem.

▶ **Problem 2.** *In the planar-subdivision induced by the plane with weight* 1*, and a set of non-overlapping convex regions consisting of obstacles with weight* ∞*, and* 0*-regions with weight* 0*, given an approximation error* $0 < \varepsilon < 1$*, find a* $(1+\varepsilon)$*-approximate weighted shortest path from point* s *to point* t*.*

In this section, let $\mathbf{d}(A, B)$ denote the minimum distance between two geometric regions A and B in a $0/1/\infty$ weighted setting. The Θ-graph can be constructed in an environment with obstacles. Clarkson [10] described such construction over points and polygonal obstacles, and proved that a path that $(1+\varepsilon)$-approximates $\mathbf{d}(a, b)$ exists in the Θ-graph, where a and b are vertices. We will use this Θ-graph in the rest of the paper.

Like in the previous section, we will first describe the construction of the data structure $\mathcal{B}$ and analyse the time and space complexity. We then show that $\mathcal{G} \in \mathcal{B}$ contains a good path between every pair of sample points. We use this to argue the approximation ratio for arbitrary s and t.

4.1 Construction of the data structure

In order to deal with obstacles, we need to define two new types of sample points. For clarity, we refer to the sample points defined previously as the *original sample points*. In Section 2, a trapezoidal map $\mathcal{M}(k)$ was only used to determine if two 0-regions should be connected, and we did not explicitly compute the intersection of a vertical segment and the boundary of a region. With the introduction of obstacles, we do need such intersections. Consider a sample point a. When constructing $\mathcal{M}(k)$, we shoot two vertical rays from $a \in A$, one upwards and one downwards. Let p be the first intersection of $r(a, k)$ with the boundary of some region that is not A. We call p a *propagated sample point*.

The other type of sample points we need is the *tangent sample points*. Given two disjoint obstacles A and B, and a common tangent l, if l touches A at point a and B at b, we add a and b as tangent sample points. When we say a point a is a sample point, a can be any type of sample point.

Recall that $\texttt{simpl}(A)$ is the simplified region by connecting every pair of adjacent sample points of A. $\texttt{simpl}(\mathcal{Z})$ is the set of simplified 0-regions, and $\texttt{simpl}(\mathcal{O})$ is the set of simplified obstacles. We formally define the construction of our data structure. Given a set $\mathcal{O}$ of convex obstacles and a set $\mathcal{Z}$ of convex 0-regions, we build our data structure using Algorithm 2. See the full version for the analysis.

▶ **Lemma 13.** *Given an approximation error* $0 < \varepsilon < 1$*, and* n *non-overlapping convex regions including* 0*-regions and obstacles with total complexity* N*, one can build the data structure* $\mathcal{B}$ *in* $O(N + (n/\varepsilon^3)(\log(n/\varepsilon) + \log N))$ *expected time, and the total size of* $\mathcal{B}$ *is* $O(N + n/\varepsilon^3)$*.*

The structure of the rest of the section is as follows. By Lemma 14, the distance between two adjacent sample points on the boundary of an obstacle approximates the straight line segment. Therefore, we show that we can "snap" the vertices of an optimal path to our sample points. For every segment $pq \subseteq P^*$, we then argue that either the trapezoidal map or the Θ-graph contains a path approximating $\|pq\|$ to within a factor of $1+\varepsilon$.

Algorithm 2 Construct $\mathcal{B}$ with 0-regions and obstacles.

This algorithm takes as input a set of non-overlapping and convex regions, including 0-regions $\mathcal{Z}$ and obstacles $\mathcal{O}$, and constructs a data structure $\mathcal{B}$. The undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is initially empty.

1) Compute the original sample points $\mathcal{SP}(\mathcal{Z}) \cup \mathcal{SP}(\mathcal{O})$, and add them to $\mathcal{V}$.
2) For each direction $r(k\theta)$, generate a trapezoidal map $\mathcal{M}(k)$ using $\texttt{simpl}(\mathcal{Z}) \cup \texttt{simpl}(\mathcal{O})$.
3) For every $\mathcal{M}(k)$, do the following for every face F adjacent to A and B, $A \neq B$.
   a. Compute the propagated sample points, and add them to $\mathcal{V}$.
   b. If A and B are both 0-regions, add $e = (ak(A), ak(B))$ to $\mathcal{E}$, and set $\mathbf{w}(e) = \|AB\|$.
   c. If at least one of A and B is an obstacle, let ab and $a'b'$ be the vertical segments defining F, where $a, a' \in A$, and $b, b' \in B$. Add edges $e_1 = (a, b)$ and $e_2 = (a', b')$ to $\mathcal{E}$. Set $\mathbf{w}(e_1) = \|ab\|$ and $\mathbf{w}(e_2) = \|a'b'\|$.
   d. If A and B are both obstacles, we compute their common tangents. For each common tangent that touches A at a and B at b, add a and b to $\mathcal{V}$.
4) Redefine $\texttt{simpl}(A)$ as the polygon generated by connecting adjacent sample points of A, original, propagated, and tangent sample points included. With $\mathcal{V}$ and $\texttt{simpl}(\mathcal{O})$ as the input, generate a Θ-graph $\mathcal{G}_\Theta = (\mathcal{V}, \mathcal{E}_\Theta)$.
   - For each edge $(p, q) \in \mathcal{E}_\Theta$, if p and q belongs to different regions A and B, add $e = (p, q)$ to $\mathcal{E}$ and set $\mathbf{w}(e) = \|pq\|$.
5) For every pair of adjacent sample points a, a' on an obstacle, add $e = (a, a')$ to edges, and set $\mathbf{w}(e) = \|aa'\|$.
6) Pick an arbitrary sample point as the anchor $ak(A)$ for every $A \in \mathcal{Z}$. For each sample point a of a 0-region A, add $e = (a, ak(A))$ to $\mathcal{E}$, and set $\mathbf{w}(e) = 0$.
7) Return $\mathcal{B} = \{\mathcal{M}(k) \mid \forall k \in [0, 2\pi/\theta), k \in \mathbb{Z}\} \cup \{\mathcal{G}, \mathcal{G}_\Theta\}$ as the data structure.

4.2 Walking on the boundary is not expensive

We first argue that if a and a' are adjacent sample points of A, then $\|aa'\|$ approximates $\|\partial A(a, a')\|$ within a factor of $\sec(\theta/2)$. Therefore, if we find a path P amidst the simplified obstacles, we only need to pay a small factor to transform P into a path amidst the original obstacles. Then, we prove that if pq is part of the optimal path, we can replace pq with a path $P \subseteq \mathcal{E}$, such that $\mathbf{w}(P)$ approximates $\|pq\|$. See the full version for a formal proof.

▶ **Lemma 14.** *Let a and b be adjacent sample points on ∂A, where a appear after b in a counter-clockwise walk. We have that $\|\partial A(a, b)\| \leq \sec(\theta/2) \cdot \|ab\|$.*

The above lemma implies the following. Let $\mathbf{d_S}(a, b)$ be the distance between point a and point b amidst simplified obstacles and simplified 0-regions, and let P be the path achieving this distance. If we partition P using the sample points, in the worst case, each segment connects adjacent sample points on obstacles. This implies the following corollary.

▶ **Corollary 15.** *Let a and b be two adjacent sample points. We have that $\mathbf{d}(a, b) \leq \sec(\theta/2) \cdot \mathbf{d_S}(a, b)$.*

4.3 Snapping a segment of the optimal path to the sample points

Gewali et al. [15] defined three types of *locally optimal* edges joining two simple polygonal regions, and they proved that the shortest path from s to t must be comprised of these locally optimal edges [15, Lemma 2.5] (ignoring edges in 0-regions). For convex obstacles and 0-regions, we need to consider only four types of segments.

▶ **Fact 16.** *If segment pq is in the optimal weighted path P^* amidst convex and non-overlapping 0-regions and obstacles, there must exists two supporting lines $l_t(p)$ and $l_t(q)$ such that pq belong to one of the following cases (ignoring segments in 0-regions).*

1) *pq connects two 0-regions such that $pq \perp l_t(q)$ and $pq \perp l_t(p)$.*
2) *pq connects the point p on a 0-region A and the point q on an obstacle B such that $pq \subset l_t(q)$, and $pq \perp l_t(p)$.*
3) *pq lies on one of the common tangent of two different obstacles.*
4) *p and q are two points on the same obstacle, and $pq = \partial A(p, q)$ or $pq = \partial A(q, p)$.*

In Section 4.3.1, 4.3.2, and 4.3.3, we handle each type of edge in Case 1, 2, and 3, respectively. For an edge pq in each of these cases, we argue that there exists a good path in $\mathcal{G} \in \mathcal{B}$ that approximates pq. Section 4.4 summarises the approximation ratio using the Case 4 edges. In the following lemmas, we assume without loss of generality that pq lies between the direction $r(k\theta)$ and $r((k+1)\theta)$, and α (resp. β) is the measure of the acute angle between $r(p, k\theta)$ (resp. $r(p, (k+1)\theta)$) and pq. By Corollary 15, we consider only simplified obstacles. We delegate the full proofs to the appendix.

4.3.1 Case 1: pq connects two 0-regions

We observe that, unfortunately, Lemma 8 does not trivially apply. When we rotate pq to pq', if pq' overlaps obstacles, a path generated using the skewed set of obstacles can be much longer than $\|pq'\|$, since the path would have to take a detour around the obstacles. The following lemmas resolve this issue. See the full version for full proofs of the lemmas below.

▶ **Lemma 17.** *Let A and B be two 0-regions. Let $pq \subseteq P^*$, where $p \in \partial A$, and $q \in \partial B$. If $\|qb\| \geq (\sin(\alpha)/\cos(\theta)) \cdot \|pq\|$, where b is a sample point adjacent to q, then there exists a sample point $a \in \partial A$ such that $\mathbf{d_S}(a, b) \leq (\cos(\beta)/\cos(\theta)) \cdot \|pq\|$.*

▶ **Lemma 18.** *Let A and B be two 0-regions. Let $pq \subseteq P^*$, where $p \in \partial A$, and $q \in \partial B$. If $\|qb\| < (\sin(\alpha)/\cos(\theta)) \cdot \|pq\|$ and $\|pa\| < (\sin(\beta)/\cos(\theta)) \cdot \|pq\|$, where a (resp. b) is a sample point adjacent to p (resp. q), then $\mathbf{d_S}(a, b) \leq (\sin(\alpha) + \sin(\beta))/(\cos(\theta)\sin(\theta)) \cdot \|pq\|$.*

We combine Lemma 17 and Lemma 18 to obtain a bound on the path length where p and q both lie on 0-regions. Note that when two points p and a lie on the boundary of the same 0-region, $\mathbf{d_S}(a, p) = 0$.

▶ **Lemma 19.** *Let A and B be two convex 0-regions. Let $pq \subseteq P^*$, where $p \in \partial A$ and $q \in \partial B$. There exists a pair of sample points $a \in A$ and $b \in B$, such that $\mathbf{d_S}(p, a) + \mathbf{d_S}(a, b) + \mathbf{d_S}(b, q) \leq \max\{(\cos(\beta)/\cos(\theta)), (\sin(\alpha) + \sin(\beta))/(\cos(\theta)\sin(\theta))\} \cdot \|pq\|$.*

4.3.2 Case 2: pq connects two obstacles

Using Fact 16, we know that if pq connects two obstacles, then pq must lie on a common tangent of A and B. When two obstacles are close, p (and q) may be very far from its adjacent original sample points. Hence, we need the tangent sample points when two obstacles are close (connected via a trapezoidal map). We now show that if p and q lie on obstacles, an approximate path in $\mathcal{B}$ exists. See the full version for a full proof of the lemma below.

▶ **Lemma 20.** *Let A and B be two convex obstacles. Let $pq \subseteq P^*$, where $p \in \partial A$ and $q \in \partial B$. There exists a pair of sample points $a \in A$ and $b \in B$, such that $\mathbf{d_S}(p, a) + \mathbf{d_S}(a, b) + \mathbf{d_S}(b, q) \leq (1/\cos(\theta)) \cdot \|pq\|$.*

4.3.3 Case 3: pq connects a 0-region and an obstacle

In this section, we prove that if pq connects an obstacle A and a 0-region B, there is a path P that approximate pq. See the full version for a full proof of the lemma below.

▶ **Lemma 21.** *Let A be a convex obstacle and let B be a convex 0-region. Let $pq \subseteq P^*$, where $p \in \partial A$ and $q \in \partial B$. There exists a pair of sample points $a \in A$ and $b \in B$, such that $\mathbf{d_S}(p,a) + \mathbf{d_S}(a,b) + \mathbf{d_S}(b,q) \leq (\cos(\beta)/\cos(\theta)) \cdot \|pq\|$.*

4.4 The quality of the path

In Lemma 19, 20, and 21, we have shown that for every segment pq in Case 1-3 in Fact 16, either there exists a path $P \subseteq \mathcal{E}$ such that $\mathbf{w}(P)$ approximates $\|pq\|$, or there exist two sample points a and b such that $\mathbf{d_S}(p,a) + \mathbf{d_S}(a,b) + \mathbf{d_S}(b,q)$ approximates $\|pq\|$. Taking the maximum ratio in the three lemmas, we have the following.

$$\mathbf{d_S}(p,a) + \mathbf{d_S}(a,b) + \mathbf{d_S}(b,q) \leq \frac{2\sin(\frac{\theta}{2})}{\cos(\theta)\sin(\theta)} \cdot \|pq\|$$

For a Case 4 segment $\partial A(p,q)$, where both p and q lies on the obstacle A, assume without loss of generality that p occurs before q in P^*, and let $pq = \partial A(p,q)$.

If $\partial A(p,q)$ contains no sample point, then assume that the optimal path uses segment $p'p$ to reach A, and qq' to leave A. We argue that there exists an approximate path P that approximates $\|p'p\| + \|\partial A(p,q)\| + \|qq'\|$. Let a (resp. b) be the closest sample point to p (resp. q), such that $\partial A(p,q) \subseteq \partial A(a,b)$. In Lemma 19, 20, and 21, we have payed for a path $P_p \subseteq \mathcal{E}$ from p' to p though a and a path $P_q \subseteq \mathcal{E}$ from q to q' through b. Since there is no sample point on $\partial A(p,q)$, instead of going from a to p and q to b, we take the path ab directly. The unused cost of $\mathbf{d_S}(a,p)$ and $\mathbf{d_S}(q,b)$ pays for $\|ab\|$.

Let $p_\perp$ (resp. $q_\perp$) be the orthogonal projection of p (resp. q) on ab. Clearly, $\|\partial A(a,p)\| \geq \|ap_\perp\|$ and $\|\partial A(q,b)\| \geq \|bq_\perp\|$. By Lemma 14, $\|\partial A(p,q)\| \leq \sec(\theta/2) \cdot \|p_\perp q_\perp\|$. Therefore, we connect P_p and P_q using ab to generate a path P, and we have that

$$\|P\| \leq \frac{2\sin(\frac{\theta}{2})}{\cos(\theta)\sin(\theta)} \cdot (\|p'p\| + \|\partial A(p,q)\| + \|qq'\|).$$

If $\partial A(p,q)$ contains at least one sample point $\{a, ..., b\}$, then by Lemma 14, we have that

$$\mathbf{d_S}(p,a) + \mathbf{d_S}(a,b) + \mathbf{d_S}(b,q) \leq \sec\left(\frac{\theta}{2}\right) \cdot \|pq\|.$$

Bose and van Renssen [6] showed that in an environment with polygonal obstacles, the Θ-graph described by Clarkson [10] has a spanning ratio of at most $r_\theta = 1+2\sin(\theta/2)/(\cos(\theta/2)-\sin(\theta/2))$. We also need to apply the factor to traverse the boundaries of convex obstacles to account for the difference compared to the boundaries of simplified obstacles, as in Lemma 14. For a desired approximation factor $1+\varepsilon$, we can compute a $\theta \in O(\varepsilon)$, and thus $\mathcal{B}$ contains a $(1+\varepsilon)$-spanner of the sample points.

▶ **Lemma 22.** *In $\mathcal{B}$, there exists a path $P \subseteq \mathcal{E}$ between any pair of sample points (a,b) such that $\mathbf{w}(P) \leq (1+\varepsilon) \cdot \mathbf{d}(a,b)$.*

Given a pair of query points s and t, we add s and t to $\mathcal{B}$. We treat both s and t as obstacles with no interior to enable previous lemmas, yielding the theorem below. See the full version for details.

▶ **Theorem 3.** *Consider a planar subdivision induced by a plane with a weight of* 1*, consisting of two sets of convex and non-overlapping regions: 0-regions $\mathcal{Z}$ with a weight of* 0*, and obstacles $\mathcal{O}$ with a weight of ∞. Let $n = |\mathcal{Z}| + |\mathcal{O}|$ and let N denote be the total number of vertices in $\mathcal{Z} \cup \mathcal{O}$. For any approximation factor $0 < \varepsilon < 1$, a data structure $\mathcal{B}$ can be constructed over $\mathcal{Z} \cup \mathcal{O}$ in $O(N + (n/\varepsilon^3)(\log(n/\varepsilon) + \log N))$ expected time, with a total size of $O(N + n/\varepsilon^3)$. When queried with arbitrary points s and t, $\mathcal{B}$ returns a path P from s to t in $O(N + n/\varepsilon^3 + (n/\varepsilon^2)\log(n/\varepsilon) + (\log N)/\varepsilon)$ time, ensuring that $\mathbf{w}(P) \leq (1+\varepsilon) \cdot \mathbf{w}(P^*)$, where P^* is the optimal weighted shortest path from s to t.*

References

1 Lyudmil Aleksandrov, Anil Maheshwari, and Jörg-Rüdiger Sack. Approximation algorithms for geometric shortest path problems. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, pages 286–295, New York, NY, USA, May 2000. Association for Computing Machinery. doi:10.1145/335305.335339.

2 Lyudmil Aleksandrov, Anil Maheshwari, and Jörg-Rüdiger Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *Journal of the ACM*, 52(1):25–53, January 2005. doi:10.1145/1044731.1044733.

3 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 05:75–91, March 1995. doi:10.1142/S0218195995000064.

4 Prosenjit Bose, Jean-Lou de Carufel, Pat Morin, André van Renssen, and Sander Verdonschot. Towards tight bounds on theta-graphs: more is not always better. *Theoretical Computer Science*, 616:70–93, February 2016. doi:10.1016/j.tcs.2015.12.017.

5 Prosenjit Bose, Guillermo Esteban, David Orden, and Rodrigo I. Silveira. On approximating shortest paths in weighted triangular tessellations. *Artificial Intelligence*, 318:103898, May 2023. doi:10.1016/j.artint.2023.103898.

6 Prosenjit Bose and André van Renssen. Spanning properties of Yao and theta-graphs in the presence of constraints. *International Journal of Computational Geometry & Applications*, 29(02):95–120, June 2019. doi:10.1142/S021819591950002X.

7 Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 645–654, USA, January 2009. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611973068.71.

8 Kevin Buchin, Tim Ophelders, and Bettina Speckmann. SETH says: weak Fréchet distance is faster, but only if it is continuous and in one dimension. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2887–2901, USA, January 2019. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611975482.179.

9 Siu-Wing Cheng, Jiongxin Jin, and Antoine Vigneron. Triangulation Refinement and Approximate Shortest Paths in Weighted Regions. In *Proceedings of the 2015 Annual ACM-SIAM Symposium on Discrete Algorithms*, Proceedings, pages 1626–1640. Society for Industrial and Applied Mathematics, December 2014. doi:10.1137/1.9781611973730.108.

10 Kenneth L. Clarkson. Approximation algorithms for shortest path motion planning. In *Proceedings of the nineteenth annual ACM Symposium on Theory of Computing*, pages 56–65, New York, NY, USA, January 1987. Association for Computing Machinery. doi:10.1145/28395.28402.

11 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-77974-2.

12 Sarita de Berg, Guillermo Esteban, Rodrigo I. Silveira, and Frank Staals. Exact solutions to the Weighted Region Problem, February 2024. arXiv:2402.12028 [cs]. doi:10.48550/arXiv.2402.12028.

13 Jean-Lou de Carufel, Amin Gheibi, Anil Maheshwari, Jörg-Rüdiger Sack, and Christian Scheffer. Similarity of polygonal curves in the presence of outliers. *Computational Geometry*, 47(5):625–641, July 2014. doi:10.1016/j.comgeo.2014.01.002.

14 Jean-Lou de Carufel, Carsten Grimm, Anil Maheshwari, Megan Owen, and Michiel Smid. A note on the unsolvability of the weighted region shortest path problem. *Computational Geometry*, 47(7):724–727, August 2014. doi:10.1016/j.comgeo.2014.02.004.

15 Laxmi P. Gewali, Alex C. Meng, Joseph S. B. Mitchell, and Simeon Ntafos. Path planning in $0/1/\infty$ weighted regions with applications. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, pages 266–278, New York, NY, USA, January 1988. Association for Computing Machinery. doi:10.1145/73393.73421.

16 Mark Lanthier, Anil Maheshwari, and Jörg-Rüdiger Sack. Approximating weighted shortest paths on polyhedral surfaces. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, pages 274–283, New York, NY, USA, August 1997. Association for Computing Machinery. doi:10.1145/262839.262984.

17 Joseph S. B. Mitchell and Christos H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *Journal of the ACM*, 38(1):18–73, January 1991. doi:10.1145/102782.102784.

18 Zheng Sun and John H. Reif. On finding approximate optimal paths in weighted regions. *Journal of Algorithms*, 58(1):1–32, January 2006. doi:10.1016/j.jalgor.2004.07.004.

19 Victor Andreevich Toponogov. *Differential Geometry of Curves and Surfaces: A Concise Guide*. Birkhauser, 2006.

Link Diameter, Radius and 2-Point Link Distance Queries in Polygonal Domains

Mart Hagedoorn
Technische Universität Dortmund, Germany

Valentin Polishchuk
Linköping University, Sweden

Abstract

We show how to preprocess a polygonal domain with holes so that the link distance (the number of links in a minimum-link path) between two query points in the domain can be reported efficiently. Using our data structures, the link diameter of the domain (i.e., the maximum number of links that may be required in a minimum-link path between two points in the domain) as well as the link center and radius of the domain (i.e., the point minimizing the maximum link distance to the furthest point in the domain and this maximum link distance) can be found in polynomial time. We also give a simpler algorithm for finding the link diameter, not using the link distance query structures. Answering 2-point link distance queries and computing the link diameter/radius/center in polygonal domains have been open questions since these problems were studied for simple polygons in the 90's.

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

Keywords and phrases Minimum-link paths, link distance, diameter, center, radius, 2-point distance queries

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.34

1 Introduction

In computational geometry, the complexity of paths within polygonal domains is often measured not only by their length but also by their structural simplicity. This is particularly relevant in areas such as motion planning, robotics, and geographic information systems, where simplifying complexity can reduce computational overhead. A minimum-link (minlink) path between points s, t in a polygonal domain P is a polygonal s-t path with the minimum number of edges (links). The number of links in a minlink path is the *link distance* between s and t, providing such a measure of path complexity. The *link diameter* of P is the maximum possible link distance between any two points in the domain. The *link center* of the domain is the point minimizing the maximum link distance to points of P; this maximum distance is the *link radius*. The above notions are analogous to the same concepts (diameter, center, radius) for arbitrary metric spaces.

1.1 On the complexity of computing optimal geometric paths

Both geodesic (shortest) and minlink paths may be more intricate than it feels from a glance:

- It is folklore that geodesic paths may be found by searching the visibility graph of the domain; however, even if the vertices have integer coordinates, it is not known whether one can compare, in polynomial time, the length of a path to an integer – the length is the sum of square roots and no polynomial-time algorithm is known for comparing the sum to a number.
- As explained in Section 2.1, vertices of a minlink path cannot always be snapped to a discrete set of points, leading to high bit complexity of the paths even in simple polygons [19, 21]. In a polygonal domain with holes, it is not immediately clear how to

© Mart Hagedoorn and Valentin Polishchuk;
licensed under Creative Commons License CC-BY 4.0

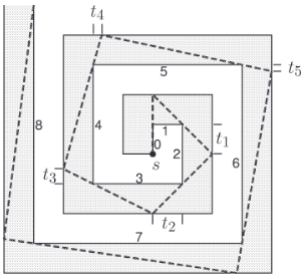


Figure 1 Figure 1 from [19]: the coordinates of the vertex t_1 of the path are obtained as a solution to a system of 2 linear equations with 2 unknowns and integer coefficients – they are thus represented by a ratio of integers; subsequent bend points are solutions to systems of linear equations whose coefficients have higher and higher bit complexity (they are ratios of larger and larger integers).

find a minlink path in polynomial time. One may compute the k-link reachable regions, from the starting point, for increasing k, but bounding the complexity of the regions (and hence the algorithm's runtime) requires an insight [25].

The problems considered in this paper (2-point shortest path queries, diameter and radius) take the complexity to the next level. Quoting [5]: "no algorithm for computing the diameter and radius under the link distance is known, not even one that runs in exponential time." In particular, nothing directly prevents the problems from being NP-hard or $\exists\mathbb{R}$-hard. We show that the problems admit polynomial-time solutions. The running times are high: we leave the speed up as future work. (Note that there are plenty of algorithms with high running times that "just" show polynomiality of long-standing open problems; computational geometry examples range from a 20yr-old $O(n^{29}\log n)$-time approximation algorithm [13] for covering an n-gon with fewest convex subpolygons to a recent $O(n^{107})$-time algorithm for star-partitioning a simple polygon [1].)

2 Preliminaries and related work

Let n denote the number of vertices of P. The *visibility polygon* (VP) of a point $p \in P$ is the set of points in P seen by p. The *weak visibility polygon* of a subset $S \subseteq P$ is the union of the VPs of the points of S; equivalently, the weak visibility polygon is the set of points seen by at least one point of S. The *visibility graph* (VG) of P is the graph on vertices of P, whose edges connect pairs of mutually visible vertices. We assume that edges of P are also edges of the VG (i.e., that neighboring vertices of P see each other along the boundary edge).

2.1 Link distance and bit complexity

Computing the link diameter and radius/center of a polygonal domain with holes has been open since analogous problems were considered for simple polygons some 30 years ago [4, 28]. The reason why even an exponential-time solution is not obvious is because vertices of a minlink path do not necessarily lie on VG or some other discrete structure determined from the polygon (Fig. 1): it was shown [19, 21] that $\Theta(n\log n)$ bits may be required to represent coordinates of vertices of the minlink path, even if the vertices of P have integer coordinates specified with $O(\log n)$ bits.

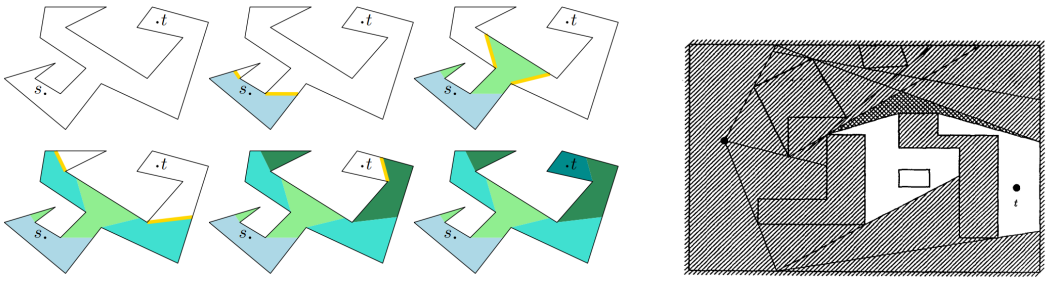


Figure 2 Staged illumination in a simple polygon (left) and in a polygonal domain with holes (right). Left: Figure 3 from [24]: The windows are yellow. Right: Figure 5 from [25]: The cross-hatched area is what is swept by the lower envelope of a set of windows when this lower envelope is a "rubber band" pinned at its endpoints: when released, the rubber band snaps onto edges of VG.

2.2 Staged illumination

Algorithms for computing the link distance from a source point s [15, 17, 25, 26] employ the "staged illumination" paradigm (see, e.g., the handbooks [23, Chapter 12] and [30, Chapter 31.3]): At the first stage, place the light source at s and illuminate VP of s – this is the set of points with link distance 1 from s. At the beginning of any subsequent stage, the boundary between the lit and the dark portions of P is defined by a set of line segments we call *windows* (any window starts at a vertex and ends on the boundary of P) which bound the weak visibility polygon of the area lit at the previous stage.

Figure 2 (left)t illustrates the staged illumination in a simple polygon. To efficiently run the staged illumination in a polygonal domain with holes [25], some chains of windows are replaced by their relative convex hulls within P – essentially snapping windows to edges of VG. See Figure 2 (right) and [25], in particular, Figures 2–5 therein for the details.

The *greedy* path [4, 22] from s to a window w is the minlink path whose links are all aligned with windows and whose last link is aligned with w. E.g., the path in Fig. 1 is greedy.

2.3 Distance map

The *link distance map*, denoted LDM(s), from the *source* point s is a decomposition of P into cells such that the link distance from s to any point within one cell is the same. Note that it is not required that the cells of LDM are maximal (i.e., that the link distance necessarily changes as you step from a cell to a neighboring cell): the only requirement is that the distance never changes within a cell.

For simple polygons, an LDM is really a by-product of the staged illumination (the edges of the map are the windows). Both a single minlink path and the LDM can be computed in $O(n)$ time (because in a simple polygon the windows are pairwise disjoint, the LDM in it is also called a "window partition" [4, 28]).

In polygons with holes the LDMs can have $\Omega(n^4)$ complexity [29], and extra care must be taken in order to produce the LDM. In [25, Theorem 12] it was shown how to build the LDM in $\tilde{O}(n^5)$ time, where $\tilde{O}$ notation suppresses polylogarithmic factors. The map is defined by an arrangement of windows (called "illumination edges" in [25]) within the domain.

Similarly to constructing an LDM from a point, one can build LDM from a source *chord* c of P (Fig. 3): the weak VP of c is the set of points reachable with 1 link from c, and the subsequent steps of the illumination are defined in the same way as for a point source. LDMs from segments were built already in early work [28] on link distance; in particular, LDMs from chords were used for finding the link center of a simple polygon [11].

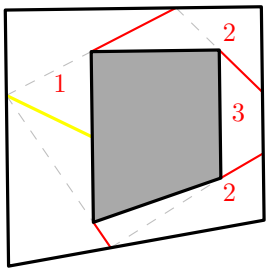


Figure 3 LDM(c) is obtained by starting the illumination from c (yellow); the edges of the map are solid red, and the cells are marked with the link distance to c.

2.4 1-point queries

After being preprocessed for point location, an LDM allows one to report the link distance from the source to a query point q in optimal $O(\log n)$ time, by locating the cell of the map that contains q (enhanced with the backpointers, the map lets one report an actual minlink path in the additional time proportional to the number of links). An LDM is thus analogous to the *shortest-path map* (SPM(s)) in P, which decomposes the domain into cells such that the length of the geodesic (shortest) path from s to any point in one cell is given by the same formula. Note that while SPMs have linear complexity and can be built in linear (for simple polygons) or near-linear (for polygons with holes) time [33, 34], finding even a single minlink path in polygons with holes is 3SUM-hard [24] (an $\tilde{O}(n^2)$-time algorithm is given in [25]).

3 Prior work on 2-point queries and our results

While LDMs and SPMs give complete answers to the *1-point* link and geodesic distance query resp. (for a *fixed* source, report the distance from a query point to the source), for the *2-point* distance query problem (report the distance between two query points s and t), the solutions achieving optimal $O(\log n)$ query time are much more involved [4, 9, 10, 16, 32]:

- An optimal (linear-preprocessing) algorithm is known only for 2-point geodesic distance queries in simple polygons [16].
- A cubic-size data structure for link distance queries in simple polygons was given in [4].
- For geodesic distance queries in polygonal domains, two methods were presented in [9]:
  - **2d SPM equivalence decomposition** splits P into (a polynomial number of) cells such that SPM(s) remains combinatorially the same (called "topologically equivalent" in [7, 9]) while s stays within one cell, and stores a *parametric* SPM for each cell (the parameter being the location of s within the cell: the parametric SPM records how the edges of SPM(s) change depending on where s is). After preprocessing the parametric SPM for parametric point location [9, Section 4.2], 2-point geodesic distance queries may be answered by locating s in the decomposition of P and then locating t in the parametric SPM(s).
  - **4d geodesic distance equivalence decomposition** splits $P \times P$ into (a polynomial number of) cells such that the geodesic distance between any pair of points (s, t) within one cell is given by the same formula, i.e., by the same function of (s, t). (On a detailed note, [9] presented two 4d structures: the vanilla $O(n^{22+\varepsilon})$-space 4d decomposition introduced in [9, Section 3] first, and the improved $O(n^{15+\varepsilon})$-size data structure in [9, Theorem 3.1] – we refer here to the former decomposition because the improved one, being specific to geodesic paths, does not work for us; similarly we did not see how to extend to link distance the recent data structure for geodesic paths from [10].)

- No data structure for 2-point link distance queries in polygonal domains with holes has been known previously. One of the contributions of this paper is working out such data structures (analogous to the 2d and the 4d decomposition for geodesic queries [9]) by extending the solution of [4] (for 2-point link distance queries in simple polygons) to polygons with holes. Specifically, we provide the following two data structures:
  2d LDM equivalence decomposition is the decomposition of P into (a polynomial number of) cells such that LDM(s) remains combinatorially the same while s stays in one cell. The map, i.e., its vertices and edges (windows), is a known function of s (the functions may be different in different cells, but within one cell the function is same).
  4d link distance equivalence decomposition decomposes $P \times P$ into (a polynomial number of) cells such that the link distance between any pair (s, t) in one cell is the same.

3.1 Diameter and radius/center

Computing the diameter and radius is intimately connected to the 2-point distance query structures, described above, via the following observation applicable to both geodesic and link distances – hence the prefix Meta:

▶ **MetaTheorem 1.** *If the 2d map equivalence decomposition of P or the 4d distance equivalence decomposition of $P \times P$ can be built in polynomial time, then the diameter and the radius/center of P can be computed in polynomial time.*

Proof. Go through every cell σ of the 2d map equivalence decomposition.

- For the geodesic distance, the maximum distance from s is attained at a vertex of SPM(s) [7, Lemma 1]. Since in every cell σ, it is known how SPM(s) changes with s, distances from s to all vertices are also known, and hence the maximum distance from $s \in \sigma$ to a point in P is also known (from the upper envelope of the distances).
- For the link distance, the maximum distance from $s \in \sigma$ is the distance to points in the cell of LDM(s) farthest from s (any point in a cell of LDM(s) has the same link distance from s – no need for the envelope).

To identify the diameter, take the maximum distance to the furthest point, and for the radius take the minmax; for the final answer, choose the best σ accordingly. To find the diameter, one may also go through every cell of the 4d decomposition and in every cell find the pair (s, t) that maximizes the s-t distance – the diameter will be given by the overall maximum. ◀

3.2 State of the art and our contribution

Solutions based on MetaTheorem 1 are not very efficient, and faster algorithms exist for finding both the geodesic diameter [6] and radius [31] of polygonal domains (in simple polygons, linear-time algorithms exist for both geodesic diameter [18] and radius [3]; link diameter [27] and radius [11] of simple polygons can be found in near-linear time). Existing results regarding diameter and radius are summarized in Table 1: essentially, the only case that has been missing is link distance in polygons with holes. Similarly, for simple polygons 2-point distance query data structures are known both for geodesic [16] and link distance [4], while for polygonal domains the problem has been solved only for the geodesic distance [9].

In this paper, we fill the gaps by showing how to construct the 2d map equivalence decomposition of a polygonal domain P and the 4d distance equivalence decomposition of $P \times P$ under the link distance. By MetaTheorem 1, this leads to polynomial-time algorithms for finding the link diameter and radius/center of the domains P.

Table 1 A sub-table of [5, Table 1]: known results for computing the diameter and radius in simple polygons (left) and polygonal domains (right).

	simple polygon		polygonal domain	
	Diameter	Radius	Diameter	Radius
Geodesic	$O(n)$ [18]	$O(n)$ [3]	Poly-time [6]	Poly-time [7, 31]
Link	$O(n \log n)$ [27]	$O(n \log n)$ [11, 20]	Poly-time [this paper]	

The rest of the paper is organized as follows. Section 4 gives a simple algorithm to compute the link diameter (without resorting to MetaTheorem 1), and Section 5 presents data structures for 2-point link distance queries. Our data structures reuse, extend and combine the techniques of [4] (link distance queries in simple polygons) and [9] (geodesic distance queries in polygonal demands with holes): analogously to [9], our data structures are the 2d LDM equivalence decomposition and the 4d link distance equivalence decomposition. We conclude in Section 6 with some open problems.

4 A simple algorithm for link diameter

Throughout the paper, "distance" will mean link distance, "length" of an s-t path will mean the link distance between s and t along the path, etc. Slightly abusing the terminology, we will use the term *diameter* to mean both a minlink path between two diametrical endpoints and the number d of links in such a path.

The idea of the algorithm in this section is that finding the diameter is easy if one of the following holds: the diameter is 1 or 2, a diameter endpoint is on a vertex, or a diameter edge is aligned with an edge of VG. The main structural observation is that we are always in one of the above-listed easy cases. The diameter is 1 if and only if P is a convex polygon. The diameter is at most 2 if and only if for any $s, t \in P$ there are rays from s and t which intersect each other before exiting the domain: the intersection point p is the bend of a 2-link s-t path. We may rotate sp about s, sliding p along pt, until the link hits a vertex of P (or until p is on t, but we assume that we deal with minlink paths only, so in particular it is not possible to decrease the number of links by rotating a link of the path). Thus, $d = 2$ iff for any $s, t \in P$ there are vertices v and u of P, visible to s and t resp., such that the rays sv, tu intersect, at a point p, before exiting P. We look at the question from the "other" side, i.e., from the point of view of v and u: we go through every pair of vertices v, u and compute the set $P_{vu} \subset P \times P$ of pairs (s, t) such that v and u may support the links of a 2-link s-t path. If the union of the sets P_{vu}, for all pairs v, u, fully covers $P \times P$, then $d \leq 2$; otherwise, there exist points s, t with link distance at least 3. To compute P_{vu}, we first of all decompose P by extensions of VG edges: any point p within one cell of the decomposition sees the same set of vertices; in particular, we consider only the cells from which both v and u are seen. As p moves within such a cell σ, the rays pv, pu rotate around v, u resp., sweeping the locations for s, t resp. and thus creating $P^{\sigma}_{vu} \subseteq P_{vu}$ – the subset of P_{vu} corresponding to having p in σ. To implement the creation of P^{σ}_{vu}, triangulate σ: the subset of P^{σ}_{vu} obtained from p in one triangle, has constant description complexity, and P^{σ}_{vu} is the union of such subsets over all the triangles.

5 Two-point link distance queries

In this section we show how to compute the 2d LDM equivalence decomposition and the 4d link distance equivalence decomposition – the data structures for 2-point link distance queries, analogous to the structures of [9] for geodesic queries (see Section 3). Our data structures extend the data structure of [4] (for 2-point link distance queries in simple polygons) to polygonal domains with holes. The extension, allowing us to track combinatorial changes in LDM(s) for $s \in P$, is two-fold:

Same windows. In addition to decomposing the boundary of the polygon into "atomic segments" (as was done in [4]), we decompose the entire P into "atomic cells" by overlaying LDMs from extensions of VG edges – for all s in one cell σ of the decomposition, LDM(s) has the same set $W(\sigma)$ of windows. Moreover, the same subset $R(\sigma) \subseteq W(\sigma)$ of windows rotates as s moves within σ, while the other windows, $W(\sigma) \setminus R(\sigma)$, are not influenced by the location of s in the cell.

Same arrangement of windows. We track possible intersections among windows in LDM(s), which may change because the rotating windows sweep through the domain as s moves.

5.1 Extended visibility graph

For a line segment ℓ in P let $\overline{\ell}$ be the chord of P containing ℓ (i.e., $\overline{\ell}$ is ℓ extended maximally within P). Define the Extended Visibility Graph EVG $= \{\overline{uv} : uv \text{ is an edge of VG}\}$ as the set of chords of P obtained by maximally extending every edge of VG (we remind that edges of P are also edges of VG).

5.2 Re[de]fining LDM

Recall [25, Section 8] (see also Section 2.3) that an LDM is defined by an arrangement of windows together with edges of P. Recall also that there is no requirement that the cells of LDM are maximal (i.e., that the link distance necessarily changes as you move between cells) – the only requirement is that the distance never changes within a cell. Thus LDM(s), for some point $s \in P$, remains LDM(s) even if one subdivides its cells into smaller pieces.

For the rest of the paper, we redefine LDM to include all chords of EVG. That is, we refine LDM by chords of EVG (but we continue to call the result LDM). In other words, we assume that EVG chords are windows of LDM(s) for any s: we thus use the term "windows" to refer both to "original" windows of LDM (as defined in [25]) and chords of EVG.

5.3 Static and rotating windows

Every window w necessarily goes through a vertex of P (Fig. 4). If the chord $\overline{w}$ goes through another vertex ($\overline{w} \in$ EVG), we call w a *pinned window* – such windows do not change as s moves locally; in particular, chords of EVG are pinned windows. Otherwise we call w a *bash window*, and the endpoints of $\overline{w}$ – *bash points*. A bash point lies in the interior of an edge of P; we call such an edge a *bash wall*. (The term "pinned" is borrowed from [4] where analogous edges were called "(rotationally) pinned"; the term "bash" is borrowed from [22] where path vertices lying in the interior of polygon edges were called "bash points" and the edges were called "bash walls".)

As s moves slightly, pinned windows of LDM(s) do not change. A bash window w may remain still or may rotate, depending on whether there is a pinned window appearing on the greedy path from s to w (recall from Section 2.2 that a greedy path is the minlink path following the windows):

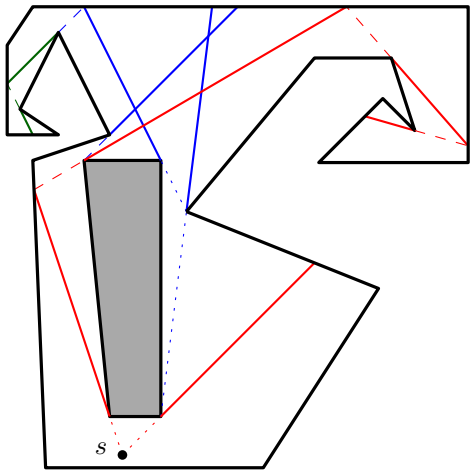


Figure 4 Pinned windows (those not aligned with edges of P) are blue (the aligned VG edges are dotted blue), static bash windows are green, rotating windows are red. For clarity, some windows have been omitted from the image.

Static bash windows. Any bash window w appearing after a pinned window w' does not move with s because w is a window in $\mathrm{LDM}(\overline{w'})$ and w' does not move with s.

Rotating bash windows (bash-bash windows). If all windows preceding w are bash windows, then they all rotate around vertices of P, with the bash points (which are the bend points of the greedy path from s to w) sliding along the bash walls; we call such windows *bash-bash windows* and the greedy path from s to w a *bash path*. (Refer to Fig. 4 – as s moves left, all windows rotate, so they are bash-bash windows: in fact, it is exactly bashing that leads to high bit complexity of minlink path vertices and that makes studying minlink paths challenging, as the bash points do not lie on any predefined structure which could be computed from P – differently from, e.g., geodesic paths which follow a VG.)

Overall, we obtain the classification of LDM windows into static and rotating. The former may be EVG chords or static bash windows; the latter are bash-bash windows.

5.4 LDMs overlay

We build LDMs from all chords of the EVG; let W_{EVG} denote the set of all windows in these LDMs. The first step in constructing our data structures is computing the overlay $\mathcal{O}$ of the windows in W_{EVG}. That is, we compute the overlay of LDMs from all chords of EVG – this is the same as [4] did, but we build the full overlay of the LDMs, while [4] considered only the interaction of W_{EVG} with the boundary of P.

Consider $\mathrm{LDM}(s)$ for a point s in P. As s moves, the bash-bash windows of $\mathrm{LDM}(s)$ rotate and the map may change combinatorially when either

Simple case. A window hits a vertex of P (aligning itself with a VG edge) when s crosses an edge of $\mathcal{O}$, or

New case. The arrangement of the windows changes because 3 windows pass through a common point (out of the 3 windows, at least one must be a rotating window).

We called the first event "simple" because this is the only thing that may happen in a simple polygon (where windows are pairwise disjoint). As we will show in Section 5.10, to

account for new events one needs to build LDMs from intersection points of windows in W_{EVG}, as well as to do some additional, less straightforward computations described later.

An important connection between the overlay $\mathcal{O}$ and static windows in LDMs is that all possible static windows in an LDM from any point of P are known in advance:

▶ **Observation 2.** *Any static window in any LDM belongs to W_{EVG} (i.e, $\forall s \in P$, if w is a static window in LDM(s), then $w \in W_{\mathrm{EVG}}$).*

Proof. If w is a chord of EVG, the observation is trivially true: w is a window in LDM(w). If w is a static bash window, then w is a window in LDM($\overline{w'}$) for some pinned window w', meaning that w' is aligned with a VG edge, implying that $\overline{w'}$ is a chord of EVG. ◀

Note that we do not claim that any window from W_{EVG} is necessarily a window in any LDM; we only claim that W_{EVG} is a superset of static windows in any LDM.

5.5 Atomic segments and projection functions

Edges of $\mathcal{O}$ (windows of W_{EVG}) split the edges of P into segments called *atomic* in [4] (note that vertices of P are endpoints of atomic segments too). Since any atomic segment a fully belongs to a single cell in $\mathcal{O}$, for all points $s \in a$, LDM(s) has combinatorially the same bash paths. Moreover, the exact location of the bash points (vertices of the paths) and hence the directions of the rotating bash-bash windows in LDM(s) depend on the location of s in a via "projection" functions worked out in [4] which further refers to [2] for details of computing the functions: any projection function is the ratio of 2 linear functions of s, whose parameters can be calculated window-by-window in constant time per window.

5.6 Parametric maps (for simple cases)

If the new cases are ignored, then LDMs (with the corresponding projection functions) built for all cells of $\mathcal{O}$ define the 2d LDM equivalence decomposition. The data structure can be used to answer 2-point link distance queries in the same way as the 2d SPM equivalence decomposition [9, Section 4.2] answers geodesic distance queries: given query points s and t, first s is located in a cell σ of $\mathcal{O}$ and then t is located in the parametric LDM(s).

In fact, since LDM edges (the windows) are straight line segments (not hyperbolic arcs as edges of SPM), one possibility for locating t in the parametric LDM(s) is to use the monotone subdivision method of [12]. Indeed, each comparison needed to answer the point location query for t using [12], can be done in constant time even when the edges of the monotone subdivision $\mathcal{M}$ are constant-algebraic-complexity functions of s, as long as $\mathcal{M}$ remains the same topologically (i.e., as long as the horizontal ordering of its vertices remains fixed). Since we know, via the projection functions, how LDM(s) changes, we also know how $\mathcal{M}$ changes with s. In particular, we can compute the locus $S \in \sigma$ of positions for s at which 2 vertices of $\mathcal{M}$ have the same x-coordinates: for any 2 vertices u, v of $\mathcal{M}$, this amounts to solving for s the constant-degree polynomial equation $x_v(s) = x_u(s)$ where x_u, x_v are the abscissae (i.e. the x-coordinates) of u, v. After refining σ by such sets S for all pairs of vertices of $\mathcal{M}$, we obtain the required subdivision in which parametric point location query, and hence the link distance query, can be answered in time logarithmic in the complexity of $\mathcal{M}$. Since the complexity of LDM and hence the complexities of $\mathcal{O}$ and $\mathcal{M}$ are polynomial in n, the query time is $O(\log n)$.

5.7 4d link distance equivalence decomposition (for simple cases)

We can also build the decomposition $\mathcal{O} \times \mathcal{O}$ of the 4d $P \times P$ space. That is, for any point $(s,t) \in P^2$ within one cell ρ of $\mathcal{O}^2$, we have $s \in \sigma, t \in \tau$ where σ, τ are some cells of $\mathcal{O}$. Via the projection functions, we know how each window w of LDM(s) depends on s. In 4d, the union $\bigcup_{s\in\sigma} w(s)$ of the windows w as s varies over σ defines a surface of constant algebraic degree, which we call a *curtain* (the surface is swept by the rotating w: as s moves along a ray from a vertex v of P, the window stays the same; the window rotates as vs rotates about v). We build the arrangement of the curtains in ρ, and repeat it for all cells ρ of $\mathcal{O}^2$. In the obtained decomposition (cells of $\mathcal{O}^2$, decomposed by the curtains) the link distance between any pair of points (s, t) from one cell is the same: the link distance between s and t may change only if t crosses a window of LDM(s), which happens only if t (the last two coordinates of point (s, t) in 4d) crosses a curtain. To achieve $O(\log n)$ query time, preprocess the decomposition for point location (e.g., in the same way as was done for geodesic queries in [9, Theorem 3.1]).

5.8 Triple points

What remains is to account for the new cases (windows arrangement changes due to 3 windows passing through a common point). In Section 5.10 we give an algorithm to compute the locations $\mathcal{S} \subset P$ for s such that 3 windows in LDM(s) may intersect in a common point t ($\mathcal{S}$ consists of curves of constant algebraic degree). The set $\mathcal{S}$ is overlaid with $\mathcal{O}$. By construction, for all points s in one cell of the obtained 2d overlay $\mathcal{O}^*$, LDM(s) is the same combinatorially: it has the same windows and they form the same arrangement (before the arrangement changes, 3 windows must pass through a common point, at which moment s is in $\mathcal{S}$). Our final data structure, the (full) 2d LDM equivalence decomposition ("full" in the sense that it accounts for both simple and new cases) is built from $\mathcal{O}^*$ in the same way as the data structure for handling simple cases was built from $\mathcal{O}$ (Section 5.6).

As a by-product of computing $\mathcal{S}$, we also identify the corresponding locations for the "triple points" t, thus obtaining the (super)set $\mathbb{S} \subset P \times P$ of pairs (s, t) of potential triple points. Then, (super)set $\mathbb{S}$ is overlaid with $\mathcal{O}^2$ subdivided by the curtains (Section 5.7). For all (s,t) in one cell of the obtained 4d overlay $\hat{\mathcal{O}}$, the s-t distance is the same. Preprocessing $\hat{\mathcal{O}}$ for point location, we obtain our other data structure – the (full) 4d link distance equivalence decomposition ("full" in the sense that it accounts for both simple and new cases).

5.9 Tracking bash-bash windows

To find the set $\mathcal{S}$ of sources potentially having triple points in their LDMs, we build yet another decomposition and do some preparation work. In [25], the combinatorial type of a window w was defined as the pair (v,e) where v is the vertex of P at which w begins and e is the edge on which the window ends. We refine the definition by saying that the window's combinatorial type is a pair (v,a) where a is the atomic segment on which w ends. We decompose P by rays from every atomic segment endpoint through every vertex visible to the endpoint (recall that vertices of P are also endpoints of atomic segments). For any point s in a cell of this decomposition $\mathcal{D}$ and every vertex v visible to s, the atomic segment a hit by the ray sv is the same; let $s' = s'(s) \in a$ be the point where the ray hits the segment (we write $s' = s'(s)$ to emphasize its dependence on s). Since a fully belongs to a single cell of the overlay $\mathcal{O}$, the map LDM(s') from any point $s' \in a$ has combinatorially the same bash-bash windows; moreover, the dependence of these windows on s' (and hence on s) is known via the projection functions. We thus write $w = w(s)$ for a window w in LDM(s').

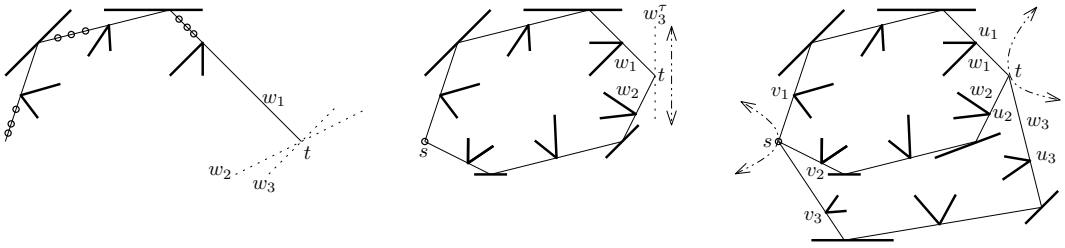


Figure 5 P is bold, static windows from W_{EVG} are dotted, bash paths are solid. Left: LDM(s) may change combinatorially when s crosses a bash-bash window of LDM(t) where $t = w_1 \cap w_2$ for some static windows $w_1, w_2 \in W_{\mathrm{EVG}}$ of LDM(s); some possible locations for such s are shown with hollow circles (if s is moved slightly off the windows, w_1 stops going through t). Middle: LDM(s) may change when there are two bash paths between s and $t \in w_3$. Right: Each of s, t moves on a curve so as to remain connected by 3 bash paths and be triple points in LDMs from each other.

5.10 Intersecting 3 windows

We are now ready to compute the (super)set $\mathcal{S}$ of points s for which LDM(s) may have a triple point t; the corresponding locations for t (i.e., the set $\mathbb{S} \subset P \times P$ for the pairs (s, t) defining the 4d link distance equivalence decomposition) are computed as well. We emphasize that we do not claim that LDM(s) has a triple point for every s in $\mathcal{S}$ (or that $\forall (s,t) \in \mathbb{S}$ LDM(s) has a triple point at t); we only claim that $\mathcal{S}$ is sufficiently rich to "catch" all possible (sources of) maps with triple points (and that $\mathbb{S}$ includes all pairs of triple points).

Let w_1, w_2, w_3 be windows of LDM(s) that intersect at t, of which at least 1 window rotates (so that LDM(s) changes combinatorially when the 3 windows intersect at t). Recall from Observation 2 that the (super)set W_{EVG} of possible static windows in LDM(s) does not depend on s. We make 3 different guesses on how many of the 3 windows w_1, w_2, w_3 are rotating (1, 2, or all 3), and do different things for each of the guesses (Fig. 5):

- Assuming only w_1 is rotating, LDM(s) changes when w_1 passes through $t = w_2 \cap w_3$ (the intersection point of two static windows w_2, w_3). At this point there is a bash path between s and t, implying that s is on a bash-bash window of LDM(t). We thus build LDM(t) from every intersection point t of two (potential) static windows $w_2, w_3 \in W_{\mathrm{EVG}}$ and add to $\mathcal{S}$ all bash-bash windows of LDMs; the 4d set $\mathbb{S}$ is correspondingly appended with the Cartesian product of t and all bash-bash windows of LDM(t).
- Assuming 2 windows (say, w_1, w_2) are rotating, LDM(s) changes when their intersection point t passes through w_3. At this point there are two bash paths between s and t (in one path $t \in w_1$, in the other $t \in w_2$). That is, s is the intersection point of two rotating windows in LDM(t) (one window belonging to the bash path from t that starts from following w_1; the other – following w_2). We thus take every window $w_3 \in W_{\mathrm{EVG}}$ as the potential static window in LDM(s) and intersect it with every cell τ of $\mathcal{O}$; let w_3^τ denote part of w_3 inside τ. For all points $t \in w_3^\tau$, LDM(t) has the same set of windows, and in particular, the same set of bash-bash windows; each window is a known function of t (via the projection functions). For every pair of the bash-bash windows (with the same link distance from t) in LDM(t) we add to $\mathcal{S}$ the curve traced by their intersection as t varies along w_3^τ; the 4d set $\mathbb{S}$ is correspondingly appended with the Cartesian product of w_3^τ and the intersection points for $t \in w_3^\tau$.
- Assuming all 3 windows are rotating, we go through all pairs of cells σ, τ in $\mathcal{D}$, guessing that $s \in \sigma, t \in \tau$. In addition, we go through all triples v_1, v_2, v_3 of vertices visible from s and all triples u_1, u_2, u_3 of vertices visible to t, guessing that the vertices support

the first and the last windows resp. of the 3 bash paths (ending with the windows w_1, w_2, w_3) between s and t. Since vertices of P are endpoints of atomic segments, the decomposition $\mathcal{D}$ includes chords of EVG, implying that the set of vertices visible to any point in a cell of $\mathcal{D}$ is the same: it is thus valid to speak about a triple of vertices visible to $s \in \sigma$ or to $t \in \tau$ without specifying the exact locations of s and t in the cells.
Using the projection functions, we know whether for some number k of links there indeed exist 3 length-k bash paths from s, with the first links of the paths supported by v_1, v_2, v_3 and last links supported by u_1, u_2, u_3. If yes, let $w_1(s), w_2(s), w_3(s)$ be these last links (windows of LDM(s)). For t to be a triple point, the system

$$\begin{cases} t \in w_1(s) \\ t \in w_2(s) \\ t \in w_3(s) \end{cases} \tag{1}$$

of 3 equations (each involving the projection function) with 4 unknowns (the coordinates of s and t) must be satisfied (for $s \in \sigma, t \in \tau$). We solve the system and obtain the decompositions both for the 2d LDM equivalence decomposition (the set $\mathcal{S} \subseteq P$ such that LDM(s) may change combinatorially only when s is crossing $\mathcal{S}$, see Fig. 5 for an example of such a combinatorial change) and for the 4d link distance equivalence decomposition (the surface $\mathbb{S} \subseteq P^2$ such that the link distance between s and t may change only when (s, t) is crossing $\mathbb{S}$). That is, the 4d set $\mathbb{S}$ consists of pairs (s, t) satisfying (1), while for the 2d set $\mathcal{S}$ we take only the first two coordinates of the 4d pairs (s, t) (i.e., the coordinates for s; the knowledge of corresponding locations for t is ignored).

5.11 Putting things together

We summarize the steps described above, needed to build our data structures:

1. Build VG and EVG (Section 5.1).
2. Designate all chords of EVG as (static) windows in any LDM ever built (Section 5.2).
3. Compute the set W_{EVG} of windows in LDMs from all chords of EVG (any static window in any LDM is part of W_{EVG}) and compute the overlay $\mathcal{O}$ of the LDMs (Section 5.4).
4. Identify atomic segments by intersecting $\mathcal{O}$ with the boundary of P (Section 5.5).
5. For each atomic segment a, compute the projection functions specifying the windows in LDM(s) $\forall s \in a$ (Section 5.5). The functions are used to identify the 2d set $\mathcal{S} \subset P$ of potential triple points and the 4d set $\mathbb{S} \subset P^2$ of potential pairs of triple points of LDMs, as well as to answer link distance queries with parametric LDMs.
6. Build the decomposition $\mathcal{D}$ of P by using rays from atomic segments though vertices of P (Section 5.9). $\mathcal{D}$ is an auxiliary structure: we do not overlay $\mathcal{D}$ with any other decomposition and do not use it in our final data structures; $\mathcal{D}$ is used only to compute the 2d set $\mathcal{S} \subset P$ of potential triple points and the 4d set $\mathbb{S} \subset P^2$ of potential pairs of triple points – these sets will be used in our data structures, refining $\mathcal{O}$ and $\mathcal{O}^2$ respectively.
7. Compute the sets $\mathcal{S}$ and $\mathbb{S}$ for triple points: build LDMs from intersection points of windows from W_{EVG}, track intersections of windows in LDM(t) as t varies along each window from W_{EVG}, go through all triples of windows in pairs of cells of $\mathcal{D}$ (Section 5.10).
8. Build the overlay $\mathcal{O}^*$ of $\mathcal{O}$ with $\mathcal{S}$ (Section 5.8). To obtain the 2d LDM equivalence decomposition data structure, preprocess $\mathcal{O}^*$ for parametric point location, e.g., by further decomposing the cells of $\mathcal{O}^*$ so that for all s in one cell of the refined decomposition, the ordering of x-coordinates of vertices of LDM(s) is the same (as done in Section 5.6 for $\mathcal{O}$).

9. Decompose $\mathcal{O} \times \mathcal{O}$ by curtains (Section 5.7) and overlay the decomposition with $\mathbb{S}$, getting 4d decomposition $\hat{\mathcal{O}}$ (Section 5.8). To obtain the 4d distance link equivalence data structure, preprocess $\hat{\mathcal{O}}$ for point location queries.

▶ **Theorem 3.** *2-point link distance queries in P can be answered in $O(\log n)$ time after polynomial-time preprocessing. Link diameter and radius of P can be found in poly time.*

6 Conclusions

We presented data structures for 2-point link distance queries in polygonal domains; we also showed how to find the link diameter and radius/center of a domain. We were only after polynomiality and do not claim any practical significance (but neither do the results on geodesic queries, diameter and radius [6,9,31], having complexities in $\Omega(n^7)$); in practice, fast solutions with small additive errors [4, Section 6] may be used. One obvious open question is improving efficiency of our algorithms. The only lower bound for the diameter/radius we can think of is 3SUM-hardness (following from simple constructions like in [24], reducing from GeomBase [14]). As far as 2-point link distance queries go, in polygonal domains with holes, even visibility queries (Do s and t see each other, i.e., is the link distance between them equal to 1?) are quite challenging [8]. It may also be interesting to give a simple(r) algorithm for finding the link radius/center, which does not resort to MetaTheorem 1.

References

1 Mikkel Abrahamsen, Joakim Blikstad, André Nusser, and Hanwen Zhang. Minimum star partitions of simple polygons in polynomial time. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 904–910, 2024. doi:10.1145/3618260.3649756.

2 Alok Aggarwal, Heather Booth, Joseph O'Rourke, Subhash Suri, and Chee K Yap. Finding minimal convex nested polygons. *Information and Computation*, 83(1):98–110, 1989. doi:10.1016/0890-5401(89)90049-7.

3 Hee-Kap Ahn, Luis Barba, Prosenjit Bose, Jean-Lou De Carufel, Matias Korman, and Eunjin Oh. A linear-time algorithm for the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 56(4):836–859, 2016. doi:10.1007/s00454-016-9796-0.

4 Esther M Arkin, Joseph SB Mitchell, and Subhash Suri. Logarithmic-time link path queries in a simple polygon. *International Journal of Computational Geometry & Applications*, 5(04):369–395, 1995. doi:10.1142/S0218195995000234.

5 Elena Arseneva, Man-Kwun Chiu, Matias Korman, Aleksandar Markovic, Yoshio Okamoto, Aurélien Ooms, André Van Renssen, and Marcel Roeloffzen. Rectilinear link diameter and radius in a rectilinear polygonal domain. *Computational Geometry*, 92:101685, 2021. doi:10.1016/j.comgeo.2020.101685.

6 Sang Won Bae, Matias Korman, and Yoshio Okamoto. The geodesic diameter of polygonal domains. *Discrete & Computational Geometry*, 50(2):306–329, 2013. doi:10.1007/s00454-013-9527-8.

7 Sang Won Bae, Matias Korman, and Yoshio Okamoto. Computing the geodesic centers of a polygonal domain. *Computational Geometry*, 77:3–9, 2019. doi:10.1016/j.comgeo.2015.10.009.

8 Danny Z Chen and Haitao Wang. Visibility and ray shooting queries in polygonal domains. *Computational Geometry*, 48(2):31–41, 2015. doi:10.1016/j.comgeo.2014.08.003.

9 Yi-Jen Chiang and Joseph SB Mitchell. Two-point Euclidean shortest path queries in the plane. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 215–224. Citeseer, 1999. doi:10.5555/314500.314560.

10 Sarita de Berg, Tillmann Miltzow, and Frank Staals. Towards space efficient two-point shortest path queries in a polygonal domain. In *40th International Symposium on Computational Geometry (SoCG 2024)*, volume 293, pages 17:1–17:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SoCG.2024.17.

11 Hristo N Djidjev, Andrzej Lingas, and Jörg-Rüdiger Sack. An $O(n \log n)$ algorithm for computing the link center of a simple polygon. *Discrete & Computational Geometry*, 8(2):131–152, 1992. doi:10.1007/BF02293040.

12 Herbert Edelsbrunner, Leonidas J Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986. doi:10.1137/0215023.

13 Stephan J Eidenbenz and Peter Widmayer. An approximation algorithm for minimum convex cover with logarithmic performance guarantee. *SIAM Journal on Computing*, 32(3):654–670, 2003. doi:10.1137/S0097539702405139.

14 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.

15 Subir Kumar Ghosh. Computing the visibility polygon from a convex set and related problems. *Journal of Algorithms*, 12(1):75–95, 1991. doi:10.1016/0196-6774(91)90024-S.

16 Leonidas J Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. In *Proceedings of the third annual symposium on Computational geometry*, pages 50–63, 1987. doi:10.1145/41958.41964.

17 John Hershberger and Jack Snoeyink. Computing minimum length paths of a given homotopy class. *Computational geometry*, 4(2):63–97, 1994. doi:10.1016/0925-7721(94)90010-8.

18 John Hershberger and Subhash Suri. Matrix searching with the shortest-path metric. *SIAM Journal on Computing*, 26(6):1612–1634, 1997. doi:10.1137/S0097539793253577.

19 Simon Kahan and Jack Snoeyink. On the bit complexity of minimum link paths: Superquadratic algorithms for problems solvable in linear time. *Computational Geometry*, 12(1):33–44, 1999. doi:10.1016/S0925-7721(98)00041-8.

20 Yan Ke. An efficient algorithm for link-distance problems. In *Proceedings of the fifth annual symposium on Computational geometry*, pages 69–78, 1989. doi:10.1145/73833.73841.

21 Irina Kostitsyna, Maarten Löffler, Valentin Polishchuk, and Frank Staals. On the complexity of minimum-link path problems. *Journal of Computational Geometry*, 8(2):80–108, 2017. Special issue on SoCG'16. doi:10.20382/jocg.v8i2a5.

22 Joseph SB Mitchell, C Piatko, and Esther M Arkin. Computing a shortest k-link path in a polygon. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 573–582. IEEE, 1992. doi:10.1109/sfcs.1992.267794.

23 Jörg-Rüdiger Sack and Jorge Urrutia. *Handbook of computational geometry*. Elsevier, 1999.

24 Joseph SB Mitchell, Valentin Polishchuk, and Mikko Sysikaski. Minimum-link paths revisited. *Computational Geometry*, 47(6):651–667, 2014. Special issue on EuroCG'11. doi:10.1016/J.COMGEO.2013.12.005.

25 Joseph SB Mitchell, Günter Rote, and Gerhard J Woeginger. Minimum-link paths among obstacles in the plane. *Algorithmica*, 8(1):431–459, 1992. doi:10.1007/bf01758855.

26 Subhash Suri. A linear time algorithm for minimum link paths inside a simple polygon. *Computer Vision, Graphics, and Image Processing*, 35(1):99–110, 1986. doi:10.1016/0734-189x(86)90070-8.

27 Subhash Suri. Computing geodesic furthest neighbors in simple polygons. *Journal of Computer and System Sciences*, 39(2):220–235, 1989. doi:10.1016/0022-0000(89)90045-7.

28 Subhash Suri. On some link distance problems in a simple polygon. *IEEE transactions on Robotics and Automation*, 6(1):108–113, 1990. doi:10.1109/70.88124.

29 Subhash Suri and Joseph O'Rourke. Worst-case optimal algorithms for constructing visibility polygons with holes. In *Proceedings of the second annual symposium on Computational geometry*, pages 14–23, 1986. doi:10.1145/10515.10517.

30 Csaba D Toth, Joseph O'Rourke, and Jacob E Goodman. *Handbook of discrete and computational geometry*. CRC press, 2017. doi:10.1201/9781315119601.

31 Haitao Wang. On the geodesic centers of polygonal domains. *J. Comput. Geom.*, 9(1):131–190, 2018. doi:10.20382/jocg.v9i1a5.

32 Haitao Wang. A divide-and-conquer algorithm for two-point L_1 shortest path queries in polygonal domains. *Journal of Computational Geometry*, 11(1):235–282, 2020. doi:10.20382/jocg.v11i1a10.

33 Haitao Wang. A new algorithm for Euclidean shortest paths in the plane. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 975–988, 2021. doi:10.1145/3406325.3451037.

34 Haitao Wang. Shortest paths among obstacles in the plane revisited. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms*, pages 810–821, 2021. doi:10.1137/1.9781611976465.51.

Succinct Data Structures for Chordal Graph with Bounded Leafage or Vertex Leafage

Meng He
Faculty of Computer Science, Dalhousie University, Halifax, Canada

Kaiyu Wu
Faculty of Computer Science, Dalhousie University, Halifax, Canada

Abstract

We improve the recent succinct data structure result of Balakrishnan et al. for chordal graphs with bounded vertex leafage (SWAT 2024). A chordal graph is a widely studied graph class which can be characterized as the intersection graph of subtrees of a host tree, denoted as a tree representation of the chordal graph. The vertex leafage and leafage parameters of a chordal graph deal with the existence of a tree representation with a bounded number of leaves in either the subtrees representing the vertices or the host tree itself.

We simplify the lower bound proof of Balakrishnan et al. which applied to only chordal graphs with bounded vertex leafage, and extend it to a lower bound proof for chordal graphs with bounded leafage as well. For both classes of graphs, the information-theoretic lower bound we (re-)obtain for $k = o(n)$ is $(k-1)n \log n - kn \log k - o(kn \log n)$ bits, where the leafage or vertex leafage of the graph is at most $k = o(n)$. We further extend the range of the parameter k to $\Theta(n)$ as well.

Then we give a succinct data structure using $(k-1)n \log(n/k) + o(kn \log n)$ bits to answer `adjacent` queries, which test the adjacency between pairs of vertices, in $O(\frac{\log k}{\log \log n} + 1)$ time compared to the $O(k \log n)$ time of the data structure of Balakrishnan et al. For the `neighborhood` query which lists the neighbours of a given vertex, our query time is $O(\frac{\log n}{\log \log n})$ per neighbour compared to $O(k^2 \log n)$ per neighbour.

We also extend the data structure ideas to obtain a succinct data structure for chordal graphs with bounded leafage k, answering an open question of Balakrishnan et al. Our succinct data structure, which uses $(k-1)n \log(n/k) + o(kn \log n)$ bits, has query time $O(1)$ for the `adjacent` query and $O(1)$ per neighbour for the `neighborhood` query. Using slightly more space (an additional $(1+\varepsilon)n \log n$ bits for any $\varepsilon > 0$) allows `distance` queries, which compute the number of edges in the shortest path between two given vertices, to be answered in $O(1)$ time as well.

2012 ACM Subject Classification Theory of computation → Data compression; Theory of computation → Data structures design and analysis

Keywords and phrases Chordal Graph, Leafage, Vertex Leafage, Succinct Data Structure

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.35

Funding This work is supported by NSERC.

1 Introduction

Chordal graphs and related graph classes are one of the most well studied classes of graphs [13, 10, 20, 18, 23, 11, 29, 6]. One of the first instances where chordal graphs are encountered is in the study of Gaussian elimination of sparse matrices [29], which leads to a characterization of chordal graphs based on an elimination ordering. Another characterization is that there are no induced cycles of length four or more [10]. A third characterization, which is more suitable towards data structures, is the characterization as an intersection graph. For each chordal graph G, there exists a host tree T (which is unrooted) and n subtrees $T(v)$ of T (one for each vertex), such that two vertices u and v are adjacent in G if and only if $T(u)$ and $T(v)$ intersect at some node of T [18].

© Meng He and Kaiyu Wu;
licensed under Creative Commons License CC-BY 4.0

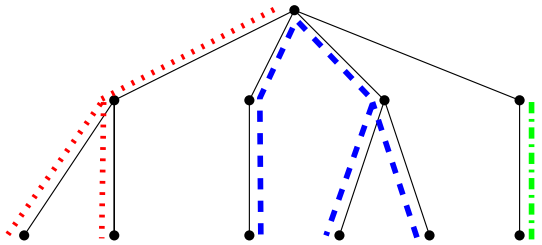


Figure 1 A tree representation with the host tree in black solid lines. The host tree has 6 leaves, so this chordal graph has leafage at most 6. 3 subtrees are depicted, a red dotted subtree, a blue dashed and a green dotted-dashed. The red subtree has 3 leaves, the blue subtree also has 3 leaves and green subtree has 2. So this chordal graph has vertex leafage at most 3.

It is this last characterization of chordal graphs that allows us to (easily) generate many interesting subclasses of chordal graphs, by imposing conditions[1] on either the subtree $T(v)$ or the host tree T. For example, if we impose the condition that the host tree is a path (which also imposes the condition that the subtrees themselves are paths), then the subclass of graphs we generate is the class of *interval graphs* [7]. If we instead only impose that the subtrees themselves are paths, then we generate the class of *path graphs* [17]. If we further impose conditions on these path subtrees, such as selecting a root in the host tree and insisting that each path is a subpath of a root-to-leaf path, then we obtain a RDV (rooted, directed, vertex) graph [24].

If we impose the condition that the host tree has at most k leaves (as the tree is unrooted, a leaf is a degree 1 node), then this parameter k is known as the *leafage* of the chordal graph [23]. We may alternatively impose that only the subtrees $T(v)$ rather than the host tree have at most k leaves each (i.e at most k degree 1 nodes). Then this parameter is known as the *vertex leafage* [11]. A chordal graph with bounded leafage k has a tree representation where the host tree has at most k leaves, and a chordal graph with bounded vertex leafage k has a tree representation where all subtrees have at most k leaves. For $k = 2$, this corresponds to interval graphs and path graphs, respectively, whereas for $k = n$, we obtain all chordal graphs. Thus, the leafage parameter measures how close a chordal graph is to an interval graph, and the vertex leafage parameter measures how close a chordal graph is to a path graph. See Figure 1.

We observe that if the host tree has at most k leaves, then all subtrees would as well. Therefore, chordal graphs with leafage k is a subset of chordal graphs with vertex leafage k. Consequently, any data structure for vertex leafage applies to leafage, and any lower bound for leafage applies to vertex leafage.

An enumeration of chordal graphs and path graphs gives lower bounds for any data structure which is able to compute adjacency, degree and neighbourhood queries. A chordal graph requires at least $n^2/4 - o(n^2)$ bits [25, 31] while both interval graphs and path graphs ([9] plus the above observation) requires at least $n \log n - O(n)$ bits [2], which gives some bounds for the extreme values of the leafage parameter.

In this paper, we study (static) data structures for (unweighted) chordal graphs parametrized by leafage and vertex leafage parameters, with an emphasize on space usage. Our aim is to store a chordal graph parametrized by either the leafage or vertex leafage parameters,

[1] This is not the only way to characterize these subclasses of graphs. For instance, interval graphs can also be characterized by being chordal and having no asteroid triples.

[2] We will use log to denote $\log_2$

using information-theoretic minimal space within an additive lower-order term, while supporting the following queries efficiently: `adjacent`(u, v), which tests the adjacency of the vertices u and v, `neighborhood`(v), which returns a list of the neighbours of the vertex v, and `distance`(u, v), which returns the distance (the number of edges on the shortest path) between u and v. We aim to improve the work of Balakrishnan et al. [3] by eliminating their linear dependence on the vertex leafage parameter in the `adjacent` query, and the quadratic dependence in the `neighborhood` query for chordal graphs with bounded vertex leafage. Furthermore, we investigate an open question of theirs which asks for a simple representation of chordal graphs with bounded leafage that supports faster queries; they only specifically considered bounded vertex leafage, and provided lower and upper bounds accordingly.

1.1 Related Work

The most pertinent work is the recent data structure of Balakrishnan et al. [3] for chordal graphs with bounded vertex leafage. They showed an information-theoretic lower bound of $(k-1)n\log n - kn\log k - O(\log n)$ bits, for parameter range $k = n^{o(1)}$. For the upper bound, they gave a data structure occupying $(k-1)n\log n + o(kn\log n)$ bits which answers `adjacent` queries in $O(k\log n)$ time. For `neighborhood` queries, they require an additional $2n\log n$ bits, with query time $O(k^2\log n \cdot d_v + \log^2 n)$ where d_v is the degree of the query vertex.

There are many succinct data structures for various classes of graphs related to chordal graphs. Munro and Wu [25] gave a succinct data structure for general chordal graphs, occupying $\frac{1}{4}n^2 + o(n^2)$ bits, with query times $O(f(n))$ for `adjacent`, $O(f^2(n))$ for `neighborhood` and $O(nf(n))$ for `distance`, for any $f(n) = \omega(1)$. This was improved by He et al. [22] to remove a factor of $f(n)$ in the time bound for queries, so that `adjacent` has $O(1)$ time, `neighborhood` has $O(f(n))$ time, and `distance` has $O(n)$ time.

The two graph classes with leafage and vertex leafage 2, interval graphs and path graphs, are studied by Acan et al. [1], He et al. [21], Balakrishnan et al. [4] and He et al. [22]. The result of these works is that interval graphs can be represented succinctly using $n\log n + o(n\log n)$ bits which supports `adjacent`, `neighborhood` [1] and `distance` [21] all in optimal time. For path graphs, a succinct $n\log n + o(n\log n)$ bit data structure supports `adjacent` in $O(\log n/\log\log n)$ time and `neighborhood` in the same time per neighbour. To achieve $O(1)$ time, they use $(3+\varepsilon)n\log n + o(n\log n)$ bits, for any constant $\varepsilon > 0$ [22].

Lastly, there are results for other related classes of intersection graphs (of various geometric and combinatorial objects), such as permutation graphs (line segments between two parallel lines) [30], graphs with small tree-width (subgraphs of chordal graphs with small maximal clique) [14], d-boxicity graphs (higher dimensional analog of interval graphs) [2].

1.2 Our Results

We give lower bounds and data structures for chordal graphs with bounded vertex leafage or bounded leafage. Our results for the former class of graphs extends the range of parameter values applicable in the lower bounds and improves the query times in the data structures, compared to the work of Balakrishnan et al. [3]. Our results for the later class of graphs answers an open question posed by them.

Lower Bounds. With respect to lower bounds for chordal graphs with bounded vertex leafage k, Balakrishnan et al. showed an information-theoretic lower bound of $(k-1)n\log n - kn\log k - O(n\log n)$ bits, applying to the parameter range $k = n^{o(1)}$. We extend the range

Table 1 Comparison of our data structure results for chordal graphs with bounded vertex leafage k with the results of Balakrishnan et al. [3]. The 4th column (*) is the extra space needed to support the `neighborhood` query. Time for `neighborhood` query is per neighbour returned.

Source	Space for Adjacency (bits)	Adjacency	*	Neighbour
[3]	$(k-1)n\log n + o(kn\log n)$	$O(k\log n)$	$2n\log n$	$O(k^2\log n)$
Thm 11	$(k-1)n\log(n/k) + o(kn\log n)$	$O(\frac{\log k}{\log\log n}+1)$	$kn\log n$	$O(\frac{\log n}{\log\log n})$
Thm 12	$(k-1)n\log n + o(kn\log n)$	$O(\frac{\log n}{\log\log n})$	$n\log n$	$O(\frac{\log n}{\log\log n})$

to $k = o(n)$ with the same lower bound, by using a simpler host tree, and using a simpler full colouring scheme rather than a partial colouring scheme. This further allows us to extend our construction to give a lower bound for $k = \Theta(n)$ as well, covering all values of k. In this range, the lower bound we obtain have the form $f(c)n^2 + o(n^2)$, where $k = cn$.

For chordal graphs with bounded leafage, we prove that they have the same lower bound as chordal graphs with bounded vertex leafage, namely $(k-1)n\log n - kn\log k - o(kn\log n)$ bits, when $k = o(n)$. Our construction also allows us to extend the parameter k to $k = \Theta(n)$. Again, the lower bound we obtain have the form $f(c)n^2 + o(n^2)$, where $k = cn$ (for a different function f). This is the first information-theoretic lower bound for this class of graphs.

Data Structures for Bounded Vertex Leafage. We give a succinct data structure occupying $(k-1)n\log(n/k) + o(kn\log n)$ bits compared to that of $(k-1)n\log n + o(kn\log n)$ bits of Balakrishnan et al. [3]. Thus our data structure is succinct for $k = o(n)$ rather than just $k = n^{o(1)}$. Furthermore, we achieve $O(\log k/\log\log n)$ query time for `adjacent` query compared to $O(k\log n)$ time for their data structure. For the `neighborhood` query, we use $O(\log n/\log\log n)$ time per neighbour rather than $O(k^2\log n)$ per neighbour. This is around a factor of k faster for `adjacent` and a factor of k^2 for `neighborhood`, which is significant, especially if k is large. A comparison can be seen in Table 1.

To do this, rather than breaking the subtree $T(v)$ into a set of paths, and storing them within a path graph data structure, we simply store the preorder numbers of the leaves of the tree within a dictionary. This allows us to both compress the space and forgo data structures which is needed to identify paths that come from the same subtree, which saves a factor of k in the time complexities. Next, by storing the leaves of the subtree together, we are able to identify a neighbour u of a vertex v using a single leaf of $T(u)$, rather than possibly returning every leaf of $T(u)$, saving the second factor of k in the `neighborhood` query.

Data Structures for Bounded Leafage. Using the additional structure that the host tree has at most k leaves, we construct the first succinct data structure for chordal graphs with bounded leafage k such that $k = o(n)$, using $(k-1)n\log(n/k) + o(kn\log n)$ bits of space. Our data structure supports `adjacent` in $O(1)$ time and `neighborhood` in $O(1)$ time per neighbour. Using an additional $(1+\varepsilon)n\log n$ bits (for any constant $\varepsilon > 0$)), we also support `distance` in $O(1)$ time. We do this by breaking the host tree into $k-1$ paths, and storing the leaf of the subtree $T(v)$ on each path (if one exists). This allows us to identify leaves of $T(v)$ by this path number, leading to more efficient accesses.

All our results are obtained in the word RAM model with $\Theta(\log n)$ bit words.

2 Preliminaries

In this section, we introduce the existing data structures that our solution use as building blocks. As we will be discussing both graphs and trees at the same time, we will use the term *vertex* to refer vertices in the graph and *node* for trees.

Sets of Integers. We begin by considering various methods of storing sets of non-negative integers (viewed in sorted order).

A key succinct data structure is the bitvector: a length n array of bits. A bitvector B supports three operations. **1)** $\texttt{access}(B, i)$ or simply $B[i]$: return the i-th bit in the bitvector. **2)** $\texttt{rank}_1(B, i)$: return the number of 1 bits in the prefix $B[1, i]$. Symmetrically for $\texttt{rank}_0$. **3)** $\texttt{select}_1(B, i)$: return the index of the i-th 1 bit in B. Symmetrically for $\texttt{select}_0$. We note that a bitvector can be viewed as storing the set of integers corresponding to the index of the 1 bits. Thus, the $\texttt{select}(i)$ for set of integers will return the i-th smallest integer. The bitvector we will use is the result of Clark and Munro:

▶ **Lemma 1** ([12]). *A bit vector of length n can be succinctly represented using $n + o(n)$ bits to support* $\texttt{rank}, \texttt{select}$ *and* $\texttt{access}$ *in $O(1)$ time.*

For a sparse set of integers, we will use the dictionary result of Raman et al. [28].

▶ **Lemma 2.** *A subset of n integers from $[0, \ldots, m-1]$ can be represented using $\lceil \log \binom{m}{n} \rceil + o(n) + O(\log \log m)$ bits, to support* $\texttt{select}$ *queries in $O(1)$ time.*

The predecessor and successor queries on a set S of integers are: **1)** $\texttt{pred}(k)$: which returns the largest element of S no greater than k, i.e. $\max\{s \leq k \mid s \in S\}$, and symmetrically, **2)** $\texttt{succ}(k)$: which returns the smallest element of S no less than than k. When the size of the integers are bounded (and allowing duplicates), we may employ the folklore solution, which stores the termwise differences of the sequence as unary in a bitvector.

▶ **Lemma 3.** *Let A be a multi-set of N non-negative integers bounded by M. Then A can be stored in $N + M + o(N + M)$ bits answer* $\texttt{select}, \texttt{pred}$*, and* $\texttt{succ}$ *queries in $O(1)$ time.*

Ordinal Trees. As we will be studying chordal graphs, using primarily the characterization that it is the intersection graph of subtrees in a tree, we will need a data structure storing said tree. Succinct ordinal tree data structures have a long line of study, with new queries being added as recently as He et al. [21]

▶ **Lemma 4** ([21]). *An ordinal tree can be represented succinctly in $2n + o(n)$ bits to support a wide variety of operations in $O(1)$ time. These operations include:* ***1)*** $\texttt{node_rank}_{\texttt{PRE}}(x)$*, which returns the index in the preorder traversal that we encounter the node x,* ***2)*** $\texttt{LCA}(x, y)$*, which returns the lowest common ancestor of nodes x and y,* ***3)*** $\texttt{parent}(x)$*, which returns the parent of a node x and* ***4)*** $\texttt{child}(x, i)$*, which returns the i-th child of a node x.*

We will refer to $\texttt{node_rank}_{\texttt{PRE}}$ of a node as its *preorder number*, and use it to identify nodes. Thus, when we refer to a node $x \in T$, x is treated as an integer, and we may write $x+1$ for the next node encountered in the preorder traversal of T. One application of ordinal trees is to range minimum/maximum queries, through a Cartesian tree. For an array $A[1, n]$ of integers, a *range minimum query* or RMQ takes two indices i, j with $i \leq j$ and returns the index of the minimum (or symmetrically, maximum) value in the subarray $A[i, j]$. We use the following result:

▶ **Lemma 5** ([15]). *There is a data structure for the RMQ problem that uses $2n + o(n)$ bits of space and answers queries in $O(1)$ query time.*

Orthogonal Range Searching. In the 2-dimensional *orthogonal range reporting* query, we are asked to preprocess a set P of n 2-dimensional points, so that given an axis aligned query rectangle $R = [x_1, x_2] \times [y_1, y_2]$, return the list of points lying in the rectangle, i.e $P \cap R$. Bose et al. [8] designed a succinct data structure for this problem when the points have integral coordinates in the $n \times n$ grid.

We will need the following two variants of this query, which deals with many points sharing the same y-coordinate. First is the *orthogonal range distinctness* query: Given an axis-aligned rectangle R, return the set of distinct y-coordinates $y^1, \ldots, y^p$ such that there exists a point $(-, y^i)$ that lies in the rectangle R. Second is the *orthogonal range distinct maximum/minimum* query: Given an axis-aligned rectangle R, and for each distinct y-coordinate in the set of points $P \cap R$, return the point (x^i, y^i) with the maximum/minimum x-coordinate among the points in the rectangle with y-coordinate y^i.

We may solve these two queries by a slight modification of the data structure of Bose et al. An inspection of their orthogonal range reporting query algorithm shows the following. Their algorithm identifies the set of y-coordinates each of which has a point in the given query rectangle. For each such y-coordinate identified, the set of points with that y-coordinate is represented in a bitvector, and the query rectangle defines a subrange of that bitvector. The algorithm traverses the bitvector by using `select` for each of the 1 bits, and then recovers the coordinates of the points represented. Thus, we may `select` for the first or the last 1 in the subrange of the bitvector defined by the query rectangle, which returns the minimum or maximum points.

▶ **Lemma 6.** *Let P be a set of n points in an $n \times n$ grid. P can be represented using $n \log n + o(n \log n)$ bits to support orthogonal range reporting in $O(k \log n / \log \log n)$ time, where k is the size of the output. It can further support orthogonal range distinctness, and orthogonal range distinct minimum/maximum queries in $O(k' \log n / \log \log n)$ time, where k' is the number of distinct y-coordinates among the k points in the rectangle.*

3 Lower Bound for Bounded Leafage and Vertex Leafage

In this section, we will re-derive the lower bound result of Balakrishnan et al. [3] for graphs with bounded vertex leafage k. We will then generalize the result for a larger set of values for k. We will also derive lower bound results for graphs with bounded leafage k, which was not considered by Balakrishnan et al.

Since chordal graphs with bounded leafage are also graphs with bounded vertex leafage, a lower bound for chordal graphs with bounded leafage also applies to chordal graphs with bounded vertex leafage. We will still give different constructions for these two classes of graphs, to emphasize how they can be extended when $k = \Theta(n)$.

3.1 Bounded Vertex Leafage

As in the lower bounds shown in Munro and Wu [25], Wormald [31] and Bender et al. [5], we will mainly look at *split graphs*. A split graph (similar to a bipartite graph) is a graph where the vertices are separated into two groups V_1 and V_2. V_1 is a complete graph, and V_2 is an independent set. Any edge between a vertex $v_1 \in V_1$ and $v_2 \in V_2$ is allowed. It is easy to see that any split graph is a chordal graph by constructing a tree representation as follows: the host tree (a star) T will consist of 1 node R as the root and $|V_2|$ children of the root as leaves, one for each vertex of V_2. Let $x_1, \ldots, x_{|V_2|}$ denote the leaves corresponding to the vertices $v_1, \ldots, v_{|V_2|} \in V_2$. For each vertex v_i of V_2, the subtree $T(v_i)$ that corresponds to it is the

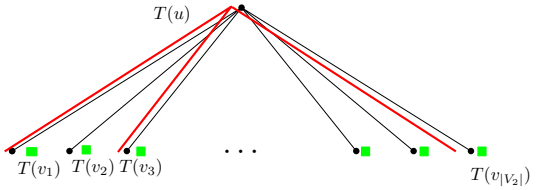


Figure 2 Tree representation of a split graph. For each vertex $v_i \in V_2$ its subtree $T(v_i)$ is a single node at one of the leaves (depicted by the green squares). A vertex $u \in V_1$ has its subtree $T(u)$ consisting of the root plus any nodes representing its neighbours in V_2. Here u would be adjacent to v_1, v_3 and $v_{|V_2|}$.

singular node x_i. For each vertex u of V_1, the subtree $T(u)$ that corresponds to it is the root of the tree r plus the nodes of $|V_2|$ adjacent to u (i.e. $T(u) = \{r\} \cup \{x_i : (u, v_i) \in E, v_i \in V_2\}$). See Figure 2. In this fashion, the subtrees corresponding to vertices of V_1 always intersect each other at the root, so the vertices of V_1 forms a clique as required. The subtrees corresponding to vertices of V_2 never intersect, so V_2 forms an independent set as required, and every edge that exist between V_1 and V_2 exists since the subtree of $u \in V_1$ extends to the node that represents $v_i \in V_2$.

We will now give an information-theoretic lower bound for chordal graphs with bounded leafage k by counting a subclass of these graphs.

▶ **Theorem 7.** *Any data structure supporting adjacency queries on chordal graphs on n vertices with bounded vertex leafage k such that $k = o(n)$ requires at least $(k-1)n \log n - kn \log k - o(nk \log n)$ bits.*

Furthermore, if $k = cn$ for a fixed constant $0 < c \leq 1/4$ then it requires at least $\widehat{f}(c)n^2 - o(n^2)$ bits where $\widehat{f}(c)$ is the maximum of the function

$$f(l,c) = \begin{cases} (1-l)l & \text{if } c \leq l \leq 2c \\ (1-l)(l\log(l/(l-c)) - c\log(c/(l-c))) & \text{if } 2c \leq l \leq 1 \end{cases}$$

in the range $c \leq l \leq 1$, and this maximum is always achieved in the range $2c \leq l \leq 1/2$. If $c \geq 1/4$, it requires $\frac{1}{4}n^2 - o(n^2)$ bits.

Proof. First consider the case where $k = o(n)$.

Let $|V_2| = o(n)$, $|V_2| = \omega(k)$ and let $|V_1| = n - |V_2| = n - o(n)$ be the sizes of the independents set and clique respectively for our split graph. Furthermore, assume that $|V_2|$ is large enough that we have $\log |V_2| = \log n - o(\log n)$ (i.e. $|V_2| = \Omega(n/\texttt{polylog}(n))$). We also have the estimate $\log \binom{|V_2|}{k} \approx k \log |V_2| - k \log k$. This arises from Stirling's approximation and that $k = o(|V_2|)$:

$$\begin{aligned} &\log \binom{|V_2|}{k} \\ &= |V_2| \log |V_2| - k \log k - (|V_2| - k) \log(|V_2| - k) + \frac{1}{2}\left(\log |V_2| - \log k - \log(|V_2| - k)\right) + O(1) \\ &= k \log |V_2| - k \log k + k \left(\frac{|V_2| - k}{k}\right) \log\left(1 + \frac{k}{|V_2| - k}\right) - \frac{1}{2} \log k + O(1) \\ &= k \log |V_2| - k \log k + \Theta(k) \end{aligned}$$

The last step uses $\left(\frac{|V_2|-k}{k}\right) \log\left(1 + \frac{k}{|V_2|-k}\right) = \Theta(1)$ by the limit definition of e: $e = \lim_{x \to \infty}(1 + 1/x)^x$.

For each vertex u of V_1 we can select k vertices from $|V_2|$ to be adjacent to. So all subtrees in the tree representation constructed above has at most k leaves, and thus the graph obtained has bounded vertex leafage k. The number of graphs we can generate is $\binom{|V_2|}{k}^{|V_1|}$ since each vertex of V_1's neighbourhood could be any size k subset of V_2. Note that if we treat this construction as a labelled graph, then all graphs we construct are distinct, as different sets of vertices of $|V_2|$ leads to a different neighbourhood and hence a different graph. Moving from labelled graphs to unlabelled graphs, we generate each unlabelled graph at most $n!$ times due to the number of potential isomorphisms of the graph. Thus the lower bound on the space needed for k vertex leafage graphs is at least

$$\begin{aligned}
&\log\left(\binom{|V_2|}{k}^{|V_1|}\right) - n\log n \\
&= k|V_1|\log(|V_2|) - k|V_1|\log k - n\log n + O(k|V_1|) \\
&= (k-1)n\log n - kn\log k - o(kn\log n)
\end{aligned}$$

Now consider the case in which $k = cn$ for some constant $c \leq 1/4$. [3] Let l be variable such that $c \leq l \leq 1$. Set $|V_1| = (1-l)n$ and $|V_2| = ln$. Applying Stirling's approximation, the number of graphs we generate is

$$\begin{aligned}
&\log\left(\binom{|V_2|}{k}^{|V_1|}\right) - n\log n \\
&= |V_1|\log\left(\binom{ln}{cn}\right) - n\log n \\
&= n(1-l)(ln\log(ln) - cn\log(cn) - (l-c)n\log((l-c)n) - O(\log n)) - n\log n \\
&= n^2(1-l)(l\log l - c\log c - (l-c)\log(l-c)) - o(n^2) \\
&= n^2(1-l)(l\log(l/(l-c)) - c\log(c/(l-c))) - o(n^2)
\end{aligned}$$

assuming that $2c \leq l$. In the case that $c \leq l \leq 2c$, then we do not choose k elements from V_2 but rather choose $ln/2$ elements since that maximizes the binomial coefficient. Doing so gives

$$\begin{aligned}
&\log\left(\binom{|V_2|}{|V_2|/2}^{|V_1|}\right) - n\log n \\
&= n(1-l)(ln) - o(n^2) \\
&= n^2(1-l)l - o(n^2)
\end{aligned}$$

Thus the lower bound is $\widehat{f}(c)n^2 - o(n^2)$ where $\widehat{f}(c)$ given a fixed $c \leq 1/4$ is the maximum of the function $f(l,c)$ on the range $c \leq l \leq 1$ defined by

$$f(l,c) = \begin{cases} (1-l)l & \text{if } c \leq l \leq 2c \\ (1-l)(l\log(l/(l-c)) - c\log(c/(l-c))) & \text{if } 2c \leq l \leq 1 \end{cases}$$

$f(l,c)$ is clearly increasing on the first interval, positive when $l = 2c$ and zero when $l = 1$. Taking the derivative, we see that $\widehat{f}(l,c)$ is obtained at some zero of the function:

$$d/dl(1-l)(l\log(l/(l-c)) - c\log(c/(l-c))) = (1-2l)\log(l/(l-c)) + c\log(c/(l-c))$$

[3] We will show that for $c = 1/4$ the lower bound we obtain is $\frac{1}{4}n^2 - o(n^2)$ bits which matches the lower bound for all chordal graphs. Thus, this would be the lower bound for all $k \geq n/4$ as well.

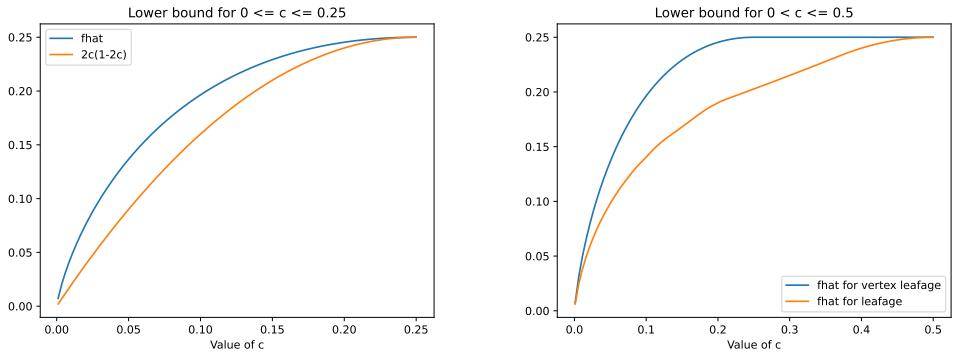


Figure 3 Left: A graph showing the value of $\widehat{f}$ for range of values of c. Right: Comparing the lower bound values for subclasses of chordal graphs (bounded vertex leafage vs bounded leafage).

Since $l \geq 2c$ in the second range, we see that $\log(l/(l-c)) > 0$ and $c\log(c/(l-c)) < 0$. Thus, in order for this sum to evaluate to 0, $1 - 2l \geq 0$, so that $l \leq 1/2$. When $c = 1/4$, we see that $l = 1/2$ is indeed a solution, which evaluates $\widehat{f}(1/4) = (1 - 1/2)(1/2) = 1/4$ matching the original chordal graph lower bound. ◀

For a graph of the value of $\widehat{f}$ (compared to the value of $f(c, 2c)$), see Figure 3.

3.2 Bounded Leafage

We now consider chordal graphs with bounded leafage k (i.e. the host tree has at most k leaves), by subdividing the edges of tree representation of the split graph with k leaves.

▶ **Theorem 8.** *Any data structure supporting adjacency queries on chordal graphs on n vertices with bounded (vertex) leafage k such that $k = o(n)$ requires at least $(k-1)n\log n - kn\log k - o(kn\log n)$ bits.*

If $k = cn$ for some constant fixed constant $0 < c \leq 1/2$, then it requires at least $\widehat{f}n^2 - o(n^2)$ bits, where $\widehat{f}$ is the maximum of the function:

$$f(c,l) = n^2 c(1-l)\left(\log\left(\lfloor l/c\rfloor + 1\right) + [l/c]\log\left(1 + \frac{1}{1+\lfloor l/c\rfloor}\right)\right)$$

Here $[x]$ denotes the fractional part of x.

Proof. We define the host tree T to be an (extended) star, with central node v_0 of degree k together with k paths of length $n/(k\log n)$, between v_0 and a leaf. We set the subtrees representing $n/\log n$ vertices of the chordal graph (which we will call *static* vertices) to be distinct singleton node in the paths, excluding v_0. For the remainder of $n - n/\log n$ vertices (which we will call *variable* vertices), their subtrees will consist of the root v_0 plus a (potentially empty) prefix of each of the paths (i.e. it is the union of k such paths). Thus each variable vertex that has a different subtree than another variable vertex will have a different set of static vertices as neighbours. See Figure 4.

The number of different possible variable vertices is $(n/(k\log n))^k$ since for each of the k paths, we have $n/(k\log n)$ different lengths. Since each of the $n - n/\log n$ variable vertices can be one of these, the total number of ways we can choose our variable vertices is $(n/(k\log n))^{k(n-n/\log n)}$. The total number of graphs we can generate is thus $(n/k\log n)^{k(n-n/\log n)}/n!$ where each graph may be generated at most $n!$ times. Thus our information theoretic lower bound is:

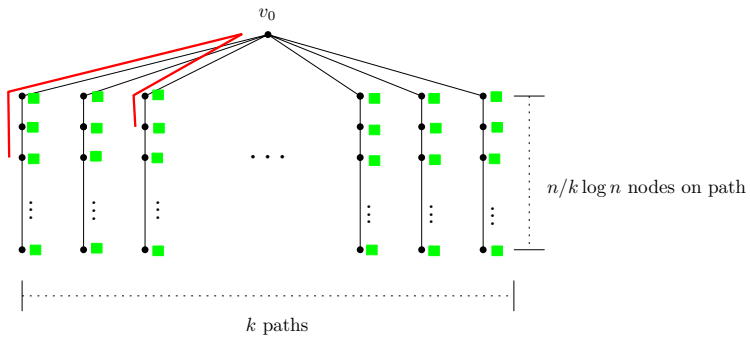


Figure 4 The host tree is a star, with k paths, each of length $n/k \log n$. Green squares represent the subtrees for vertices in V_1. A possible subtree for a vertex in $V \setminus V_1$ in red bolded lines.

$$\log\left((n/k\log n)^{k(n-n/\log n)}/n!\right) = kn(1-1/\log n)(\log n - \log k - \log\log n) - n\log n$$
$$= (k-1)n\log n - kn\log k - o(kn\log n)$$

This applies for $k \leq n/\log n$. To extend to $k = o(n)$ we may set $f(n) = o(n/k)$ and choose the number of static vertices to be $\frac{n}{kf(n)} = o(n)$ and the number of variable vertices to be $n - n/kf = n - o(n)$ without changing the above bound.

In the case that $k = cn$, we set the size of T to be $|T| = ln$ with $1 \geq l \geq c$. We set the length of each of the paths to be the same. That is, there are $\lceil l/c \rceil \times cn$ paths of length $\lfloor l/c \rfloor + 1$ and $(1 - \lceil l/c \rceil) \times cn$ paths of length $\lfloor l/c \rfloor$ (recall that we use $\lceil x \rceil = x - \lfloor x \rfloor$ to denote the fractional part of x). Enumerating as above, we obtain a lower bound that is:

$$\log\left((\lfloor l/c \rfloor + 2)^{\lceil l/c \rceil \times cn \times (1-l)n} (\lfloor l/c \rfloor + 1)^{(1-\lceil l/c \rceil) \times cn \times (1-l)n}\right)$$
$$= n^2 c(1-l)\left(\log(\lfloor l/c \rfloor + 1) + \lceil l/c \rceil \log\left(1 + \frac{1}{\lfloor l/c \rfloor + 1}\right)\right)$$

Finally, we choose l (depending on c) as to maximize this term. ◀

For a diagram of the value of this lower bound, compared to that of the lower bound for bounded vertex leafage chordal graphs in the previous subsection, see Figure 3.

4 Data Structures for Chordal Graphs with bounded Vertex Leafage

In this section we will give data structures for chordal graphs with bounded vertex leafage k. We will aim for $(k-1)n\log n - (k-1)n\log k + o(kn\log n)$ bits of space but as in Balakrishnan et al. [3], allow some extra space to support neighbourhood more efficiently. We note that this differs from the lower bound (outside of lower order terms) only from the $(k-1)n\log k$ versus $kn\log k$ terms (i.e. a difference of $n\log k$ bits). We note that if $k = n^{o(1)}$, $\log k = o(\log n)$ so this is a lower order term and can be ignored. Otherwise, if $\log k = \Theta(\log n)$, then the difference $n\log k$ is a lower order term: $n\log k = o(kn\log k)$. Thus in both instances, we may ignore this $n\log k$ difference as it is a lower order term. Hence such a data structure would be succinct for all $k = o(n)$. We give two data structures, where the second uses slightly more space and more time for the `adjacent` query, but uses less space for the `neighborhood` query.

We will begin with a tree representation of G, T, such that each $T(v)$ has at most k leaves. Root the tree T arbitrarily. For a subtree $T(v)$, denote the unique node a_v with the smallest depth as the *apex* of $T(v)$ (or of v). Furthermore, using the following lemma, we may assume that every node of T is the apex of some subtree $T(v)$.

▶ **Lemma 9.** *Let G be a chordal graph with bounded vertex leafage k. Then there is a tree representation T such that the subtree $T(v)$ of any vertex v of G has at most k leaves and that every node of T is the apex of some vertex. Furthermore, if G has bounded leafage k, then there exists a tree representation T with at most k leaves such that every node of T is the apex of some vertex.*

Proof. We will convert any tree representation of G into one with the property that all nodes of T is the apex of some vertex.

Suppose we have a tree representation such that some node $x \in T$ is not the apex of any vertex. Then any subtrees $T(v)$ containing x must also contain the parent of x. Merge x into its parent (i.e. the children of x become the children of the parent of x). Any two subtrees $T(u)$ and $T(v)$ that intersect at x also intersect in the parent of x, so no new intersections are created nor destroyed, maintaining a valid tree representation of G. Merging the node x into its parent does not increase the number of leaves of T, nor the number of leaves of any subtree $T(v)$, and so is a valid tree representation with bounded leafage or vertex leafage k.

Repeating this process to eliminate all such nodes x, we obtain a tree representation with the property that every node is the apex of some subtree $T(v)$. ◀

Since each node of T is the apex of some vertex, there are at most n nodes in T. Next, we make the following modification to the tree representation to ensure that the apex of each vertex a_v has at least two children in $T(v)$. This allows us to compute a_v via `LCA` of the leftmost and rightmost leaves of $T(v)$. We achieve this goal by adding two new children to each node a of T as the right most children of a. For each vertex v, whose apex a_v has 1 child in $T(v)$, we expand $T(v)$ to one of the newly added child of a_v in T. For each vertex v, whose apex a_v has 0 children in $T(v)$, we expand $T(v)$ to both newly added children of a_v in $T(v)$. This adds $2n$ new nodes to T so that $|T| = O(n)$ and retains the property that every *internal* node is the apex of some subtree $T(v)$ (leaves of T may not any more).

We first give a succinct data structure which allows us to recover the graph. We sort the vertices v by the preorder number of its leftmost leaf, and label the vertices $1, \dots n$. Let $L = 10^{N_1} \cdots 10^{N_{|T|}}$ where N_i is the number of vertices whose leftmost leaf is the tree node i. Then, $\texttt{rank}_1(L, \texttt{select}_0(L, v))$ gives the preorder number of the leftmost leaf of v.

For each vertex v, we will need to compute predecessor and successor queries on the preorder numbers of the leaves of $T(v)$. For each vertex v, let $Leaf(v, i)$ be the i-th leaf of $T(v)$ in preorder. We store the set of at most $k-1$ integers, $\{Leaf(v,i) \mid i \geq 2\}$ (i.e. excluding the leftmost leaf) using Lemma 2, in $(k-1)\log n - (k-1)\log k + o(k + \log n)$ bits. Using `select`, we may retrieve the i-th leaf in $O(1)$ time. Thus for simplicity, we may view $Leaf(v)$ as an array. Using binary search, we can support `pred` and `succ` queries on this array in $O(\log k)$ time.

We may speed this up slightly by storing a subset of the array to obtain an approximate predecessor. As this subset is sparse, we will use a *fusion tree* [16, 27].

▶ **Lemma 10** ([16, 27])**.** *A set of n elements can be stored in $O(n \log n)$ bits to support* `select`, `pred` *and* `succ` *queries in $O(\log_w(n)) = O(\log n / \log w)$ time, where w is the word size in the word RAM model.*

We store $f(k, n)$ evenly spaced elements of the array $Leaf(v)$ in a fusion tree (where $f(k,n) = o(k)$), using $O(nf(k,n)\log n) = o(nk \log n)$ space. To compute a predecessor, we first compute an approximate predecessor using the fusion tree in $O(\log(f(k,n))/\log\log n + 1)$ time. This gives a predecessor which may be too small. We linearly scan the segment defined

by this entry and the next sampled entry to find the true predecessor, using $O(k/f(k,n))$ time. Setting $f(k,n) = k/\log\log k$, we can find the predecessor using $O(\log k/\log\log n + 1)$ time.

This is enough for us to reconstruct the graph, as for each vertex v, the k leaves of $T(v)$ can be computed: the left-most using L, and the other $k-1$ leaves by scanning $Leaf(v)$. This defines the tree since the apex a_v is the lowest common ancestor of the first and last leaves of v (as we ensured that it has degree ≥ 2 in $T(v)$).

We concatenate the arrays $Leaf(v)$, padding ones that are smaller (i.e. when the subtree $T(v)$ has fewer than k leaves) so that they are all the same size. Thus we may access each array in $O(1)$ time, without spending extra overhead for pointers to each individual array. We will do this for all other similar structures (such as the fusion trees) to avoid the overhead from pointers. The total space used so far is the bitvector L, the n arrays $Leaf(v)$, n fusion trees, and the succinct tree storing T. Thus the total space needed so far is $(k-1)n\log(n/k) + o(kn\log n)$ bits.

4.1 Supporting `adjacent` Query

We first find the leftmost and rightmost leaves of $T(v)$. The leftmost leaf can be computed using L as above. The rightmost is $\texttt{pred}(Leaf(v), |T|)$. We then compute the apex a_v using `LCA` on the host tree T. We compare the apex of the two subtrees a_u and a_v for an ancestor-descendant relationship (i.e. check if $\texttt{LCA}(a_u, a_v) \in \{a_u, a_v\}$). If they are not in such a relationship, then $T(u)$ and $T(v)$ are disjoint, and u and v are not adjacent. When a_u and a_v is in an ancestor-descendant relationship, we may without loss of generality, assume that a_u is an ancestor of a_v.

We claim that $T(u)$ intersects $T(v)$ if and only if some leaf of $T(u)$ is a descendant of a_v. For one direction, if some leaf of $T(u)$ is a descendant of $T(v)$, then the path from u to this leaf contains a_v. Since this path is contained in $T(u)$, $a_v \in T(u) \cap T(v)$. Conversely, suppose that $T(u)$ intersects $T(v)$ at some node x. Then the path from x to a_u again contains a_v, so x is a descendant of a_v. Any leaf descendant of x in $T(u)$ is a descendant of a_v.

To check this condition (some leaf of $T(u)$ is a descendant of a_v), we first compute the range of preorder numbers of the subtree rooted at a_v. The smallest such preorder number is a_v, and the largest is the last child of a_v. Let $[l,r]$ be this range of preorder numbers. We wish to check if $[l,r]$ contains any leaf nodes of $T(u)$. To do so, we look at the predecessors of l and r in the sequence of leaf nodes of $T(u)$: $Leaf(u,l)$ and $Leaf(u,r)$. We observe that either l or r are leaf nodes (so $Leaf(u,l) = l$ or $Leaf(u,r) = r$) or a leaf in the range (l,r) would cause $Leaf(u,l) \neq Leaf(u,r)$. Thus computing whether a leaf of $T(u)$ is a descendant of v can be done using a constant number of bitvector, succinct tree, and predecessor queries, which takes $O(\log k/\log\log n + 1)$ time.

4.2 Supporting `neighborhood` Queries

We now consider efficiently computing the neighbourhood of a vertex v. We will break the neighbourhood into two cases, those vertices u such that a_u is a descendant of (or equal to) a_v, and those vertices u such that a_u is an ancestor of a_v.

4.2.1 Case 1: vertices u with a_u a descendant of a_v

For any case 1 neighbours u, $a_u \in T(v)$. We will traverse $T(v)$, and list the vertices u whose apex is the node of T that we currently traverse. To traverse $T(v)$, we may simulate a post-order traversal of $T(v)$, by starting at the leftmost leaf of $T(v)$ and using `parent` and `LCA`. This takes $O(|T(v)|)$ time.

To report the vertices whose apex is a given node of T, we define the operation `apex_list` which takes a node x of T and returns all vertices u such that $a_u = x$.

To support `apex_list`, we store a bitvector $A = 1^{M_1}0 \dots 1^{M_{|T|}}0$ where M_i is this number of vertices with apex at node with preorder number i. Then, $\texttt{rank}_1(A, \texttt{select}_0(A, x)) - \texttt{rank}_1(A, \texttt{select}_0(A, x-1))$ is the number of vertices with apex x (using the convention that the root of the tree T has preorder number 1). We also store an array AL of the vertices in order by their apexes. Then $\texttt{apex_list}(x)$ computes the range that stores the appropriate vertices and returns them from AL, that is,

$$AL[\texttt{select}_1(A, x), \texttt{select}_1(A, x+1) - 1]$$

As `apex_list` computes each neighbour with a given apex in $O(1)$ per neighbour, and we ensured each internal node of T is the apex of some vertex, the time complexity is $O(k + d_v)$ where d_v is the degree of vertex v. The k term is for the k leaves, which may not give a neighbour. We may easily remove this term, by storing a length k bitvector, which for each leaf of $T(v)$, stores a 1 if the leaf contains the apex of some neighbour u, or it is the leftmost child of an internal node of $T(v)$ (so we do not skip any internal nodes of $T(v)$ if all its children contributes no neighbours). During the post-order traversal of $T(v)$ we may skip the leaves that is a 0 using `select`, to only visit those leaves which has a neighbour. Thus, we return all case 1 neighbours in $O(1)$ time per neighbour.

The total space needed is a length $n + |T| = O(n)$ bit vector A, a length k bitvector for each vertex (total $O(kn)$ bits) and a size $n \log n$ bit array AL.

Lastly, we note that this is essentially storing a permutation of the vertices, between the two orderings obtained by sorting leftmost leaf of $T(v)$, and sorting by apex of $T(v)$.

4.2.2 Case 2: vertices u with a_u an ancestor of a_v

As discussed in `adjacent`, vertices u in this case has the property that some leaf of $T(u)$ is a descendant of a_v. To compute these, we will store a 2D orthogonal range reporting data structure of Lemma 6. The points we will store is the set

$$\{(l_i, u) \mid u \in V,\ l_i \text{ is the } i\text{-th leaf of } T(u),\ 2 \le i \le k\}$$

This structure stores the leaves of all vertices u. Thus we will convert the adjacency criteria into something that can be expressed using axis-aligned rectangles.

We again compute the range of preorder numbers of the subtree rooted at a_v as $[l, r]$, where $l = a_v$ and r is the rightmost child of a_v. For the x-coordinate, we may use l and r to ensure that the leaf is within the subtree rooted at a_v. For the y-coordinate, since we use the label of the vertex rather than its preorder number, we need to covert l and r into the appropriate vertex labels. To determine the set of labels of vertices whose leftmost leaf falls within the subtree rooted at a_v, we find the 0-bits (which represents vertices) which belong to the l-th and r-th 1 bit in L (which denote the delimiting nodes in T). That is we have $l' = \texttt{rank}_0(L, \texttt{select}_1(L, l)) + 1$, which finds the first 0 after the l-th 1 in L and $r' = \texttt{rank}_0(L, \texttt{select}_1(L, r+1))$ which finds the first 0 before the $r+1$-th 1 in L.

Consider $T(u)$ which contains a_v and such that a_u is an ancestor of a_v. Since a_u is an ancestor of a_v, at least one leaf is outside of the subtree rooted at a_v (since a_u has degree at least 2) and since $T(u)$ contains the node a_v, at least one leaf is within the subtree rooted at a_v. There are two cases, either the leftmost leaf of $T(u)$ is outside of the subtree rooted at a_v or if the leftmost leaf falls within this subtree, at least one other leaf falls outside of the subtree.

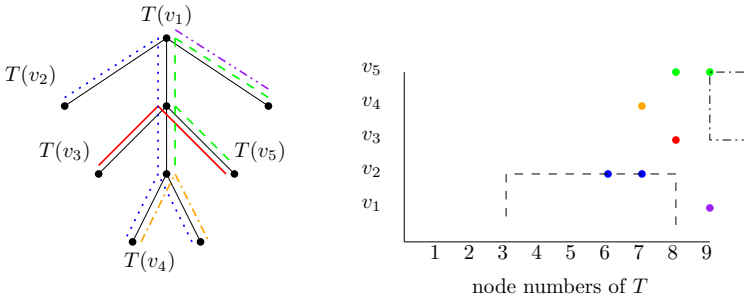


Figure 5 Example of our range search data structure. Left: a tree representation of a chordal graph (leafage 3), with 5 vertices. $T(v_1)$ in purple dash-dot-dot lines, $T(v_2)$ in blue dotted lines, $T(v_3)$ in red unbroken lines, $T(v_4)$ in orange dash-dot lines, and $T(v_5)$ in green dashed lines. For clarity, none of the simplifying operations (such as ensuring every node is the apex of some subtree) are applied to this tree representation. Right: The range search data structure for these subtrees. For v_1, the $(9, 1)$ is stored (since we do not store a point corresponding to the leftmost leaf) etc. The two types of rectangles are given for `neighborhood`(v_3). The subtree rooted at $a_{v_3} = 3$ spans the nodes $[3, 8]$ in T. The vertices whose leftmost leaf falling in this subtree are $[3, 5]$. Thus the first rectangle (in dashed lines) is $[3, 8] \times [-\infty, 2]$ and the second rectangle (in dash-dot lines) is $[9, \infty] \times [3, 5]$. In this case, the two vertices returned are v_2 and v_5.

In the first case, consider the rectangle, $[l, r] \times [-\infty, l')$. A point (l_i, u) that falls within this rectangle has the property that $l_i \in [l, r]$ so that the leaf l_i is a descendant of a_v. Furthermore, since $u < l'$, the leftmost leaf of $T(u)$ falls outside of the subtree rooted at a_v. See Figure 5.

In the second case where the leftmost leaf of the vertex u falls in the subtree rooted at a_v but some other leaf does not. This is captured by the rectangle $(r, \infty] \times [l', r']$.

Using the orthogonal range distinctness query, we retrieve each vertex once, since all the points corresponding to the same vertex has the same y-coordinate by construction. Thus, by Lemma 6 the time complexity is $O(\log n / \log \log n)$ per neighbour. The space cost of the 2D orthogonal range reporting data structure (on $(k-1)n$ points) is $(k-1)n \log kn + o(kn \log n)$ bits.

▶ **Theorem 11.** *Let G be a chordal graph with bounded vertex leafage k. Then there exists a data structure occupying $(k-1)n \log n - (k-1)n \log k + o(kn \log n)$ bits which answers* `adjacent` *in $O(\log k / \log \log n + 1)$ time. With additional $kn \log(kn)$ bits, it can answer* `neighborhood` *in $O(\log n / \log \log n)$ time per neighbour.*

4.3 Second data structure

We note that the previous data structure stores the coordinates of the leaves of the subtrees $T(v)$ twice, once in the array $Leaf(v)$ and once in the orthogonal range searching data structure to support `neighborhood` queries. This causes the space to be unnecessarily large to support `neighborhood` queries. We observe that the orthogonal range search data structure can be used to mimic access into the $Leaf$ array. By doing so, the time complexity for `adjacent` increases slightly, from $O(\log k / \log \log n)$ to $O(\log n / \log \log n)$, but the overall space cost can be reduced.

Recall that our orthogonal range search data structure stores the points:

$$\{(l_i, u) \mid u \in V, l_i \text{ is the } i\text{-th leaf of } T(u),\ 2 \leq i \leq k\}$$

To be able to compute all the queries, we need to be able to simulate access to the array $Leaf(v)$. For our purposes, this means predecessor and successor queries.

First we note that the relevant points corresponds to the rectangle $[-\infty, \infty] \times [v, v]$. Thus in $O(k \log n / \log \log n)$ time, we may retrieve the preorder numbers of each of the leaves for a given vertex.

For the predecessor and successor queries, the relevant points our query forms a horizontal strip, containing a single y-coordinate. Since predecessor queries corresponds to the point in the rectangle $[-\infty, i] \times [v, v]$ with the largest x-coordinate this corresponds to a orthogonal range distinct maximum query, with a single y-coordinate in the output. Thus, we may support it in $O(\log n / \log \log n)$ time. Symmetrically for successor queries.

Therefore, as accessing the array $Leaf(v)$ using predecessor and successor queries, using the orthogonal range search data structure incurs a cost of $O(\log n / \log \log n)$, we obtain the following theorem:

▶ **Theorem 12.** *Let G be a chordal graph with bounded vertex leafage k. There exists a data structure occupying $(k-1)n \log n + (k-1)n \log k + o(kn \log n)$ bits which answers* `adjacent` *in $O(\log n / \log \log n)$ time and, using $n \log n + o(n \log n)$ additional bits,* `neighborhood` *in $O(\log n / \log \log n)$ time per neighbour.*

4.4 Distances

In this section we investigate the `distance` query in chordal graphs with bounded vertex leafage k. In the same vein as Munro and Wu, we show a close relationship to a variation of the set intersection oracle problem. The set intersection oracle (or set disjointness) problem is the following:

Preprocess a collection of sets $\{S_1, \ldots, S_n\}$ with each set $S_i \subseteq U$ from some universe U. Answer queries of the form: given i, j is $S_i \cap S_j = \emptyset$?

There are a few ways to measure the size of this problem. One way is n, the number of sets. Another is $N = \sum_i |S_i|$, the total number of elements in the sets. As such, there are conjectures related to both of these measurements.

Patrascu and Roditty [26] gave the following conjecture, based on the number of sets in the input.

▶ **Conjecture 13** (Conj. 3 [26]). *Let $\{S_1, \ldots, S_n\}$ be a collection of sets, with each set $S_i \subseteq U$ from some universe U. Then any data structure answering set disjointness queries in $O(1)$ time, must use $\Omega(n^2)$ bits of space, even if $|U| = \log^c n$ for a large enough constant c.*

Observe that with n^2 bits we can write down the result matrix $M[i, j] = 1$ if $|S_i \cap S_j| > 0$ which allows the computation of the query in $O(1)$ time, and so this conjecture essentially states that this is the best we can do, even with small universes.

Using the total sizes of the sets as the input size, Goldstein et al. [19] gave the following conjecture.

▶ **Conjecture 14** (Conj 3. [19]). *Let $S_1, \ldots, S_n$ be a collection of sets, and let $N = \sum_i |S_i|$ be the total number of elements of the sets. Then any data structures answering set disjointness queries in T time must use $\Omega(N^2/T^2)$ space.*

These conjectures suggest that the set intersection oracle problem is quite difficult to solve without using trivial solutions such as writing down all the answers, or writing down the answers for large sets and naively iterating through smaller sets (i.e. we could check for every $x \in S_i$, is $x \in S_j$?, which would be quick if $|S_i|$ was small).

Recall our lower bound construction proof. For $k = o(n)$ we constructed two sets V_1 (which is a clique) and V_2 (which is an independent set). In this construction, for two vertices $v_1, v_2 \in V_2$, $\texttt{distance}(v_1, v_2) = 2$ exactly when there exists some vertex $u \in V_1$ that is adjacent to both. Otherwise, we pick any neighbour of v_1 and any neighbour of v_2. They both lie in V_1 which is a clique, so are adjacent. Hence the distance would be 3. Thus $\texttt{distance}(v_1, v_2)$ is either 2 or 3 depending on the condition: there exists some $u \in V_1$ adjacent to both v_1 and v_2.

Fix $v_i \in V_2$ and construct the set $S_i = \{u \in V_1 \mid (u, v_i) \in E\}$. Then the existence of u adjacent to both v_i and v_j is exactly the non-emptiness of the intersection $S_i \cap S_j$.

Munro and Wu further gave the reduction as follows: Given the sets $S_1, \ldots S_n \subseteq [1, \ldots, n]$ construct the split graph with $V_1 = [1, \ldots n]$ and $|V_2| = n$. For each $v_i \in V_2$, the neighbours of V_i is exactly the set S_i. Then the `distance` query on G answers the set disjointness problem.

If we do exactly the same thing, we obtain an obstacle. Since each vertex in V_1 has at most k neighbours in V_2 in a bounded vertex leafage chordal graph, this corresponds to the condition that: for every element $i \in U$, it belongs to at most k of the sets $S_1, \ldots, S_n$. This implies that $N \leq kn$, and also implies that $k^2 n$ bits suffice to write down the results of all queries, since for element $i \in U$, it causes at most k^2 pairs of sets to intersect.

Using this sparsity like condition, we give the following conjecture:

▶ **Conjecture 15.** *Let $\{S_1, \ldots, S_n\}$ be a collection of sets with universe $|U| = n$ such that for every $i \in U$, i appears in at most k sets. Any data structure which has query time $O(1)$ must use $\Omega(\min(k^2 n, n^2))$ bits of space.*

For chordal graphs with bounded vertex leafage, unless the above conjecture were to be false, we would be unable to answer `distance` queries in $O(1)$ time using succinct space (or even a little bit more space).

5 Data Structure for Chordal Graphs with Bounded Leafage

In this section, we consider data structures for chordal graphs with bounded leafage k. We note that a chordal graph with bounded leafage k is also a chordal graph with bounded vertex leafage k, and thus the data structures in the previous section applies. As this is a more structured class of graphs, our goal is to achieve much better query times, on top of not requiring additional space for neighbourhood queries. Furthermore, this structure allows us to bypass the difficulties with supporting the `distance` query. We will highlight some of the differences in our data structure that is made possible by this additional structure, which allows for more efficient computation.

By definition, a chordal graph with bounded leafage k has a tree representation that consists of a host tree T with at most k leaves. First, root T at one of its leaves. By doing so, we only need to worry about the $k - 1$ other leaves of the tree. We may apply Lemma 9, to ensure that $|T| \leq n$ and that every node of T is the apex of some subtree $T(v)$.

Since there are at most $k - 1$ leaves besides the root, we may break the tree down into $k-1$ disjoint paths $P_1, \ldots, P_{k-1}$, where each path contains 1 leaf of the tree, except P_1 which contains both a leaf and the root (which was also a leaf of the unrooted tree by construction). For simplicity, we may choose the paths in a preorder traversal, by descending down the tree until we reach the first leaf to obtain the first path, then following the preorder traversal to reach the second leaf to obtain the second path etc. In this way, the preorder traversal of the tree T gives $k - 1$ blocks of nodes corresponding to the $k - 1$ paths. In the following, we

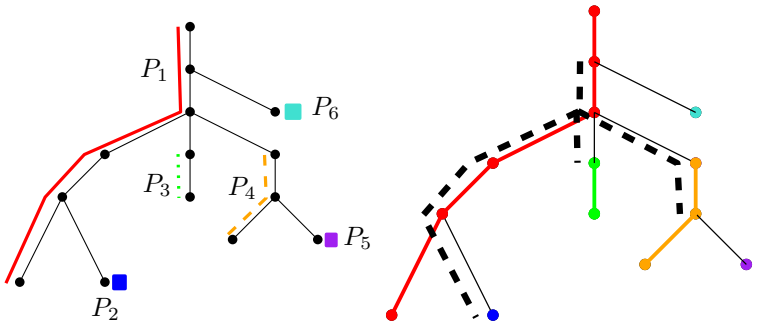


Figure 6 Left: An example host tree with 7 leaves (leafage 7), broken down into 6 paths so that the nodes of the paths are consecutive in preorder. The bitvector PN for this tree would be 10000011010011. Right: an example subtree $T(v)$ in black dashed with the host tree's path decomposition coloured in. The list of depths of the deepest node of $T(v)$ in the paths in order is 4, 0, 0, 1, -1, -1.

will refer to leaves of a subtree $T(v)$ which lie on some path P_i. However it may be the case that $T(v) \cap P_i$ is non-empty but no leaf of $T(v)$ exists on P_i. To ensure that each path P_i which contains nodes from $T(v)$ "has" a leaf of $T(v)$, we will consider the the largest depth node of $T(v)$ on a path P_i to be a leaf. See Figure 6, the subtree on the right, depicted by dashed black lines, has a "leaf" at the node of depth 4, on the red path (P_1).

Because there are only $k-1$ paths, for a subtree $T(v)$ to have k leaves, one of its leaves must also be the apex. Therefore, *rather than sorting by leftmost leaf as in the case of bounded vertex leafage, we now sort by their apex instead*, as obtaining the apex, will also obtain the leftmost leaf, if that is the case.

Sort the vertices by the preorder number of their apex, breaking ties arbitrarily and label the vertices as $1, \ldots, n$ by their index in this sorted order. From now on, when we refer to a vertex v, we refer to its label (i.e. v is an integer between 1 and n). Store a bitvector $L = 10^{N_1-1} \cdots 10^{N_{|T|}-1}$, where N_i is the number of vertices whose apex is the node with preorder number i. Then the preorder number of the apex of a vertex v is $\texttt{rank}_1(L, v)$. We note that by Lemma 9, this bitvector is well defined, as $N_i > 0$.

For each vertex v and for each of the $k-1$ paths, we store the depth (in the path) of the largest depth node (i.e. the leaf of $T(v)$ on that path) of $T(v)$ on that path, and if $T(v)$ does not intersect that path, we store -1 instead. Let $Leaf(v, j)$ be the depth of this leaf of $T(v)$ on path P_j. We store the partial sums $\sum_{1 \le \le j}(Leaf(v, j) + 2)$ (to ensure that the terms are increasing). Applying Lemma 2, and using the fact that the total sum equals $n + k$ (as the paths are disjoint, so their total lengths equal $|T| \le n$), and that $k = o(n)$, this uses $(k-1)\log(n+k) - (k-1)\log k + o(k + \log n)$ bits. To compute $Leaf(v, j)$, we have $Leaf(v, j) = \texttt{select}(j) - \texttt{select}(j-1) - 2$.

We note that since the i-th leaf always falls on the i path, instead of being arbitrarily placed in the tree as in the bounded vertex leafage case, we may retrieve it in $O(1)$ time, rather than requiring predecessor queries which takes longer.

Finally we store a bitvector PN (path number) in the preorder traversal of the tree T, with a 1 bit if a node is the first node of some path P_i. Given a node $x \in T$ (with preorder number x), it lies on the path $\texttt{rank}_1(PN, x)$.

The total space needed so far is $(k-1)n \log n - kn \log k + o(kn \log n)$ bits.

Again, using these data structures, we are able to compute the exact subtree $T(v)$ for any vertex v given as an integer denoting its position in sorted order: the root of $T(v)$ is the node with preorder number $\texttt{rank}_1(L, v)$, and we store the at most $k-1$ leaves of

the tree $T(v)$ by their depths $Leaf(v, i)$. For path i, the first node (by preorder number) of the path is $\texttt{select}_1(PN, i)$, so the leaf of $T(v)$ is the node of T with preorder number $\texttt{select}_1(PN, i) + Leaf(v, i)$ (if $Leaf(v, i) \neq -1$). Thus by reconstructing the subtress $T(v)$ we are able to recover the graph.

Now we will show how to answer the queries efficiently.

5.1 `adjacent` (u, v)

Find a_u and a_v using L. Then by performing `LCA` in T, determine if one is the ancestor of the other. If not, return not adjacent. Otherwise, suppose that a_u is an ancestor of a_v. Let j be the path number of a_v. We check if the j-th leaf of $T(u)$, $Leaf(u, j)$, has depth at least that of a_v (relative to P_j). If so, return adjacent, and return not adjacent otherwise. This uses a constant number of bitvector and succinct tree operations, and so takes $O(1)$ time.

5.2 `neighborhood` (v)

We consider two cases. Case 1: vertices u adjacent to v such that a_u is an ancestor of a_v. Case 2: vertices u adjacent to v such that a_u is a descendant of (or equal to) a_v.

Case 1. Let j be the path number of a_v, and let d_v be the depth of a_v relative to this path. As in `adjacent`, we want vertices u whose j-th leaf has depth at least as much as d_v. *Compared to the previous section, knowing that we only need to consider the j-th leaf, rather than an arbitrary leaf allows us to forgo a orthogonal range search data structure.* Let $D_j[u] = Leaf(u, j)$ be a conceptual array. We are looking for indices u such that $u < v$ (so that it is an ancestor) and $D_j[u] > d_v$. It is well known that this can be accomplished in $O(1)$ time per index by equipping D_j with a RMQ data structure (Lemma 5). The space needed is $O(kn)$ bits as we need one RMQ data structure per path.

Case 2. We consider each path in order. For each vertex, we store a length k bitvector indicating whether the j-th leaf is -1, and using `select` we iterate over each path with a node in $T(v)$. For path j if $T(v) \cap P_j$ is non-empty, let x_1, x_2 be the first and last vertices of $T(v) \cap P_j$. x_2 is simply the j-th leaf, and x_1 is the first node of the path, which we obtain using $\texttt{select}(PN, j)$, except on the path that a_v is on, in which case, $x_1 = a_v$. Since we sorted the vertices by their apex, the set of vertices with apex in $[x_1, x_2]$ forms a consecutive block, which we may find using `select` on L as $[\texttt{select}_1(L, x_1), \texttt{select}_1(L, x_2 + 1) - 1]$. This takes $O(1)$ time per neighbour.

5.3 Iteration through Neighbourhood

It maybe the case that for a graph algorithm, storing the entire neighbourhood of a vertex, especially over a large number of recursive calls is too costly. Instead it would be desirable to iterate over the list of neighbours, so that at any point, only a constant amount of space is needed to store where in the list of neighbours the computation is currently at.

To do so, we consider the two cases in the $\texttt{neighborhood}(v)$ query. Let a_v be the apex of v. First consider the second case, where we output neighbours u of v such that a_u is a descendant of a_v in the host tree T. In this case, we examined each of the $(k-1)$-paths which contained a node of $T(v)$, then for each node on the j-th path $P_j \cap T(v)$, we returned all vertices with that node as its apex. The vertices whose apex is on this subpath $P_j \cap T(v)$ forms a single interval of vertex labels. Therefore, to store where we are in this computation,

we need to store the current path number (from which we may compute the subpath in $O(1)$ time), and the current node in this single interval that we have returned. This uses $O(1)$ words of space.

Now consider the first case, where we output neighbours u of v such at a_u is an ancestor of a_v. Let i be the path number of a_v. In this case, we used a RMQ query on the conceptual array D_i which stored the leaf numbers of each vertex in the i-th path (which may be -1 if a vertex did not have a vertex on the i-th path). In this case, we wished to output all indices u in the prefix $D_i[1, \ldots, v']$ such that $D_i[u] \geq \texttt{depth}_{P_j}(a_v)$.

A solution to this was given by Tsakalidis et al. [30] in their work for permutation graphs. Their extension to the regular RMQ data structure can be stated as:

▶ **Lemma 16** (Lemma 3.2 of [30]). *Let $A[1..n]$ be a static array of comparable elements. For any constant $\varepsilon > 0$, there is a data structure using εn bits of space on top of A that supports the following queries in $O(\frac{1}{\varepsilon})$ time (making as many queries to A) and using $O(1)$ words of working memory:*

1. *range-maximum queries (or symmetrically range-minimum queries), $RMQ_A(\ell, r)$,*
2. *next-above queries (or symmetrically next-below), $next_above_A(\ell, r, y; i)$, enumerating $\{i \in [\ell, r] : A[i] \geq y\}$ in amortized $O(\frac{1}{\varepsilon})$ time.*
   Formally, next_above implicitly defines a sequence $(i_j)_{j \geq 0}$ via $i_0 = RMQ_A(\ell, r)$ if $A[i_0] \geq y$ and $i_0 = null$ otherwise, and $i_{j+1} = next_above_A(\ell, r, y; i_j)$ if $i_j \neq null$, and $i_{j+1} = null$ otherwise. Then we require $\{i_j : i_j \neq null\} = i \in [\ell, r] : A[i] \geq y$.

We note that this data structure requires access to the array A, as opposed to the data structure of Lemma 5. Here $A = D_i$, and accessing elements of $D_i[u] = Leaf(u, i)$ can be done in $O(1)$ time. Lastly, we note that the *next_above* operation iterates over all indices that we need.

Therefore, combining these two cases, we may store a reference to where we are in the list of neighbours in $O(1)$ words.

5.4 $\texttt{distance}(u, v)$

In the previous section, for chordal graphs with bounded vertex leafage, we argued that it is difficult to support the distance query, due to a close connection to the set disjointness problem. Applying the same reduction, we see that we no longer have such an obstacle. Indeed, as the host tree contains at most k leaves, the instances of the set disjointness problem that can be reduced to chordal graphs with bounded leafage are exactly those with at most k input sets. As we noted, we may store the output matrix in this case using k^2 bits, which is much less than $(k-1)n \log n$ bits. Thus, the distance query seems hopeful to be solved for the bounded leafage case.

To compute the distance between two vertices, we will use the greedy algorithm of Munro and Wu [25] for general chordal graphs. Informally, to connect two subtrees $T(u)$ and $T(v)$ (using a sequence of subtrees), at least one subtree must pass through the lowest common ancestor of a_u and a_v. To reach this LCA from a_u, we greedily pick the subtree $T(w)$ intersecting $T(u)$ such that a_w is as high up in the tree as possible (and repeat).

The successor operation on the tree T is defined by: $succ(x) = y$ (x, y are nodes of T) where y is the smallest depth node such that there exists a vertex w with $y = a_w$ and $x \in T(w)$. A shortest path between u and v can be found by first computing the node $h = \texttt{LCA}(a_u, a_v)$, then repeatedly take the successor operation from a_u and a_v. We stop when the next successor operation would give a node that has smaller depth than h. The result of the two chains of successor operations are two nodes h_u (an ancestor of a_u) and h_v (an

ancestor of a_v). Lastly, we compute whether there exists a subtree $T(w)$ such that $h_u \in T(w)$ and $h_v \in T(w)$. The shortest path between any vertex with apex h_u and any vertex with apex h_v is $h_u \to w \to h_v$ if such a subtree $T(w)$ exists and $h_u \to succ(h_u) \to succ(h_v) \to h_v$ if no such $T(w)$ exists.

We assume that $h \neq a_u, a_v$ (i.e. h_u and h_v are not in an ancestor-descendant relationship), since if one is the ancestor of the other, then there is a single chain of successor operations and we do not need to compute if $T(w)$ exists or not.

Suppose that we are given h_u and h_v, two nodes of T. Let P_u, P_v be the two paths containing h_u and h_v. For now, assume that P_u and P_v are such that no node of one path is an ancestor of another path. For each node of P_u and P_v we wish to store a bit which is 1 if there exists a vertex w whose subtree $T(w)$ contains the two nodes. This allows us to finish the distance computation by checking this bit using the nodes h_u and h_v. Naively, this will use $|P_u||P_v|$ bits of space if we store a bit for each pair of nodes of the two paths.

Let $B_{u,v}[i][j]$ be this bit for the node at depth i in P_u and depth j in P_v.

▶ **Lemma 17.** *The bitarray $B_{u,v}[i]$ consists of a block of 1s followed by a block of 0s.*

Proof. Let x be the node of depth i on the path P_u. Let j be an index such that $B_{u,v}[i][j] = 1$. Then there exists some subtree $T(w)$, which contains both x and the depth j node of P_v. Since we assume that no node of P_u is an ancestor of P_v (and vice versa), $T(w)$ must also contain the node at depth $j-1$ on path P_v. Hence any 1 bit of $B_{u,v}[i]$ is preceded by another 1 bit, and we conclude that this bitarray consists of a block of 1s followed by a block of 0s. ◀

Using the above lemma, we may describe this bitvector $B_{u,v}[i]$ using a single number: the index of the last 1. Thus, consider the array $A_{u,v}$ of length $|P_u|$ which contains these indices of the last 1.

▶ **Lemma 18.** *The array $A_{u,v}$ is non-increasing.*

Proof. We show that $A_{u,v}[i] \geq A_{u,v}[i+1]$ for all i. Let $T(w_i)$ be the subtree which realizes $A_{u,v}[i]$. That is, it is the subtree which contains the node x_i of depth i on P_u (among all such subtrees) which contains the deepest node on P_v (which is of depth $A_{u,v}[i]$). Similarly, let $T(w_{i+1})$ be the subtree which realizes $A_{u,v}[i+1]$. Note that since $T(w_{i+1})$ contains the node x_{i+1} it must also contain x_i (we assumed that no node of either path is an ancestor of nodes of the other path). Therefore, by definition of $T(w_i)$, its deepest nodes on P_v is at least as deep as that of $T(w_{i+1})$. ◀

Now, as $A_{u,v}$ is a non-increasing sequence (i.e. sorted) of length $|P_u|$ with maximum value $|P_v|$, we may store it using $|P_u| + |P_v| + o(|P_u| + |P_v|)$ bits by Lemma 3. To obtain $B_{u,v}[i][j]$, we compute $j \leq A_{u,v}[i]$, and if so the bit is 1.

Next we deal with the case that some nodes of P_u and P_v and in an ancestor-descendant relationship. Since we assume that h_u and h_v are not in an ancestor-descendant relationship, if the paths P_u and P_v contain nodes that are in an ancestor-descendant relationship, we may ignore them, as we will never query those pairs. Suppose that $P_u[0, l]$ are ancestors of the nodes of P_v (observe that only one path can contain such nodes and they must be a prefix), then we perform the above calculations using $P'_u = P_u[l+1, -]$, and store $A[0] = A[1] = \cdots = A[l] = A[l+1]$ so that sequence is still non-increasing. Note that for this pair of paths, $A[0, \ldots, l]$ will never be accessed so their values can be anything. See Figure 7.

The total size of these non-increasing sequences, stored using Lemma 3 is then $\sum_{i<j} |P_i| + |P_j| \leq k \sum_i |P_i| = kn + o(kn)$ bits. The time to compute the distance query is $O(1)$. We may also compute the shortest path by storing the vertex w_i giving rise to the value of $A_{u,v}[i]$. This takes a further $kn \log n$ bits.

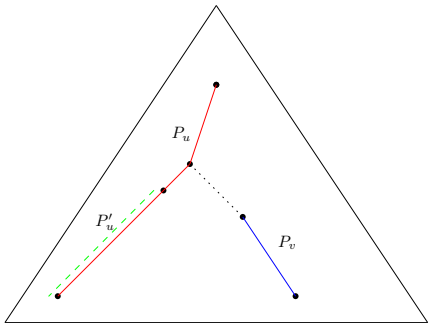


Figure 7 The case when some nodes of one path are ancestors of nodes of the other path. We ignore such nodes and work with the subpath that are not ancestors P'_u, before padding the final sequence with the ignored elements.

The total space needed is $(1+\varepsilon)n\log n$ bits for the necessary data structures from Munro and Wu's data structure. Our data structure for computing the existence of the vertex w uses $O(kn)$ bits for the `distance` query. However $kn\log n$ bits is needed to store the vertices w themselves for the `spath` query which asks for the vertices on a shortest path.

Putting the above together, we obtain the following theorem:

▶ **Theorem 19.** *Let G be a chordal graph with bounded leafage k. There exists a data structure occupying $(k-1)n\log n-(k-1)n\log k+o(kn\log n)$ bits of space supporting* `adjacent` *queries in $O(1)$ time, and* `neighborhood` *in $O(1)$ time per neighbour. Furthermore, we may iterate through the neighbourhood using $O(1)$ words to denote the current neighbour in the list, in $O(1)$ amortized time per neighbour.*

We may support `distance` *queries using $(1+\varepsilon)n\log n+O(kn)$ additional bits in $O(1)$ time (for any constant $\varepsilon>0$), and shortest path queries using $(k+1+\varepsilon)n\log n$ additional bits in $O(1)$ time per vertex on the path.*

References

1 Hüseyin Acan, Sankardeep Chakraborty, Seungbum Jo, and Srinivasa Rao Satti. Succinct encodings for families of interval graphs. *Algorithmica*, 83(3):776–794, 2021. doi:10.1007/s00453-020-00710-w.

2 Girish Balakrishnan, Sankardeep Chakraborty, Seungbum Jo, N. S. Narayanaswamy, and Kunihiko Sadakane. Succinct data structure for graphs with d-dimensional t-representation. In Ali Bilgin, James E. Fowler, Joan Serra-Sagristà, Yan Ye, and James A. Storer, editors, *Data Compression Conference, DCC 2024, Snowbird, UT, USA, March 19-22, 2024*, page 546. IEEE, 2024. doi:10.1109/DCC58796.2024.00063.

3 Girish Balakrishnan, Sankardeep Chakraborty, N. S. Narayanaswamy, and Kunihiko Sadakane. Succinct data structure for chordal graphs with bounded vertex leafage. In Hans L. Bodlaender, editor, *19th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2024, June 12-14, 2024, Helsinki, Finland*, volume 294 of *LIPIcs*, pages 4:1–4:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SWAT.2024.4.

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

5 E. A. Bender, L. B. Richmond, and N. C. Wormald. Almost all chordal graphs split. *Journal of the Australian Mathematical Society. Series A. Pure Mathematics and Statistics*, 38(2):214–221, 1985. doi:10.1017/S1446788700023077.

6 C. Berge. Some classes of perfect graphs. In *Graph Theory and Theoretical Physics*, pages 155–165. Academic Press, London-New York, 1967.

7 Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

8 Prosenjit Bose, Meng He, Anil Maheshwari, and Pat Morin. Succinct orthogonal range search structures on a grid with applications to text indexing. In Frank K. H. A. Dehne, Marina L. Gavrilova, Jörg-Rüdiger Sack, and Csaba D. Tóth, editors, *Algorithms and Data Structures, 11th International Symposium, WADS 2009, Banff, Canada, August 21-23, 2009. Proceedings*, volume 5664 of *Lecture Notes in Computer Science*, pages 98–109. Springer, 2009. doi:10.1007/978-3-642-03367-4_9.

9 Boris Bukh and R. Amzi Jeffs. Enumeration of interval graphs and d-representable complexes, 2023. doi:10.48550/arXiv.2203.12063.

10 Peter Buneman. A characterisation of rigid circuit graphs. *Discret. Math.*, 9(3):205–212, 1974. doi:10.1016/0012-365X(74)90002-8.

11 Steven Chaplick and Juraj Stacho. The vertex leafage of chordal graphs. *Discret. Appl. Math.*, 168:14–25, 2014. Fifth Workshop on Graph Classes, Optimization, and Width Parameters, Daejeon, Korea, October 2011. doi:10.1016/j.dam.2012.12.006.

12 Clark, David. *Compact PAT trees*. PhD thesis, University of Waterloo, 1997. URL: http://hdl.handle.net/10012/64.

13 G. A. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25:71–76, 1961. URL: https://api.semanticscholar.org/CorpusID:120608513.

14 Arash Farzan and Shahin Kamali. Compact navigation and distance oracles for graphs with small treewidth. *Algorithmica*, 69(1):92–116, 2014. doi:10.1007/s00453-012-9712-9.

15 Johannes Fischer and Volker Heun. A new succinct representation of rmq-information and improvements in the enhanced suffix array. In Bo Chen, Mike Paterson, and Guochuan Zhang, editors, *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pages 459–470, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-74450-4_41.

16 Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993. doi:10.1016/0022-0000(93)90040-4.

17 Fănică Gavril. A recognition algorithm for the intersection graphs of paths in trees. *Discrete Mathematics*, 23(3):211–227, 1978. doi:10.1016/0012-365X(78)90003-1.

18 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974. doi:10.1016/0095-8956(74)90094-X.

19 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In Faith Ellen, Antonina Kolokolova, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures*, pages 421–436, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-62127-2_36.

20 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 1980.

21 Meng He, J. Ian Munro, Yakov Nekrich, Sebastian Wild, and Kaiyu Wu. Distance oracles for interval graphs via breadth-first rank/select in succinct trees. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 181 of *LIPIcs*, pages 25:1–25:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ISAAC.2020.25.

22 Meng He, J. Ian Munro, and Kaiyu Wu. Succinct data structures for path graphs and chordal graphs revisited. In Ali Bilgin, James E. Fowler, Joan Serra-Sagristà, Yan Ye, and James A. Storer, editors, *Data Compression Conference, DCC 2024, Snowbird, UT, USA, March 19-22, 2024*, pages 492–501. IEEE, 2024. doi:10.1109/DCC58796.2024.00057.

23 Douglas B. West In-Jen Lin, Terry A. McKee. The leafage of a chordal graph. *Discussiones Mathematicae Graph Theory*, 18(1):23–48, 1998. doi:10.7151/dmgt.1061.

24 Clyde L. Monma and Victor K.-W. Wei. Intersection graphs of paths in a tree. *Journal of Combinatorial Theory, Series B*, 41(2):141–181, 1986. doi:10.1016/0095-8956(86)90042-0.

25 J. Ian Munro and Kaiyu Wu. Succinct data structures for chordal graphs. In *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, pages 67:1–67:12, 2018. doi:10.4230/LIPIcs.ISAAC.2018.67.

26 Mihai Patrascu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. *SIAM J. Comput.*, 43(1):300–311, 2014. doi:10.1137/11084128X.

27 Mihai Patrascu and Mikkel Thorup. Dynamic integer sets with optimal rank, select, and predecessor search. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 166–175, 2014. doi:10.1109/FOCS.2014.26.

28 Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4):43, November 2007. doi:10.1145/1290672.1290680.

29 Donald J Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32(3):597–609, 1970. doi:10.1016/0022-247X(70)90282-9.

30 Konstantinos Tsakalidis, Sebastian Wild, and Viktor Zamaraev. Succinct permutation graphs. *Algorithmica*, 85(2):509–543, 2023. doi:10.1007/s00453-022-01039-2.

31 Nicholas C. Wormald. Counting labelled chordal graphs. *Graphs and Combinatorics*, 1(1):193–200, 1985. doi:10.1007/BF02582944.

On the Complexity of Minimising the Moving Distance for Dispersing Objects

Nicolás Honorato-Droguett
Nagoya University, Japan

Kazuhiro Kurita
Nagoya University, Japan

Tesshu Hanaka
Kyushu University, Fukuoka, Japan

Hirotaka Ono
Nagoya University, Japan

Abstract

We study Geometric Graph Edit Distance (GGED), a graph-editing model to compute the minimum edit distance of intersection graphs that uses moving objects as an edit operation. We first show an $O(n \log n)$-time algorithm that minimises the total moving distance to disperse unit intervals. This algorithm is applied to render a given unit interval graph (i) edgeless, (ii) acyclic and (iii) k-clique-free. We next show that GGED becomes strongly NP-hard when rendering a weighted interval graph (i) edgeless, (ii) acyclic and (iii) k-clique-free. Lastly, we prove that minimising the maximum moving distance for rendering a unit disk graph edgeless is strongly NP-hard over the L_1 and L_2 distances.

2012 ACM Subject Classification Theory of computation → Mathematical optimization

Keywords and phrases Intersection graphs, Optimisation, Graph modification

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.36

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2502.12903 [17]

Funding *Kazuhiro Kurita*: This work is partially supported by JSPS KAKENHI Grant Numbers JP21K17812, JP22H03549, and JST ACT-X Grant Number JPMJAX2105.
Tesshu Hanaka: This work is partially supported by JSPS KAKENHI Grant Numbers JP21K17707, JP23H04388, JST CRONOS Grant Number JPMJCS24K2.
Hirotaka Ono: This work is partially supported by JSPS KAKENHI Grant Numbers JP20H05967, JP21K19765, JP22H00513, JST CRONOS Grant Number JPMJCS24K2.

1 Introduction

Graph modification is a fundamental topic to address graph similarity and dissimilarity, where a given graph is deformed by adding or deleting vertices or edges to satisfy a specific non-trivial graph property, while minimising the cost of edit operations. The problem of determining this cost is commonly known as *graph modification problem* (GMP) and has applications in various disciplines, such as computer vision [3], network interdiction [15], and molecular biology [14]. GMPs are often categorised into vertex and edge modification problems, with edit operations restricted to the vertex and edge sets, respectively.

The cost of a single edit operation in a GMP is often determined by the specific application. In theoretical studies, a unit-cost model is often assumed, where each addition or deletion of a vertex or edge has a uniform cost. However, for such models, it is known that determining whether a graph can be modified to obtain a member of a given class is NP-hard for a

© Nicolás Honorato-Droguett, Kazuhiro Kurita, Tesshu Hanaka, and Hirotaka Ono;
licensed under Creative Commons License CC-BY 4.0

wide range of graphs and classes [2, 12, 19, 25]. These negative bounds of GMPs motivate alternative formulations for graph editing that consider domain-specific constraints and cost measures.

The choice of edit operations and their associated costs is a crucial aspect of GMPs, as different formulations capture different structural properties and computational challenges. Analogous to string similarity analysis, where modifications are based on biologically significant operations such as DNA mutations and repeats [20], graph modification problems should reflect the inherent constraints and structural properties of the graphs being studied. In particular, *geometric intersection graphs* (hereafter intersection graphs) provide a suitable framework for studying GMPs for scenarios where graphs represent spatial relationships (see, e.g., [6, 10, 23]). Given a collection of geometric objects $\mathcal{S}$, an *intersection graph* $G(\mathcal{S})$ is a graph where there is a one-to-one correspondence between the vertex set $V(G(\mathcal{S}))$ and $\mathcal{S}$, and two vertices are adjacent if and only if their corresponding objects intersect. This model includes many well-known graph classes, such as interval graphs and disk graphs. These graphs can be frequently found in real-world applications such as network modelling and bioinformatics [22].

Motivated by this context, this paper investigates GMPs for intersection graphs. In this context, two natural questions arise:

1. Are standard graph edit operations suitable for modifying intersection graphs?
2. How can the geometric properties of objects be exploited to overcome the hardness of GMPs?

To answer these questions, we introduce Geometric Graph Edit Distance, a model for modifying intersection graphs from a geometric perspective.

In the intersection graph model, a natural edit operation is to move the objects in $\mathcal{S}$. We treat this movement as a graph edit operation and focus on minimising the cost required to modify an intersection graph so that the resulting graph is in a specific graph class. The cost is quantified by the total moving distance, which is the sum of the distances by which objects in $\mathcal{S}$ are moved. More precisely, we define the problem as follows:

Geometric Graph Edit Distance

Input: A collection of n geometric objects $\mathcal{S}$.
Output: The minimum total moving distance of the objects in $\mathcal{S}$ so that the resulting intersection graph $G(\mathcal{S})$ is in Π.

We assume that Π is given by an oracle, i.e. we have an algorithm to determine whether the intersection graph $G(\mathcal{S})$ is in Π.

Related work

Numerous GMPs are known to be computationally hard. In the early 1980s, Lewis and Yannakakis [19] showed that vertex-deletion problems are NP-complete for any hereditary graph class. Similarly, many edge modification problems have been shown to be NP-complete, such as transforming a graph into a perfect, chordal, or interval graph [2]. As a result, the past decade has seen a growing interest in addressing these problems from the perspective of parameterised complexity. The recent survey by Crespelle et al. [5] provides a comprehensive overview of this subject (see also [8]).

Although classical GMPs focus on structural modifications of graphs, recent studies have explored models that include geometric constraints. Honorato-Droguett et al. [16] introduced the above geometric approach to graph modification, demonstrating that graphs

of certain classes, such as graph completeness and the existence of a k-clique, can be efficiently obtained on interval graphs. Their work highlights how the underlying geometric properties of intersection graphs can be exploited to design appropriate modification models.

In a similar vein, Fomin et al. [10] studied the *disk dispersal* problem, where a set $\mathcal{S}$ of n disks, an integer $k \geq 0$, and a real number $d \geq 0$ are given, and the goal is to determine whether an edgeless disk graph can be realised by moving at most k disks by at most d distance each. They proved that this problem is NP-hard when $d = 2$ and $k = n$ and also FPT when parameterised by $k+d$. Furthermore, they showed that the problem becomes W[1]-hard when parameterised by k when disk movement is restricted to rectilinear directions.

Expanding on this line of research, Fomin et al. [11] conducted a parameterised complexity study of edge modification problems where *scaling* objects is considered as the edit operation. Their results illustrate how alternative edit operations in geometric intersection graphs can impact computational complexity, enabling further study of geometric modification graph models. In particular, their work includes several FPT results to achieve independence, acyclicity and connectivity on disk graphs.

Our work continues these developments by introducing Geometric Graph Edit Distance, a model that considers object movement as an edit operation to modify intersection graphs. Unlike prior studies that focus on vertex and edge modifications or object scaling, our approach explicitly considers movement costs by quantifying the total moving distance required to obtain a graph in a given class. This approach enables the exploration of new algorithmic and complexity-theoretic questions in the context of geometric intersection graphs.

Our contribution

Our results are mainly focused on interval graphs and summarised in Table 1. In this paper, we deal with the following graph classes: $\Pi_{\texttt{edgeless}}$ (edgeless graphs), $\Pi_{\texttt{acyc}}$ (acyclic graphs) and $\overline{\Pi_{k\texttt{-clique}}}$ (k-clique-free graphs).

In [16], the model presented is studied mainly for classes of dense graphs. This inspires the present paper as a subsequent work, where we instead focus on classes for sparse graphs. As two fundamental classes of sparse graphs, we consider edgeless graphs ($\Pi_{\texttt{edgeless}}$) and acyclic graphs ($\Pi_{\texttt{acyc}}$). These classes have also been studied in related work on geometric intersection graphs [10, 11].

Table 1 Summary of our results. In this table, L_1 and L_2 are the Manhattan and Euclidean distances, respectively. The terms IG, UIG and UDG are abbreviations of interval graphs, unit interval graphs and unit disk graphs, respectively.

Problem Type	Graph	Target Graph Class	Metric	Weighted	Complexity
minsum	UIG	$\Pi_{\texttt{edgeless}}$	$L_2(=L_1)$	No	$O(n \log n)$
	UIG	$\Pi_{\texttt{acyc}}$	$L_2(=L_1)$	No	$O(n \log n)$
	UIG	$\overline{\Pi_{k\texttt{-clique}}}$	$L_2(=L_1)$	No	$O(n \log n)$
	IG	$\Pi_{\texttt{edgeless}}$	$L_2(=L_1)$	Yes	strongly NP-hard
	IG	$\Pi_{\texttt{acyc}}$	$L_2(=L_1)$	Yes	strongly NP-hard
	IG	$\overline{\Pi_{k\texttt{-clique}}}$	$L_2(=L_1)$	Yes	strongly NP-hard for any $1 \leq k \leq n$
minimax	UDG	$\Pi_{\texttt{edgeless}}$	L_2, L_1	Yes	strongly NP-hard

As we shall detail, $\Pi_{\texttt{acyc}}$ is contained in $\overline{\Pi_{k\texttt{-clique}}}$ in our context. As a result, one might argue that the distinction of both classes is irrelevant. However, we still consider them distinctively, as forests are a well-known class of graphs. Our analysis highlights the computational complexity of modifying intersection graphs while considering movement-based edit operations, a perspective distinct from prior work that focuses on exclusively modifying the graph structure.

Paper Organisation. Section 2 formally describes the definitions needed to address the above ideas. Section 3 presents the problem Interval Dispersal and shows that it can be solved in $O(n \log n)$ time. Using this algorithm, we establish that Geometric Graph Edit Distance can also be solved in $O(n \log n)$ time for classes $\Pi_{\texttt{edgeless}}$, $\Pi_{\texttt{acyc}}$, and $\overline{\Pi_{k\texttt{-clique}}}$ on unit interval graphs. Section 4 demonstrates that Geometric Graph Edit Distance becomes strongly NP-hard on weighted interval graphs for classes $\Pi_{\texttt{edgeless}}$, $\Pi_{\texttt{acyc}}$, and $\overline{\Pi_{k\texttt{-clique}}}$. Section 5 shows that the minimax version of Geometric Graph Edit Distance is strongly NP-hard on weighted unit disk graphs for $\Pi_{\texttt{edgeless}}$ under both the L_1 and L_2 distance metrics. Section 6 concludes with remarks on our results and potential future directions.

Due to space restrictions, we omit in-depth explanations and all full proofs of statements with a $\star$-mark. The reader is referred to the full version of this paper [17] for these details.

2 Preliminaries

This section provides the main definitions used in the paper, referencing geometry, graph, and convexity terminology from textbooks [1, 4, 7, 24].

Objects. An *interval* I is a line segment on the real line of length $\text{len}(I) \in \mathbb{R}^+$. Intervals are assumed to be open, unless explicitly stated otherwise. An interval such that $\text{len}(I) = 1$ is called *unit interval*. The *left endpoint* $\ell(I)$ of an interval I is the point that satisfies $\ell(I) \leq y$ for any $y \in I$. Similarly, the *right endpoint* $r(I)$ of I is the point that satisfies $y \leq r(I)$ for any $y \in I$. The *centre* $c(I)$ of I is the point $c(I) = (r(I) - \ell(I))/2$. Throughout the paper, we assume that the indices of a collection of intervals $\mathcal{I} = \{I_1, \ldots, I_n\}$ follow the order given by centres of intervals. That is, $c(I_i) \leq c(I_{i+1})$ for all $1 \leq i \leq n-1$. However, it is not assumed that collections are ordered when given as the input graph. Given a radius $r > 0$ and a point $p \in \mathbb{R}$, a *disk* D centred at p is the set $D = \{x \in \mathbb{R}^2 \mid \|x, p\|_2 \leq r\}$. An *open disk* D is a disk without its boundary circle; that is, $D = \{x \in \mathbb{R}^2 \mid \|x, p\|_2 < r\}$. We assume that the disks are open, unless we mention the contrary. A *unit disk* is a disk of radius $r = 1/2$.

Graphs. Throughout the paper, a graph $G = (V, E)$ is assumed to be a simple, finite, and undirected graph with vertex set V and edge set E. An *edgeless graph* is a graph $G = (V, E)$ such that $E = \emptyset$. A *k-clique* of a graph $G = (V, E)$ is a subset $W \subseteq V$ such that $|W| = k$ and for all $u, v \in W$, $u \neq v$, $\{u, v\} \in E$, for $k \leq n$. If such W exists in V, we say that G *contains a k-clique*. An *interval graph* is an intersection graph $G(\mathcal{I}) = (V, E)$ where the vertex set $V = \{v_1, \ldots, v_n\}$ corresponds to a collection of intervals $\mathcal{I} = \{I_1, \ldots, I_n\}$ and an edge $\{v_i, v_j\} \in E$ exists if and only if $I_i \cap I_j \neq \emptyset$, for any $1 \leq i, j \leq n$, $i \neq j$. An interval graph is called *unit interval graph* if $\text{len}(I) = 1$ for all $I \in \mathcal{I}$. Similarly, a *disk graph* is an intersection graph $G(\mathcal{D}) = (V, E)$ where the vertex set $V = \{v_1, \ldots, v_n\}$ corresponds to a disk collection $\mathcal{D} = \{D_1, \ldots, D_n\}$. An edge $(v_i, v_j) \in E$ exists if and only if $D_i \cap D_j \neq \emptyset$, for any $1 \leq i, j \leq n$, $i \neq j$. A *unit disk graph* is a disk graph in which the collection contains

exclusively unit disks. Unless stated otherwise, all intersection graphs are assumed to be *unweighted*. A *weighted intersection graph* assigns a multiplicative weight, called the *distance weight*, to the moving distance function of each object. The formal definition of distance weight appears in later sections when required. An (infinite) set of graphs Π is a *graph class* (or simply a class), and we say that G *is in* Π if $G \in \Pi$. A graph class Π is *non-trivial* if infinitely many graphs belong to Π and infinitely many graphs do not belong to Π. In this paper, we deal with the following non-trivial classes: (i) $\Pi_{\texttt{edgeless}} = \{G : G \text{ is an edgeless graph.}\}$, (ii) $\Pi_{\texttt{acyc}} = \{G : G \text{ is an acyclic graph.}\}$, (iii) $\Pi_{k\texttt{-clique}} = \{G : G \text{ contains a } k\text{-clique.}\}$ and (iv) $\overline{\Pi_{k\texttt{-clique}}} = \{G : G \notin \Pi_{k\texttt{-clique}}\}$.

3 Rendering Unit Interval Graphs Edgeless in $O(n \log n)$ time

We show that a graph in $\Pi_{\texttt{edgeless}}$ can be obtained in $O(n \log n)$ time given a collection of n unit intervals. We start by defining a problem that we call Interval Dispersal and then use the algorithm designed to obtain a graph in $\Pi_{\texttt{edgeless}}$, $\Pi_{\texttt{acyc}}$ and $\overline{\Pi_{k\texttt{-clique}}}$. Interval Dispersal receives as input a collection $\mathcal{I}$ of n intervals and a real $s \geq 1$, and asks for the minimum value of the total moving distance to obtain a collection $\mathcal{I}'$ that satisfies $c(I'_j) - c(I'_i) \geq s$ for each $I'_i, I'_j \in \mathcal{I}'$, $i < j$. When $s = 1$, Interval Dispersal is equivalent to Geometric Graph Edit Distance on unit interval graphs and $\Pi_{\texttt{edgeless}}$. For simplicity, the intervals are assumed to be open. This avoids the need to address infinitesimally small distances required to separate closed intervals. We must first introduce some basic definitions and notation to describe the algorithm. Given a collection of n intervals $\mathcal{I} = \{I_1, \ldots, I_n\}$, let $D = (d_1, \ldots, d_n)$ be a vector such that d_i is the moving distance applied to I_i. We denote by $\mathcal{I}^D = \{I_1^D \ldots, I_n^D\}$ the collection of intervals such that $c(I_i^D) = c(I_i) + d_i$. The set $\mathcal{D}(\mathcal{I}) \subseteq \mathbb{R}^n$ is the set of vectors that describe the moving distance applied to intervals such that the condition of Interval Dispersal is satisfied. In other words, for all $D = (d_1, \ldots, d_n) \in \mathcal{D}(\mathcal{I})$, $c(I_j^D) + c(I_i^D) \geq s$ holds for $i < j$. We use $\mathcal{D}^{opt}(\mathcal{I}) \subseteq \mathcal{D}(\mathcal{I})$ to denote the subset of vectors in $\mathcal{D}(\mathcal{I})$ that minimises the total moving distance applied to intervals; i.e. $\mathcal{D}^{opt}(\mathcal{I}) = \{D = (d_1, \ldots, d_n) \in \mathcal{D}(\mathcal{I}) \mid \sum_{1 \leq i \leq n} |d_i| = \min_{D' = (d'_1, \ldots, d'_n) \in \mathcal{D}(\mathcal{I})} \sum_{1 \leq i \leq n} |d'_i|\}$.

Intuitively, we aim to find a vector $D \in \mathcal{D}^{opt}(\mathcal{I})$ to move each interval so that the distance between each pair of intervals is at least s. Given an arbitrary $D \in \mathcal{D}^{opt}(\mathcal{I})$, the order of $\mathcal{I}^D$ may be different from the order of $\mathcal{I}$. However, it was previously shown [16] that there is always a vector $D \in \mathcal{D}^{opt}(\mathcal{I})$ such that the order of $\mathcal{I}^D$ preserves the order of $\mathcal{I}$. This implies that there always exists an optimal solution of Interval Dispersal for which checking the inequality $(c(I_{i+1}) + d_{i+1}) - (c(I_i) + d_i) \geq s$ for $\ \leq i \leq n-1$ is sufficient.

We now define the *equispace function*, which moves intervals so that the distance between their centres is exactly s, maintaining the order induced by interval centres.

▶ **Definition 1** (Equispace function). *Let $(\mathcal{I}, s)$ be an instance of Interval Dispersal where $\mathcal{I}$ is a collection of unit intervals. The* equispace function *of $\mathcal{I}$ to a point x is a function $E : \mathcal{I} \times \mathbb{R} \to \mathbb{R}$ defined as:*

$$E(\mathcal{I}, x) = \sum_{i=1}^{n} f_i(x), \quad f_i(x) = |x - c(I_i) - (n-i)s|.$$

The vector that describes the moving distances given by $E(\mathcal{I}, x)$ is defined as $E_x(\mathcal{I}) = (e_1, \ldots, e_n) = (\alpha_1 f_1(x), \ldots, \alpha_n f_n(x))$ where $\alpha_i = 1$ if $x \geq c(I_i) + (n-i)s$ and $\alpha_i = -1$ otherwise, for $1 \leq i \leq n$. We also denote by $\mathcal{I}^{E_x(\mathcal{I})} = \{I_1^{E_x(\mathcal{I})} \ldots, I_n^{E_x(\mathcal{I})}\}$ the collection of intervals where $c(I_i^{E_x(\mathcal{I})}) = c(I_i) + \alpha_i f_i(x)$ for $1 \leq i \leq n$. ⌟

By the above, $E_x(\mathcal{I}) \in \mathcal{D}(\mathcal{I})$ for all $x \in \mathbb{R}$. Moreover, $c(I_{i+1}^{E_x(\mathcal{I})}) - c(I_i^{E_x(\mathcal{I})}) = s$ for all $1 \leq i \leq n-1$. We first prove that for certain collections of intervals, minimising E gives a vector contained in $\mathcal{D}^{opt}(\mathcal{I})$.

▶ **Lemma 2** (⋆). *The equispace function $E(\mathcal{I}, x)$ is a piecewise-linear convex function.*

We define the *set of breakpoints of* $E(\mathcal{I}, x)$ to be the set $B_{\mathcal{I}} = \{b_1^{\mathcal{I}}, \ldots, b_n^{\mathcal{I}}\} = \{c(I_i) + (n-i)s \mid I_i \in \mathcal{I},\ 1 \leq i \leq n\}$. Given a collection of intervals $\mathcal{I}$, we define the equispace function $E(\mathcal{I}, x)$ as a sequence of linear functions $E_1(\mathcal{I}, x), \ldots, E_{|\mathcal{I}|+1}(\mathcal{I}, x)$. The slope of $E_i(\mathcal{I}, x)$ is less than the slope of $E_j(\mathcal{I}, x)$ for $1 \leq i < j \leq |\mathcal{I}|$. Since the equispace function is convex and piecewise linear, the points that minimise E are located within a range $b_\ell \leq x \leq b_r$, where $b_\ell \leq b_r$ and $b_\ell, b_r \in B_{\mathcal{I}}$. We prove that b_ℓ and b_r can be easily found.

▶ **Lemma 3** (⋆). *The minimum value of $E(\mathcal{I}, x)$ is given by the breakpoint $b_{(n+1)/2}^{\mathcal{I}}$ if n is odd, and by breakpoints $b_{n/2}^{\mathcal{I}}$ and $b_{(n/2)+1}^{\mathcal{I}}$ otherwise.*

By Lemma 3, the minimum value of E for an arbitrary collection of intervals $\mathcal{I}$ is given by the median value(s) of $B_{\mathcal{I}}$. We now show which collections allow minimising E to obtain a vector in $\mathcal{D}^{opt}(\mathcal{I})$, characterised as follows:

▶ **Definition 4** (Optimally Equispaceable Collections). *Given a collection of intervals $\mathcal{I}$, we say that $\mathcal{I}$ is* optimally equispaceable *if there exists a $D \in \mathcal{D}^{opt}(\mathcal{I})$ such that $D = E_{x^*}(\mathcal{I})$ and $x^* \in \arg\min_{x \in \mathbb{R}} E(\mathcal{I}, x)$. Equivalently, $\mathcal{I}$ is optimally equispaceable if $E_{x^*}(\mathcal{I}) \in \mathcal{D}^{opt}(\mathcal{I})$ for all $x^* \in \arg\min_{x \in \mathbb{R}} E(\mathcal{I}, x)$.*

▶ **Lemma 5.** *Let $\mathcal{I} = \{I_1, \ldots, I_n\}$ be a collection of unit intervals such that $c(I_{i+1}) - c(I_i) \leq s$ for $1 \leq i \leq n-1$. Then $\mathcal{I}$ is optimally equispaceable. Moreover, there exists a $D \in \mathcal{D}^{opt}(\mathcal{I})$ such that $c(I_{i+1}^D) - c(I_i^D) = s$ holds for all $1 \leq i \leq n-1$.*

Proof. We only prove the latter, as the existence of D in $\mathcal{D}^{opt}(\mathcal{I})$ directly implies the optimal equispaceability of $\mathcal{I}$. That is, we show that $\mathcal{I}^D$ satisfies $c(I_{i+1}^D) - c(I_i^D) = s$, for $1 \leq i \leq n-1$. By the definition of Interval Dispersal, we have $c(I_{i+1}^D) \geq c(I_i^D)$ and $c(I_{i+1}^D) - c(I_i^D) \geq s$ for $1 \leq i \leq n-1$. Suppose that there exists a pair of intervals I_i and I_{i+1} that satisfies $c(I_{i+1}^D) - c(I_i^D) > s$. Let $s' = c(I_{i+1}^D) - c(I_i^D)$ and $\delta = s' - s$. We show how to obtain a total moving distance D' such that $\sum_{d \in D'} |d| < \sum_{d \in D} |d|$ and $c(I_{i+1}^{D'}) - c(I_i^{D'}) = s$.

We divide the proof into three cases: (i) $d_i \geq 0$, (ii) $d_{i+1} \leq 0$ and (iii) $d_i \leq 0$ and $d_{i+1} \geq 0$. For case (i), it follows that $d_j \geq d_{j-1} \geq 0$ for $i+1 \leq j \leq n$ and $(c(I_{i+1}^D) - \delta) - c(I_i^D) = c(I_{i+1}) + (d_{i+1} - \delta) - (c(I_i) + d_i) = s$ holds. Let $D'(d'_1, \ldots, d'_n) = (d_1, \ldots, d_i, d_{i+1} - \delta, \ldots, d_n - \delta)$. The dispersal condition is satisfied by $\mathcal{I}^{D'}$. Furthermore, since $\delta > 0$, the total moving distance satisfies $\sum_{d \in D'} |d| = \sum_{j=1}^{i} |d_j| + \sum_{j=i+1}^{n} d_j - \delta < \sum_{d \in D} |d|$, which contradicts the optimality of D.

For case (ii), $d_j \leq d_{j+1}$ for $1 \leq j \leq i$ holds, and the argument for case (i) applies analogously for $D' = (d'_1, \ldots, d'_n) = (d_1 + \delta, \ldots, d_i + \delta, d_{i+1}, \ldots, d_n)$.

We only need to prove case (iii). Let $\delta = s' - s$ as in the previous cases. If $\delta \leq d_{i+1}$, then we move the intervals as in the first case. If $\delta \leq -d_i$, then we move intervals as in the second case. In both cases, the same argument applies and the total moving distance contradicts the optimality of D. Thus we assume that $\delta > d_{i+1}, -d_i$ holds. Without loss of generality, we move intervals I_j for $i+1 \leq j \leq n$ by d_{i+1} to the left by $\delta' = d_{i+1}$ and intervals I_j for $1 \leq j \leq i$ to the right by $\delta'' = (c(I_{i+1}^D) - \delta') - c(I_i^D) - s$. Then $(c(I_{i+1}^D) - \delta') - (c(I_i^D) + \delta'') = s$ holds since $d_{i+1} - \delta' = 0$. Let $D' = (d'_1, \ldots, d'_n) = (d_1 + \delta'', \ldots, d_i + \delta'', d_{i+1} - \delta', \ldots, d_n - \delta')$. The inequality $\sum_{d \in D'} |d| = \sum_{j=1}^{i} d_j + \delta'' + \sum_{j=i+1}^{n} d_j - \delta' < \sum_{d \in D} |d|$ holds since $\delta', \delta'' > 0$, which contradicts the optimality of D. Therefore, in an optimal solution, $\mathcal{I}$ must satisfy $c(I_{i+1}) + d_{i+1} - (c(I_i) + d_i) = c(I_{i+1}^D) - c(I_i^D) = s$, for $1 \leq i \leq n-1$. ◀

Let $\mathcal{I} = \{I_1, \ldots, I_n\}$ and $\mathcal{J} = \{J_1, \ldots, J_m\}$ be two collections of unit intervals and let $x_1, x_2 \in \arg\min_{x \in \mathbb{R}} E(\mathcal{I}, x)$, $x_1 \leq x_2$, and $y_1, y_2 \in \arg\min_{x \in \mathbb{R}} E(\mathcal{J}, x)$, $y_1 \leq y_2$, be the breakpoints that minimise E for $\mathcal{I}$ and $\mathcal{J}$, respectively. We say that $\mathcal{I}$ *and* $\mathcal{J}$ *intersect when equispaced* when $y_1 \leq x_2 + |\mathcal{J}|s$. In other words, $\mathcal{I}$ and $\mathcal{J}$ intersect when equispaced whenever there exist points $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$ such that there exist $I \in \mathcal{I}^{E_x(\mathcal{I})}$ and $I \in \mathcal{J}^{E_y(\mathcal{J})}$ for which $c(J) - c(I) < s$.

▶ **Lemma 6** (⋆). *Given that $\mathcal{I} \cup \mathcal{J} = \{I_1, \ldots, I_n, J_1, \ldots, J_m\}$, $\mathcal{I} \cup \mathcal{J}$ is optimally equispaceable if and only if $y_1 \leq x_2 + |\mathcal{J}|s$.*

Corollary 7 is directly implied by Lemma 6.

▶ **Corollary 7.** *If $y_1 > x_2 + |\mathcal{J}|s$, then $\mathcal{I} \cup \mathcal{J}$ is not optimally equispaceable. Moreover, the minimum total moving distance for dispersing $\mathcal{I} \cup \mathcal{J}$ is equal to $E(\mathcal{I}, x) + E(\mathcal{J}, y)$ for arbitrary $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$.*

Given a collection $\mathcal{I}$ of n unit intervals, we note that $\mathcal{I}$ can be partitioned into $m \leq n$ subcollections $\mathcal{I}_{a_1,b_1}, \ldots, \mathcal{I}_{a_m,b_m}$ such that for all $1 \leq i \leq m$, $c(I_{j+1}) - c(I_j) \leq s$ for $a_i \leq j \leq b_i - 1$. By Lemma 5, each $\mathcal{I}_{a_i,b_i}$ is an optimally equispaceable collection. We use Lemma 6 and prove the statement of Lemma 8.

▶ **Lemma 8** (⋆). *Let $\mathcal{I} = \{I_1, \ldots, I_n\} = \mathcal{I}_{a_1,b_1} \cup \cdots \cup \mathcal{I}_{a_m,b_m}$ be a collection of n unit intervals partitioned as above. If there exist integers $\alpha_1, \ldots, \alpha_k$ such that $\mathcal{I}_{a_{\alpha_i},b_{\alpha_i}}$ and $\mathcal{I}_{a_{\alpha_i+1},b_{\alpha_i+1}}$ intersect when equispaced, then there exists an optimal solution for dispersing $\mathcal{I}$ that disperses the intervals in a way that $c(I_{j+1}) + d_{j+1} - (c(I_j) + d_j) = s$ holds for $1 \leq i \leq k$ and $a_{\alpha_i} \leq j < b_{\alpha_i+1}$.*

Outline of Algorithm 1

Given a collection of unit intervals $\mathcal{I}$ and a dispersal value $s \geq 1$, the algorithm starts by sorting and partitioning $\mathcal{I}$ into $m \leq n$ disjoint subcollections $\mathcal{I}_{a_1,b_1}, \ldots, \mathcal{I}_{a_m,b_m}$ such that each $\mathcal{I}_{a_i,b_i}$ satisfies Lemma 5. Subsequently, the optimal breakpoints are determined for each $E(\mathcal{I}_{a_i,b_i}, x)$. Whenever there exist two subcollections $\mathcal{I}_{a_i,b_j}$, $i \leq j$ and $\mathcal{I}_{a_k,b_\ell}$, $k \leq \ell$ that intersect when equispaced, the algorithm considers both subcollections as a unique subcollection $\mathcal{I}_{a_i,b_\ell} = \mathcal{I}_{a_i,b_j} \cup \mathcal{I}_{a_k,b_\ell}$ and recursively determines the optimal breakpoints of $E(\mathcal{I}_{a_i,b_\ell}, x)$ using the breakpoint sets of $E(\mathcal{I}_{a_i,b_j}, x)$ and $E(\mathcal{I}_{a_k,b_\ell}, x)$. Lemma 8 ensures that this recursion partitions $\mathcal{I}$ into non-intersecting subcollections when equispaced. Lastly, the algorithm returns the total moving distance, which is calculated as the sum of the optimal values of E for each subcollection.

Before showing the complexity of the algorithm, we must characterise the set of breakpoints further. When a collection of unit intervals $\mathcal{I} = \{I_1, \ldots, I_n\}$ is partitioned into m disjoint subcollections $\mathcal{I}_{a_1,b_1}, \ldots, \mathcal{I}_{a_m,b_m}$ of intervals that satisfy Lemma 5, the set of breakpoints $B_{\mathcal{I}_{a_i,b_i}}$ is equal to $\{c(I_j) + (|\mathcal{I}_{a_i,b_i}| - j)s \mid I_i \in \mathcal{I},\ a_i \leq j \leq b_i\}$ for each $1 \leq i \leq m$. Consequently, $B_{\mathcal{I}}$ can be reformulated as follows:

$$B_{\mathcal{I}} = \left\{ c(I_j) + \left(|\mathcal{I}_{a_i,b_i}| - j + \sum_{k=i+1}^{m} |\mathcal{I}_{a_k,b_k}| \right) s \mid 1 \leq i \leq m,\ a_i \leq j \leq b_i \right\}.$$

As a result, if b and b' are the breakpoints for I in $B_{\mathcal{I}_{a_i,b_i}}$ and $B_{\mathcal{I}}$, respectively, then $b' = b - \sum_{j=i+1}^{m} |\mathcal{I}_{a_j,b_j}|$ holds. Moreover, the breakpoints of any union of subcollections $\mathcal{I}_{a_i,b_j} = \mathcal{I}_{a_i,b_i} \cup \cdots \cup \mathcal{I}_{a_j,b_j}$ can be calculated in the same way by subtracting $\sum_{k=j+1}^{m} |\mathcal{I}_{a_k,b_k}|$ from any breakpoint $b \in B_{\mathcal{I}}$ calculated using an interval $I \in \mathcal{I}_{a_i,b_j}$. It follows that the order of $B_{\mathcal{I}_{a_i,b_j}}$ is the same as the order of the corresponding breakpoints in $B_{\mathcal{I}}$.

Algorithm 1 Dispersing n unit intervals in $O(n \log n)$ time.

1 **Procedure** DispersingIntervals($\mathcal{I}$,s)
2 Sort and partition $\mathcal{I}$ into $m \leq n$ subcollections $\mathcal{I}_{a_1,b_1}, \ldots, \mathcal{I}_{a_m,b_m}$ such that for all $1 \leq i \leq m$, $c(I_{j+1}) - c(I_j) \leq s$ for $a_i \leq j \leq b_i - 1$.;
3 Compute and sort $B^*_{\mathcal{I}_{a_i,b_i}}$ for all $1 \leq i \leq n$.;
4 $x^1_{a_i,b_i}, x^2_{a_i,b_i}$ are the breakpoint $b^{\mathcal{I}_{a_i,b_i}}_{(n+1)/2}$ if $|\mathcal{I}_{a_i,b_i}|$ is odd and $b^{\mathcal{I}_{a_i,b_i}}_{n/2}$ and $b^{\mathcal{I}_{a_i,b_i}}_{(n/2)+1}$ otherwise.;
5 $D^{opt} \leftarrow \bigcup_{1 \leq i \leq n} \{(B^*_{\mathcal{I}_{a_i,b_i}}, x^1_{a_i,b_i}, x^2_{a_i,b_i})\}$;
6 **while** $x^1_{a_k,b_\ell} \leq x^2_{a_i,b_j} + |\mathcal{I}_{a_k,b_\ell}|s$, $1 \leq i \leq j < k \leq \ell \leq n$ **do**
7 $B^*_{\mathcal{I}_{a_i,b_l}} \leftarrow merge(B^*_{\mathcal{I}_{a_i,b_j}}, B^*_{\mathcal{I}_{a_k,b_\ell}})$;
8 $x^1_{a_i,b_\ell}, x^2_{a_i,b_\ell} \leftarrow b^{\mathcal{I}_{a_i,b_\ell}}_{(n+1)/2}$ if $|B^*_{\mathcal{I}_{a_i,b_\ell}}|$ is odd and $x^1_{a_i,b_\ell} \leftarrow b^{\mathcal{I}_{a_i,b_\ell}}_{n/2}$, $x^2_{a_i,b_\ell} \leftarrow b^{\mathcal{I}_{a_i,b_\ell}}_{(n/2)+1}$ otherwise.;
9 $D^{opt} \leftarrow \left(D^{opt} \setminus \left\{(B^*_{\mathcal{I}_{a_i,b_j}}, x^1_{a_i,b_j}, x^2_{a_i,b_j}), (B^*_{\mathcal{I}_{a_k,b_l}}, x^1_{a_k,b_\ell}, x^2_{a_k,b_\ell})\right\}\right) \cup \left\{(B^*_{\mathcal{I}_{a_i,b_\ell}}, x^1_{a_i,b_\ell}, x^2_{a_i,b_\ell})\right\}$;
10 **return** $\sum_{(B^*_{\mathcal{I}_{a_i,b_j}}, x_1, x_2) \in D^{opt}} E(\mathcal{I}_{a_i,b_i} \cup \cdots \cup \mathcal{I}_{a_j,b_j}, x_1)$;

The above implies that the breakpoints of any (union of) subcollection(s) can be obtained from $B_{\mathcal{I}}$. We denote the set $\bigcup_{i \leq k \leq j} \left\{ b + s \sum_{l=k+1}^{m} |\mathcal{I}_{a_l,b_l}| \mid b \in B_{\mathcal{I}_{a_k,b_k}} \right\}$ by $B^*_{\mathcal{I}_{a_i,b_j}}$ and call it the *cumulative set of breakpoints of* $B_{\mathcal{I}_{a_i,b_j}}$. We prove that $B^*_{\mathcal{I}_{a_1,b_1}}, \ldots, B^*_{\mathcal{I}_{a_m,b_m}}$ can be found in $O(n \log n)$ time.

▶ **Lemma 9** (⋆). *Let $\mathcal{I} = \{I_1, \ldots, I_n\} = \mathcal{I}_{a_1,b_1} \cup \cdots \cup \mathcal{I}_{a_m,b_m}$ be a collection of n unit intervals partitioned as above. Then the cumulative sets of breakpoints $B^*_{\mathcal{I}_{a_1,b_1}}, \ldots, B^*_{\mathcal{I}_{a_m,b_m}}$ such that each $B^*_{\mathcal{I}_{a_i,b_i}}$ is sorted can be obtained in $O(n \log n)$ total time.*

▶ **Lemma 10** (⋆). *Let $\mathcal{I} = \{I_1, \ldots, I_n\} = \mathcal{I}_{a_1,b_1} \cup \cdots \cup \mathcal{I}_{a_m,b_m}$ be a collection of n unit intervals partitioned as above. If cumulative breakpoint sets $B^*_{\mathcal{I}_{a_1,b_1}}, \ldots, B^*_{\mathcal{I}_{a_m,b_m}}$ are given so that each $B^*_{\mathcal{I}_{a_i,b_i}}$ is sorted, then merging them into one sorted set can be done in $O(n \log n)$ total time.*

▶ **Theorem 11.** *Given a collection of unit intervals $\mathcal{I}$ and a value $s \geq 1$, Interval Dispersal can be solved in $O(n \log n)$ time.*

Proof. We show the complexity of Algorithm 1. Line 2 can be done in $O(n \log n)$ time for sorting and $O(n)$ time to determine the initial m partitions. Similarly, line 3 can be done in $O(n \log n)$ time by Lemma 9. Given that each $B^*_{\mathcal{I}_{a_i,b_i}}$ is sorted, the $((|\mathcal{I}_{a_i,b_i}|+1)/2)$th element (resp. $(|\mathcal{I}_{a_i,b_i}|/2)$th and $((|\mathcal{I}_{a_i,b_i}|/2)+1)$th element) can be calculated in $O(\log |\mathcal{I}_{a_i,b_i}|)$ time using binary search on $B^*_{\mathcal{I}_{a_i,b_i}}$. This ensures that line 4 is done for all $1 \leq i \leq m$ in $O(m \log n)$ total time. We initialise D^{opt} as a doubly linked list where each node i contains the information of $(B^*_{\mathcal{I}_{a_i,b_i}}, x^1_{a_i,b_i}, x^2_{a_i,b_i})$. We show the complexity of the loop in line 6. We merge both $B^*_{\mathcal{I}_{a_i,b_j}}$ and $B^*_{\mathcal{I}_{a_k,b_\ell}}$ to obtain a sorted $B^*_{\mathcal{I}_{a_i,b_\ell}}$. Hence, the median value(s) of $B^*_{\mathcal{I}_{a_i,b_\ell}}$ can be calculated in $O(\log n)$ time by binary search. At each execution of line 7, two partitions are merged; thus the number of partitions is reduced by one unit at each iteration. Initially, there exist m partitions, and hence the loop of line 6 iterates at most $m-1$ times. Moreover, merging m cumulative sets of breakpoints into one sorted set can be done in $O(n \log n)$ time by Lemma 10, which implies that any partial merge of these sets

is also bounded by $O(n \log n)$. Consequently, the total running time of line 6 is $O(n \log n)$ time. Lastly, in line 9 the two merged sets are deleted and the new one is added. Since D^{opt} is a doubly linked list, this can be done in $O(1)$ time by connecting the previous and next node of $B^*_{\mathcal{I}_{a_i,b_j}}$ and $B^*_{\mathcal{I}_{a_k,b_\ell}}$ to a new node containing $B^*_{\mathcal{I}_{a_i,b_\ell}}$, respectively. Once there is no pair of subcollections left to merge, the total moving distance is calculated in $O(n)$ time in line 10 following the definition of cumulative set of breakpoints, which concludes that the total running time of Algorithm 1 is $O(n \log n)$ time. ◀

Theorem 11 implies the following result for $\Pi_{\texttt{edgeless}}$ on unit interval graphs when $s = 1$.

▶ **Corollary 12** (⋆). *Given a collection of n unit intervals $\mathcal{I}$, Geometric Graph Edit Distance can be solved in $O(n \log n)$ time so that $G(\mathcal{I}) \in \Pi_{edgeless}$.*

3.1 Classes $\Pi_{\texttt{acyc}}$ and $\overline{\Pi_{k\texttt{-clique}}}$ on Unit Interval Graphs

This section shows how to use Algorithm 1 for obtaining graphs in $\Pi_{\texttt{acyc}}$ and $\overline{\Pi_{k\texttt{-clique}}}$ on unit interval graphs. We first show the case for $\overline{\Pi_{k\texttt{-clique}}}$.

It is shown in [16] that given a collection of unit intervals $\mathcal{I}$, $G(\mathcal{I})$ does not contain a k-clique if and only if $c(I_{i+k-1}) - c(I_i) \geq 1$ for all $1 \leq i \leq n-k+1$. This inequality can be decomposed into $k-1$ inequalities of the following form: for each $0 \leq r \leq k-2$, $c(I_{i+k-1}) - c(I_i) \geq 1$ for all $1 \leq i \leq n-k+1$ such that $i \bmod k-1 = r$. If $\mathcal{I}$ is decomposed into $k-1$ subcollections such that $\mathcal{I} = \bigcup_{1 \leq i \leq k-1} \mathcal{I}_i$, $\mathcal{I}_i = \{I_j \in \mathcal{I} \mid 1 \leq j \leq n,\ j \pmod{k-1} = i\}$, then Algorithm 1 can be applied to each $\mathcal{I}_i$ independently for $s = 1$ to satisfy the above inequalities. Since unit interval graphs are chordal, G is acyclic if it is triangle-free; i.e. G is contained in $\overline{\Pi_{3\texttt{-clique}}}$. Consequently $\Pi_{\texttt{acyc}}$ is equivalent to $\overline{\Pi_{k\texttt{-clique}}}$ when $k = 3$. The above ideas imply Corollary 13.

▶ **Corollary 13.** *Given a collection of n unit intervals $\mathcal{I}$, Geometric Graph Edit Distance can be solved in $O(n \log n)$ time so that (i) $G(\mathcal{I}) \in \Pi_{acyc}$ and (ii) $G(\mathcal{I}) \in \overline{\Pi_{k\text{-}clique}}$.*

4 Minimising the Total Moving Distance for $\Pi_{\texttt{edgeless}}$ on Weighted Interval Graphs is Hard

In this section we show that Geometric Graph Edit Distance is strongly NP-hard on weighted interval graphs for $\Pi_{\texttt{edgeless}}$. We show a reduction from 3-Partition [13]. 3-Partition receives as input a set A of $3m$ elements, a bound $B \in \mathbb{Z}^+$ and a size $s(a) \in \mathbb{Z}^+$ such that $B/4 < s(a) < B/2$ and $\sum_{a \in A} s(a) = mB$, and the task is to decide whether A can be partitioned into m disjoint sets $A_1, \ldots, A_m$ such that for $1 \leq i \leq m$, $|A_i| = 3$ and $\sum_{a \in A_i} s(a) = B$.

Given an instance (A, B, s) of 3-Partition, we construct a collection of intervals $\mathcal{I}_A$ and show that A can be partitioned if and only if $\mathcal{I}_A$ can be modified so that $G(\mathcal{I}_A) \in \Pi_{\texttt{edgeless}}$ with at most total moving distance T. Given two intervals I, I' such that $c(I) \leq c(I')$, we say that I and I' intersect if $c(I') - c(I) < (\mathrm{len}(I') + \mathrm{len}(I))/2$.

We show the construction of $\mathcal{I}_A$ (see Figure 1). We define $\mathcal{I}_A$ as the collection $\mathcal{I} \cup \mathcal{I}^s \cup \mathcal{I}^b$ where $\mathcal{I} = \{I_1, \ldots, I_{3m}\}$, $\mathcal{I}^s = \{I^s_1, \ldots, I^s_{m-1}\}$, $\mathcal{I}^b = \{I_\ell, I_r\}$ and,

- **(i)** for $1 \leq i \leq 3m$, I_i is an interval such that $\mathrm{len}(I_i) = s(a_i)$ and $c(I_i) = -s(a_i)/2$ (that is, $r(I_i) = 0$),
- **(ii)** for $1 \leq i \leq m-1$, I^s_i is an interval where $\mathrm{len}(I^s_i) = B$ and $c(I^s_i) = (2i-1)B + B/2$ and
- **(iii)** I_ℓ and I_r are intervals such that $\mathrm{len}(I_\ell) = \mathrm{len}(I_r) = 3Bm^2 + \max_{a \in A} s(a)$, $c(I_\ell) = -3Bm^2/2$ and $c(I_r) = (2m-1)B + 3Bm^2/2$.

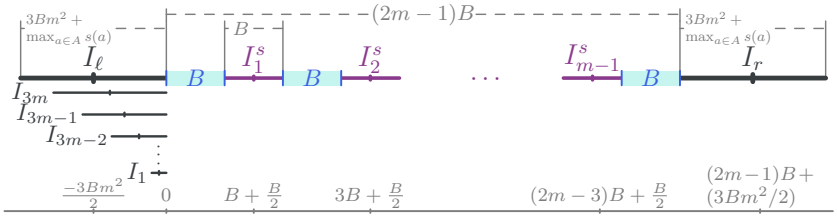


Figure 1 Reduction Overview.

For an interval $I \in \mathcal{I}_A$, we define the moving distance function $d_I : \mathbb{R} \to \mathbb{R}$ as:

$$d_I(x) = \begin{cases} |c(I) - x|, & I \in \mathcal{I}, \\ 12Bm^2|c(I) - x|, & I \in \mathcal{I}^s \cup \mathcal{I}^b. \end{cases}$$

Given an instance (A, B, s) of 3-Partition, we show the following properties.

▶ **Lemma 14** (⋆). *Given an arbitrary partition of A of m disjoint sets $A_1, \ldots, A_m$ such that $A_i = \{a_1^i, a_2^i, a_3^i\}$ for $1 \le i \le m$, $\sum_{i=1}^m 6(i-1)B + \sum_{i=1}^m (3a_1^i + 2a_2^i + a_3^i) < 3Bm^2$ holds.*

We note that Lemma 14 works for any partition of A as described above, even without the restrictions of the 3-Partition output.

▶ **Lemma 15.** *Given an instance (A, B, s) of 3-Partition, A can be partitioned into m disjoint sets $A_1, \ldots, A_m$ such that for $1 \le i \le m$ $A_i = \{a_1^i, a_2^i, a_3^i\}$, $|A_i| = 3$ and $\sum_{a\in A_i} s(a) = B$ if and only if $I\mathcal{I}_A$ can be modified so that $G(\mathcal{I}_A) \in \Pi_{\texttt{edgeless}}$ with total moving distance of at most $3Bm^2$.*

Lastly, we remark that the polynomial construction of $\mathcal{I}_A$ is straightforward by iterating over A and following the definitions given at the beginning of the section. We summarise the main result of this section as follows:

▶ **Theorem 16.** *Geometric Graph Edit Distance is strongly* NP*-hard on weighted interval graphs for the class $\Pi_{\texttt{edgeless}}$.*

We notice that Theorem 16 can be extended to show that obtaining graphs in $\Pi_{\texttt{acyc}}$ and $\overline{\Pi_{k\text{-}\texttt{clique}}}$ is also strongly NP-hard.

In particular, when obtaining a graph in $\overline{\Pi_{k\text{-}\texttt{clique}}}$, we create $k-1$ overlapping copies of the intervals in $\mathcal{I}^s \cup \mathcal{I}^b$ and add $k-1$ overlapping intervals of size B into the spaces between intervals of $\mathcal{I}^s \cup \mathcal{I}^b$ with the same moving distance function. Any interval forms a k-clique with the k copies of overlapping intervals. Consequently, moving the intervals of $\mathcal{I}$ with total moving distance of at most $3Bm^2$ is equivalent to removing all k-cliques from $\mathcal{I}_A$ with at most the same distance. Moreover, by the chordality of interval graphs, it is sufficient to obtain a graph in $\overline{\Pi_{k\text{-}\texttt{clique}}}$ when $k = 3$ for class $\Pi_{\texttt{acyc}}$. As a result, Corollary 17 is obtained.

▶ **Corollary 17.** *Geometric Graph Edit Distance is strongly* NP*-hard on weighted interval graphs for classes $\Pi_{\texttt{acyc}}$ and $\overline{\Pi_{k\text{-}\texttt{clique}}}$.*

5 Minimising the Maximum Moving Distance for $\Pi_{\texttt{edgeless}}$ on Unit Disk Graphs is Hard

In this section, we deal with the minimax version of Geometric Graph Edit Distance, defined as follows:

MINIMAX-GEOMETRIC GRAPH EDIT DISTANCE

Input: A collection of n geometric objects $\mathcal{S}$ and a real $K > 0$.
Task: Decide whether $\mathcal{S}$ can be modified so that $G(\mathcal{S}) \in \Pi$ by moving objects such that for all $S \in \mathcal{S}$, the moving distance of S is at most K.

We show that MINIMAX-GEOMETRIC GRAPH EDIT DISTANCE is strongly NP-hard on unit disk graphs for $\Pi = \Pi_{\texttt{edgeless}}$ over the L_1 and L_2 distances by reducing from PLANAR 3-SAT. Specifically, we show a proof for Theorem 18.

▶ **Theorem 18.** *MINIMAX-GEOMETRIC GRAPH EDIT DISTANCE is strongly* NP*-hard on unit disk graphs for* $\Pi_{\texttt{edgeless}}$ *over the* L_1 *and* L_2 *distances.*

Due to space constraints, we only give an overview of the reduction. The complete reduction and proofs can be found in the full-version of the paper [17].

5.1 Proof Overview of Theorem 18: Reducing Planar 3-SAT to minimax-Geometric Graph Edit Distance

We show a reduction from the following NP-complete variation of PLANAR 3-SAT [18,21,26]. Given CNF formula Φ equipped with a planar rectilinear embedding G_Φ, a set X of n variables, a set C of m clauses over X such that each $c \in C$ has length $|c| \leq 3$, each variable $x \in X$ appears in at most three clauses, and $\Phi = \bigwedge_{c \in C} c$, PLANAR 3-SAT asks whether Φ is satisfiable. We give a simplified overview of the reduction. The idea is to emulate each component (clauses, variables and connectors) of G_Φ using *disk gadgets* and construct a collection of disks $\mathcal{D}_\Phi$ equivalent to G_Φ. That is, our objective is to construct a $\mathcal{D}_\Phi$ such that Φ is satisfiable if and only if $\mathcal{D}_\Phi$ is a yes-instance of MINIMAX-GEOMETRIC GRAPH EDIT DISTANCE for $\Pi_{\texttt{edgeless}}$. To do this, we emulate the truth assignment using a proper movement of disks. To force the disk movement, we deliberately insert intersecting disks in $\mathcal{D}_\Phi$. In particular, we insert intersecting disks in clause gadgets and restrict the movement of such disks to moving a sequence of disks such that a *free slot* of a variable gadget is used. To allow the removal of the intersection, the gadgets are connected following the structure of G_Φ using consecutive disks separated by distance K. For example, consider the boolean formula Φ and its rectilinear embedding G_Φ, illustrated in Figure 2. A skeleton of the reduction is shown in Figure 3(a), where representations of clause and variable gadgets are connected following G_Φ. Let $c = (x_1 \vee \overline{x_2} \vee x_4)$ and suppose that x_2 is assigned to *false*. This assignment implies a movement of disks that (i) removes the intersections in the clause gadget for c and (ii) blocks the truth value of the variable gadget for x_2 (see Figure 3(b)). We must block the truth value of the variable gadget so that another clause gadget c' does not use the free slot in the variable gadget for x_2 when $x_2 = \textit{true}$. Consequently, their intersections must be removed using other gadgets. It can be shown that removing all intersections in this way is equivalent to a valid assignment of variables for which $\Phi = \textit{true}$. The disks are *moved* by assigning a new location, and the distance is calculated using a function that we call *moving*

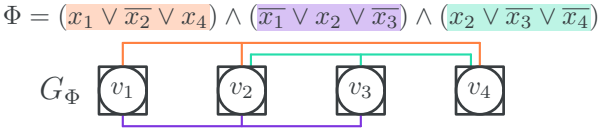


Figure 2 Reduction Overview: An arbitrary instance Φ of PLANAR 3-SAT with its rectilinear embedding G_Φ.

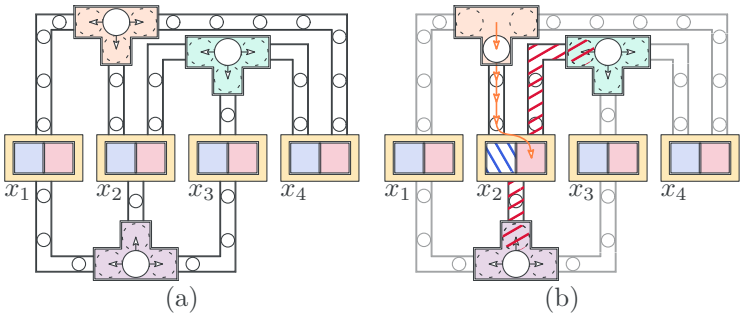


Figure 3 Reduction Overview: (a) The skeleton given by the instance (Φ, G_Φ) of Figure 2; (b) The intersection of the gadget for $c = (x_1 \vee \overline{x_2} \vee x_4)$ is removed by moving disks in a way that a free slot of the gadget for x_2 is used. Since $c = \mathit{true}$ when $x_2 = \mathit{false}$, the free slots for the other two gadgets become blocked, being unable to remove their intersection using the variable gadget for x_2.

distance function, which is the L_1 or L_2 distance metric multiplied by a *distance weight*. We employ two types of disks classified by their distance weight, called *transition disk* and *heavy disk*. The transition disks are the disks that we aim to move, whereas heavy disks are used to restrict the movement of transition disks. The moving distance function of a heavy disk is intuitively defined such that any significant movement that alters the construction exceeds a distance of K. We show that a solution that allows removing all intersections from $\mathcal{D}_\Phi$ with minimum maximum moving distance K exclusively relies on the movement of transition disks. We remark that, although heavy disks can move, their movement is negligible. Combining this condition and the above construction, it can be shown that Φ is satisfiable if and only if $\mathcal{D}_\Phi$ can be modified so that $G(\mathcal{D}_\Phi) \in \Pi_{\texttt{edgeless}}$ using minimum maximum moving distance K.

6 Concluding Remarks

The main contribution of this paper is two-fold. First, we continued the study of Geometric Graph Edit Distance originally presented in [16], showing complexity results for obtaining graphs in several classes for sparse graphs on interval graphs. In particular, we showed that obtaining a graph in $\Pi_{\texttt{edgeless}}$, $\Pi_{\texttt{acyc}}$ and $\overline{\Pi_{k\text{-}\texttt{clique}}}$ is solvable in $O(n \log n)$ time on unit interval graphs. In contrast, we showed that the problem becomes strongly NP-hard on weighted interval graphs for the same classes. Second, we defined minimax-Geometric Graph Edit Distance as a variation of the above problem and showed that it is strongly NP-hard for $\Pi_{\texttt{edgeless}}$ on weighted unit disk graphs over the L_1 and L_2 distances.

There are several directions for further research. Our results provide a comprehensive picture of the complexity of Geometric Graph Edit Distance on interval graphs. In particular, we showed that the problem becomes hard even in lower dimensions when the input is not restricted by interval size and distance weight. As a result, a potential future work is to study the complexity when exclusively one of the restrictions is applied. Another interesting direction is to study the model for $\Pi_{\texttt{edgeless}}$ in higher dimensions. Related works [9–11] suggest that our model on more complex intersection graphs becomes intractable for some of the graph classes presented in this work. In general, we deal with the edit operation that moves the objects of the given intersection graph. However, the model is not restricted to this operation. Determining Geometric Graph Edit Distance using other geometric edit operations (such as shrinking or rotating objects) is left for future research for all intersection graphs and graph classes presented in this work.

References

1 Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. doi:10.1017/cbo9780511804441.

2 Pablo Burzyn, Flavia Bonomo, and Guillermo Durán. NP-completeness results for edge modification problems. *Discrete Applied Mathematics*, 154(13):1824–1844, 2006. doi:10.1016/j.dam.2006.03.031.

3 Fan R. K. Chung and David Mumford. Chordal Completions of Planar Graphs. *J. Comb. Theory Ser. B*, 62(1):96–106, 1994. doi:10.1006/jctb.1994.1056.

4 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

5 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *Computer Science Review*, 48:100556, 2023. doi:10.1016/j.cosrev.2023.100556.

6 Mark de Berg, Sándor Kisfaludi-Bak, and Gerhard J. Woeginger. The complexity of dominating set in geometric intersection graphs. *Theoretical Computer Science*, 769:18–31, 2019. doi:10.1016/j.tcs.2018.10.007.

7 Reinhard Diestel. *Graph Theory, 5th Edition*. Graduate texts in mathematics. Springer, Berlin, Germany, 2017. doi:10.1007/978-3-662-53622-3.

8 Pål Grønås Drange. *Parameterized Graph Modification Algorithms*. PhD thesis, The University of Bergen, 2015. URL: https://bora.uib.no/bora-xmlui/handle/1956/10774.

9 Jirí Fiala, Jan Kratochvíl, and Andrzej Proskurowski. Systems of distant representatives. *Discrete Applied Mathematics*, 145(2):306–316, 2005. doi:10.1016/j.dam.2004.02.018.

10 Fedor V. Fomin, Petr A. Golovach, Tanmay Inamdar, Saket Saurabh, and Meirav Zehavi. Kernelization for spreading points. In *ESA 2023*, volume 274 of *LIPIcs*, pages 48:1–48:16, 2023. doi:10.4230/LIPICS.ESA.2023.48.

11 Fedor V. Fomin, Petr A. Golovach, Tanmay Inamdar, Saket Saurabh, and Meirav Zehavi. Parameterized geometric graph modification with disk scaling. In *ITCS 2025*, volume 325 of *LIPIcs*, pages 51:1–51:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ITCS.2025.51.

12 Fedor V. Fomin, Saket Saurabh, and Neeldhara Misra. Graph modification problems: A modern perspective. In *Frontiers in Algorithmics*, pages 3–6. Springer International Publishing, 2015. doi:10.1007/978-3-319-19647-3_1.

13 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

14 Marc Hellmuth, Manuela Geiß, and Peter F. Stadler. Complexity of modification problems for reciprocal best match graphs. *Theoretical Computer Science*, 809:384–393, 2020. doi:10.1016/j.tcs.2019.12.033.

15 Hung P. Hoang, Stefan Lendl, and Lasse Wulf. Assistance and interdiction problems on interval graphs. *Discrete Applied Mathematics*, 340:153–170, 2023. doi:10.1016/j.dam.2023.06.046.

16 Nicolás Honorato-Droguett, Kazuhiro Kurita, Tesshu Hanaka, and Hirotaka Ono. Algorithms for optimally shifting intervals under intersection graph models. In *IJTCS-FAW 2024*, volume 14752, pages 66–78. Springer, 2024. doi:10.1007/978-981-97-7752-5_5.

17 Nicolás Honorato-Droguett, Kazuhiro Kurita, Tesshu Hanaka, and Hirotaka Ono. On the complexity of minimising the moving distance for dispersing objects, 2025. doi:10.48550/arXiv.2502.12903.

18 Donald E. Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM Journal on Discrete Mathematics*, 5(3):422–427, 1992. doi:10.1137/0405033.

19 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-Complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

20 Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009. doi:10.1093/bioinformatics/btp324.

21 David Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982. doi:10.1137/0211025.

22 Terry A. McKee and F. R. McMorris. *Topics in Intersection Graph Theory*. Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 1999. doi:10.1137/1.9780898719802.

23 Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Contraction decomposition in unit disk graphs and algorithmic applications in parameterized complexity. *ACM Trans. Algorithms*, 20(2):15, 2024. doi:10.1145/3648594.

24 Franco P. Preparata and Michael Ian Shamos. *Computational Geometry - An Introduction*. Texts and Monographs in Computer Science. Springer, 1985. doi:10.1007/978-1-4612-1098-6.

25 R. Sritharan. Graph modification problem for some classes of graphs. *Journal of Discrete Algorithms*, 38-41:32–37, 2016. doi:10.1016/j.jda.2016.06.003.

26 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984. doi:10.1016/0166-218x(84)90081-7.

A Near-Linear Time Exact Algorithm for the L_1-Geodesic Fréchet Distance Between Two Curves on the Boundary of a Simple Polygon

Thijs van der Horst
Department of Information and Computing Sciences, Utrecht University, The Netherlands
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Marc van Kreveld
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Tim Ophelders
Department of Information and Computing Sciences, Utrecht University, The Netherlands
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Bettina Speckmann
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Abstract

Let P be a polygon with k vertices. Let R and B be two simple, interior disjoint curves on the boundary of P, with n and m vertices. We show how to compute the Fréchet distance between R and B using the geodesic L_1-distance in P in $\mathcal{O}(k \log nm + (n+m)(\log^2 nm \log k + \log^4 nm))$ time.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Fréchet distance, geodesic, simple polygon

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.37

Related Version *Full Version*: https://arxiv.org/abs/2401.14815

Funding *Tim Ophelders*: partially supported by the Dutch Research Council (NWO) under project no. VI.Veni.212.260.

1 Introduction

In a seminal paper from 1992, Alt and Godau [1] introduced the *Fréchet distance* to the algorithms community as a better similarity measure for two curves than, for example, the Hausdorff distance. For two polygonal curves with n and m vertices, they addressed the algorithmic problem of computing the Fréchet distance and presented an $O(nm \log nm)$ time algorithm [2], which is near-quadratic in the input size.

Since then, research efforts have developed globally in two directions: examining variations or generalizations that can still be solved in near-quadratic time, and restrictions or approximations to beat the quadratic time upper bound. In 2014, Bringmann [3] showed that the Fréchet distance cannot be computed to within a factor 1.001 in strongly subquadratic time, unless the strong exponential time hypothesis fails. This result was improved in 2019 by Buchin, Ophelders and Speckmann [5], who showed that this conditional lower bound applies even for curves in 1-dimensional space and for approximations better than a factor 3. On the other hand, Driemel, Har-Peled and Wenk [7] showed that a near-linear time $(1+\varepsilon)$-approximation exists for curves in the plane that satisfy a sparseness condition. A full overview of the results is beyond the scope of this paper.

One of the variations that has been examined concerns computing the Fréchet distance between two curves in a polygonal environment. Intuitively, the Fréchet distance is based on matching points on the two curves in a continuous and forward manner, and considering

© Thijs van der Horst, Marc van Kreveld, Tim Ophelders, and Bettina Speckmann;
licensed under Creative Commons License CC-BY 4.0

the Euclidean distance between matched points. In a polygonal environment, the Euclidean distance is replaced by the Euclidean *geodesic* distance, the length of a shortest path in the environment. In the setting of two curves inside a simple polygon, Cook and Wenk [9] showed that the Fréchet distance can still be computed in near-quadratic time. Recently, it was shown that if the two curves lie on the boundary of a simple polygon, then a $(1+\varepsilon)$-approximation exists that uses near-linear time [11]. In this paper we show that in the same setting, but using the L_1-distance to measure lengths of geodesics rather than the Euclidean distance, an exact algorithm can be given that requires near-linear time.

Preliminaries. Let P be a simple polygon. A *curve* on the boundary of P is a piecewise-linear function $F\colon [0,1] \to \partial P$ connecting a sequence of *vertices*. Consecutive vertices are connected by a straight-line *edge*. The curve is *simple* if the preimage of a point on ∂P is at most an interval. We denote by $F[x,x']$ the subcurve of F over the domain $[x,x']$. We write $|F|$ to denote the number of vertices of F.

A *reparameterization* of $[0,1]$ is a non-decreasing surjection $f\colon [0,1] \to [0,1]$. Two reparameterizations f and g describe a *matching* (f,g) between two curves R (red) and B (blue), where for every $t \in [0,1]$ the point $R(f(t))$ is matched to $B(g(t))$. The matching (f,g) is said to have *cost*

$$\max_t \overline{d}(R(f(t)), B(g(t))),$$

where in this work, $\overline{d}(p,q)$ measures the length of a shortest path in P from p to q under the L_1-norm. We call such a path an *L_1-geodesic*. There are potentially uncountably many L_1-geodesics from p to q. On the other hand, under the L_2-norm, P contains a unique shortest path from p to q, which we denote by $\pi(p,q)$. It turns out that $\pi(p,q)$ is also an L_1-geodesic, so we use it as a representative.

A matching with cost at most δ is called a *δ-matching*. The (continuous) *L_1-geodesic Fréchet distance* $\overline{d}_{\mathrm{F}}(R,B)$ between R and B is the minimum cost over all matchings. The corresponding matching is a *Fréchet matching*.

The *parameter space* of R and B is the axis-aligned rectangle $[0,1]^2$. Any point (x,y) in the parameter space corresponds to the pair of points $R(x)$ and $B(y)$ on the two curves. A point (x,y) in the parameter space is *δ-close* for some $\delta \geq 0$ if $\overline{d}(R(x),B(y)) \leq \delta$. The *$\delta$-free space* $\mathcal{F}_\delta(R,B)$ of R and B is the subset of $[0,1]^2$ containing all δ-close points. A point $q = (x',y') \in \mathcal{F}_\delta(R,B)$ is *δ-reachable* from a point $p = (x,y)$ if $x \leq x'$ and $y \leq y'$, and there exists a bimonotone path in $\mathcal{F}_\delta(R,B)$ from p to q. Alt and Godau [2] observe that there is a one-to-one correspondence between δ-matchings between $R[x,x']$ and $B[y,y']$, and bimonotone paths from p to q through $\mathcal{F}_\delta(R,B)$. We abuse terminology slightly and refer to such paths as δ-matchings.

Problem statement and results. Our input consists of a simple polygon P with k vertices and curves $R,B\colon [0,1] \to \partial P$ with n and m vertices respectively, on the boundary of P. We restrict ourselves to inputs of the following type:

(i) R and B are oppositely oriented, interior-disjoint, simple curves on the boundary of P.

We present an algorithm for computing the Fréchet distance $\overline{d}_{\mathrm{F}}(R,B)$ between R and B, when distances between points are measured as the minimum length of a path in P, under the L_1-norm. For our algorithm, we make two assumptions. Firstly, without loss of generality, assume that R is oriented clockwise with respect to P, and B counter-clockwise. Secondly, we assume general position: no edge of P has slope ± 1, and no two vertices of P have the same y-coordinate. We compute the Fréchet distance between R and B under the geodesic L_1-distance in P in $\mathcal{O}(k \log nm + (n+m)(\log^2 nm \log k + \log^4 nm))$ time.

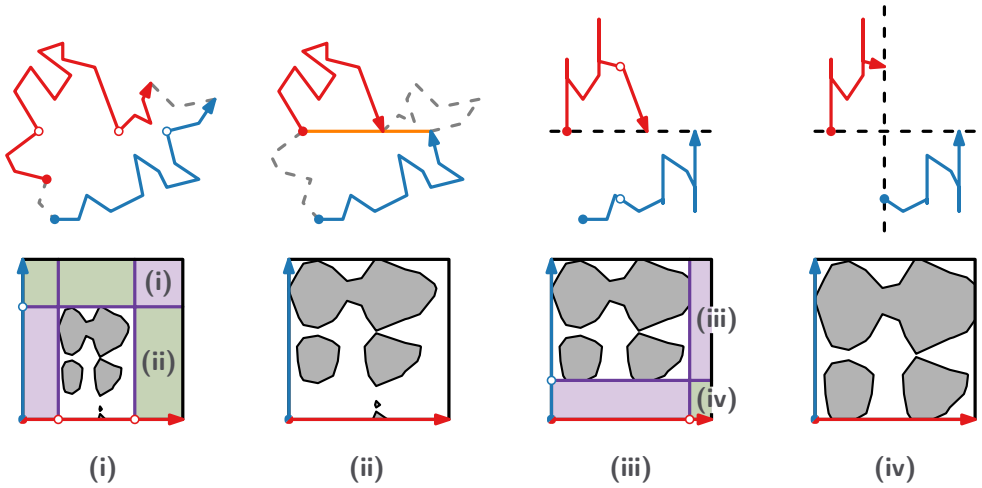


Figure 1 The four types of subproblems that arise in our divide-and-conquer scheme. The bottom row shows the parameter spaces corresponding to the top row. Between problem types, the correspondence between some subcurves and parts of the δ-free space is illustrated.

Algorithmic outline. We first outline the *decision algorithm*: For an additional parameter $\delta \geq 0$, it reports whether $\overline{d}_{\mathrm{F}}(R, B) \leq \delta$. Recall that a δ-matching between R and B corresponds to a bimonotone path in $\mathcal{F}_\delta(R, B)$ from $(0, 0)$ to $(1, 1)$. We use divide-and-conquer to decide whether such a path exists.

We first explain how to divide the problem of type **(i)** into simpler subproblems of four types **(i)**–**(iv)**, illustrated in Figure 1. In Section 3.3 we find a horizontal chord e^* of P that intersects both R and B in at most a constant number of points. We split R and B at these points, resulting in subcurves R_i of R and B_j of B. Each pair (R_i, B_j) corresponds to a simpler subproblem. Such a subproblem is either of type **(i)** where R_i and B_j have at most a constant fraction of the vertices of R and B, or of a new type with extra structure:

(ii) R_i and B_j are separated by a horizontal chord of P, namely e^*.

We handle subproblems of type **(i)** recursively.

We handle subproblems of type **(ii)** in Section 3.2. Suppose that R and B are separated by a horizontal chord e^* of P, such as the subproblems of type **(ii)**. For a point $p \in P$, denote by $NN(p)$ the point on e^* closest to p under the $\overline{d}$ metric. We leverage the property that for all $r \in R$ and $b \in B$ there exists an L_1-geodesic from r to b that goes via $NN(r)$ and $NN(b)$. Using this property, we transform the problem (R, B) of type **(ii)** into an equivalent problem of a new type that is no longer constrained by the polygon:

(iii) a pair of x-monotone[1] curves $(\bar{R}, \bar{B})$ that are separated by a horizontal line.

We handle subproblems of type **(iii)** in Section 3.1. Suppose that R and B are x-monotone curves separated by a horizontal line, such as the subproblems of type **(iii)**. We again split the problem into simpler subproblems. We find a vertical line ℓ that has a constant fraction of the vertices of $R \cup B$ on either side. We split R and B at points on ℓ, resulting in subcurves R_1 and R_2 of R, and B_1 and B_2 of B. Each pair (R_i, B_j) corresponds to a simpler subproblem. Such a subproblem is either of type **(iii)** and handled recursively, or of a new type:

(iv) R_i and B_j are x-monotone curves separated by both a horizontal and a vertical line.

[1] Throughout this work, we consider monotonicity in the weak sense. That is, a curve is x-monotone if every vertical line intersects the curve in at most one subcurve.

The above scheme splits our original problem into several subproblems of type **(iv)**, each of which corresponds to an axis-aligned rectangle in the parameter space of the two input curves. Together, these rectangles partition the parameter space into interior-disjoint regions. Recall that we want to decide whether there exists a bimonotone path from $(0, 0)$ to $(1, 1)$ in the parameter space that lies completely in the δ-free space. To do so, we maintain a set of points in the δ-free space that can be reached by a bimonotone path that starts at $(0, 0)$. If for a subproblem of type **(iv)** we know for its corresponding rectangle a set of reachable "entry" points on the bottom and left sides, then we can compute a sufficient set of "exit" points on the top and right sides that are reachable via at least one entry point. See Section 2 for details. By processing the subproblems of type **(iv)** in the correct order, we compute whether a bimonotone path from $(0, 0)$ to $(1, 1)$ through the δ-free space exists.

In Section 4.2, we use our decision algorithm to compute $\overline{d}_{\mathrm{F}}(R, B)$ exactly, by binary searching over a set of $\tilde{\mathcal{O}}((n + m)^2)$ candidate values and querying the decision algorithm at every step. Computing all candidate values explicitly would take too much time. Instead, we represent them implicitly as the Cartesian sum of two sorted sets X and Y of only $\mathcal{O}((n + m) \log^2 nm)$ values. Using algorithms for linear-time selection in such implicitly represented Cartesian sums [8, 10], the time taken to compute a candidate for δ is dominated by the time taken to perform the decision algorithm. This allows us to compute $\overline{d}_{\mathrm{F}}(R, B)$ with only logarithmic overhead compared to the decision algorithm.

We present our decision algorithm bottom-up, starting with the most specific problem **(iv)** that arises, and ending with the general problem **(i)**, in the following sections.

2 Doubly-separated x-monotone curves

In this section, we consider subproblems of type **(iv)**, where R and B are two x-monotone curves in $\mathbb{R}^2$ separated by both a horizontal line and a vertical line. Let R have n vertices and B have m vertices. We present an algorithm for *propagating reachability information* through the δ-free space of R and B. We are given a parameter $\delta \geq 0$, a set of $\mathcal{O}(n + m)$ points S on the bottom and left sides of the parameter space, and a set of $\mathcal{O}(n + m)$ points T on the top and right sides of the parameter space. The goal is to report the points in T that are δ-reachable from at least one point in S. See Figure 2. In this section, we measure distances under the L_1-norm $\|\cdot\|_1$, in which shortest paths are not restricted by any polygon.

Lemma 1 shows that the distances between R and B can be captured by two curves in one dimension that are separated by the origin, which can be constructed in linear time.

▶ **Lemma 1.** *Let R and B be two doubly-separated curves in $\mathbb{R}^2$ with n and m vertices. In $\mathcal{O}(n + m)$ time, we can construct two curves $\bar{R}$ and $\bar{B}$ in $\mathbb{R}$ that are separated by the origin, such that $\mathcal{F}_\delta(R, B) = \mathcal{F}_\delta(\bar{R}, \bar{B})$ for all δ. The curves $\bar{R}$ and $\bar{B}$ have n and m vertices as well.*

Bringmann and Künnemann [4] and Van der Horst *et al.* [11] both give near-linear time algorithms for propagating reachability information through the free space of two separated curves in one dimension. Combining this with Lemma 1, we obtain:

▶ **Lemma 2.** *Let R and B be two doubly-separated x-monotone curves in $\mathbb{R}^2$ with n and m vertices. Given a parameter $\delta \geq 0$ and sets S and T of $\mathcal{O}(n + m)$ points on the bottom and left, respectively top and right, sides of the parameter space of R and B, we can report the points in T that are δ-reachable from at least one point in S in $\mathcal{O}((n + m) \log nm)$ time.*

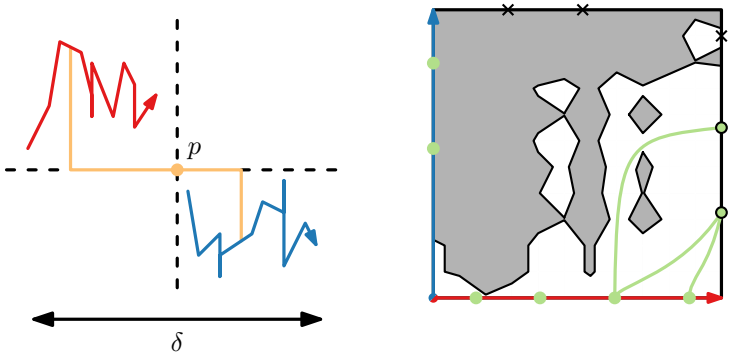


Figure 2 (left) A pair of doubly-separated x-monotone curves. We may consider the curves as lying on the union of two (degenerate) "histogram" polygons, in which case $\overline{d}$ is equal to the L_1-norm. Between any pair of red and blue points, there exists an L_1-geodesic through the point p. (right) Their δ-free space diagram. For propagating reachability, given are a set of points S (disks) on the bottom and left sides of the parameter space, and a set T (circles and crosses) of points on the top and right sides. The filled circles are δ-reachable, the crosses are not.

3 Partitioning the parameter space

In this section, we consider the main problem, of type **(i)**, where R and B are simple, interior-disjoint curves on the boundary of a simple polygon P. We divide this problem into subproblems of type **(iv)** by partitioning the parameter space of R and B into axis-aligned rectangles where either the corresponding subcurves are trivial (i.e., line segments), or form a type **(iv)** instance, in that the subcurves are equivalent to a pair of doubly-separated x-monotone curves.

An axis-aligned rectangle $[x, x'] \times [y, y']$ is *doubly-separated* if, for all δ, the δ-free space $\mathcal{F}_\delta(R[x, x'], B[y, y'])$ inside it is equivalent to the δ-free space of a pair of doubly-separated x-monotone curves $\bar{R}$ and $\bar{B}$, i.e., $\mathcal{F}_\delta(R[x, x'], B[y, y']) = \mathcal{F}_\delta(\bar{R}, \bar{B})$. A *doubly-separated partition* of the parameter space of R and B is a partition into axis-aligned rectangles $[x, x'] \times [y, y']$, where either $[x, x'] \times [y, y']$ is doubly-separated, or both $R[x, x']$ and $B[y, y']$ are line segments. In the latter case, we refer to such a region as *trivial*. A doubly-separated partition always exists, since we can take its regions to be only trivial regions, corresponding to the pairs of individual edges of the curves. Such a partition has high complexity however, having nm regions. Our goal is instead to compute a doubly-separated partition with few trivial regions, and where the doubly-separated regions correspond to subcurves with relatively few vertices in total, so that the total input complexity when applying the result of Lemma 2 is kept low.

To measure the "quality" of a doubly-separated partition, we define the following *costs*. We define the cost $\text{Cost}(\mathcal{R})$ of a trivial region to be $\mathcal{O}(1)$, and the cost of a doubly-separated region $\mathcal{R} = [x, x'] \times [y, y']$ as $\big|R[x, x']\big| + \big|B[y, y']\big|$, the total number of vertices in the two subcurves. We define the cost $\text{Cost}(\mathcal{P})$ of a doubly-separated partition $\mathcal{P}$ as $\text{Cost}(\mathcal{P}) = \sum_{\mathcal{R} \in \mathcal{P}} \text{Cost}(\mathcal{R})$. In this section we construct a doubly-separated partition of only near-linear cost, namely $\mathcal{O}((n + m) \log^2 nm)$.

We construct our partition in three phases. In Section 3.1 we construct a partition for subproblems of type **(iii)**, where the curves R and B are x-monotone and separated by a horizontal line. Then, in Section 3.2, we handle subproblems of type **(ii)**, where R and B are simple curves on the boundary of a simple polygon P, separated by a horizontal line segment in P. Finally, in Section 3.3, we handle subproblems of type **(i)**.

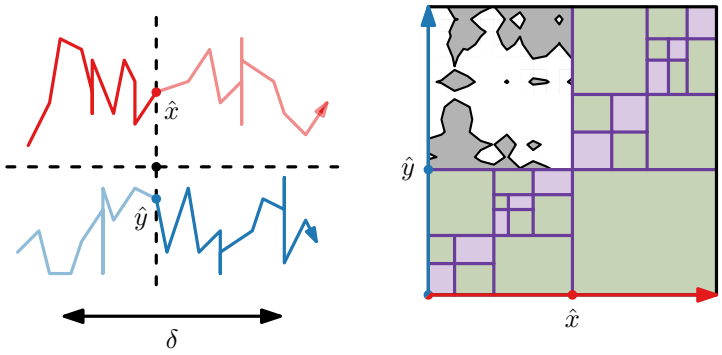


Figure 3 (left) A pair of x-monotone curves separated by a horizontal line, together with a vertical line with roughly half of all vertices on either side. (right) This line represents a partition of the parameter space into four regions, of which the top-left and bottom-right regions correspond to pairs of doubly-separated subcurves. The bottom-left and top-right regions are partitioned further. Green regions signify subproblems of type **(iv)**.

3.1 Vertically-separated x-monotone curves

In this section, we consider the setting where R and B are x-monotone curves in $\mathbb{R}^2$, separated by a horizontal line. We give a simple partition algorithm for the parameter space of R and B that yields a doubly-separated partition of cost $\mathcal{O}((n+m)\log nm)$:

▶ **Lemma 3.** *Let R and B be two vertically-separated x-monotone curves in $\mathbb{R}^2$ with n and m vertices. We can compute a doubly-separated partition of the parameter space of R and B that has cost $\mathcal{O}((n+m)\log nm)$ in $\mathcal{O}((n+m)\log nm)$ time.*

3.2 Separated curves on a simple polygon

Next we reduce the following setting to that of two vertically-separated x-monotone curves: Let P be a simple polygon, and let R and B be two simple curves on the boundary of P, separated by a horizontal line segment in P. That is, there exists a horizontal line segment e^* in P, such that splitting P at e^* creates two subpolygons where R and B lie in different subpolygons.[2] Recall that we assume R to be oriented clockwise with respect to P, and B counter-clockwise. We construct a pair of vertically-separated x-monotone curves whose δ-free space (under the L_1-norm) is identical to that of R and B, for all δ.

Every L_1-geodesic between points $r \in R$ and $b \in B$ intersects e^* in exactly one line segment. Moreover, there exists an L_1-geodesic that is composed of the following three parts: an L_1-geodesic between r and its closest point u on e^*, an L_1-geodesic between b and its closest point v on e^*, and the line segment $\overline{uv}$. For two vertically-separated x-monotone curves (as in Section 3.1), there exists a similar shortest path between two points, namely one that is composed of two vertical segments and a horizontal segment on the separator. We use this connection for our reduction, which we specify next.

We refer to the "projection" of a point $p \in P$ onto e^* (with slope other than ± 1) as the point on e^* closest to p, and we denote this projection by $NN(p)$. Above we established that between any two points $r \in R$ and $b \in B$, there exists an L_1-geodesic that goes through the projections $NN(r)$ and $NN(b)$. We introduce two real-valued functions, which we use to define the coordinates of the x-monotone curves we reduce the problem to.

[2] We consider e^* to be part of both subpolygons.

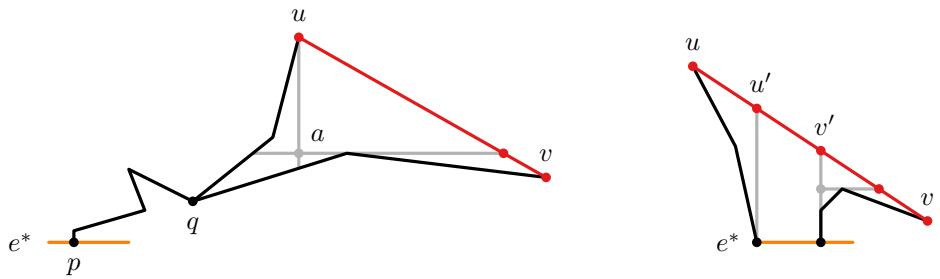


Figure 4 An illustration of the two settings discussed in Lemma 5. On the left is the case where $NN(u) = NN(v)$. On the right is the case where $NN(u) \neq NN(v)$. Marked points on $\overline{uv}$ are points where the functions φ and ψ change pieces.

We assume e^* is parameterized over $[1, 2]$. The first function, $\varphi\colon P \to [1, 2]$, indexes the projections on e^* of the points in P. That is, $e^*(\varphi(p)) = NN(p)$. The second function, $\psi\colon P \to [1, 2]$, represents the distance function from points to their projections, so $\psi(p) = \overline{d}(p, NN(p))$.

We consider the curves $\bar{R}\colon [0, 1] \to \mathbb{R}^2$ and $\bar{B}\colon [0, 1] \to \mathbb{R}^2$ given by $\bar{R}(x) = (\varphi(R(x)), \psi(R(x)))$ and $\bar{B}(y) = (\varphi(B(y)), -\psi(B(y)))$. Note that $\bar{d}(R(x), B(y)) = \|\bar{R}(x) - \bar{B}(y)\|_1$ for all $x, y \in [0, 1]^2$, and so $\mathcal{F}_\delta(R, B) = \mathcal{F}_\delta(\bar{R}, \bar{B})$ for all δ. The curves are separated by the horizontal line $(-\infty, \infty) \times \{0\}$. Next we prove that the curves are x-monotone, making the algorithm developed in Section 3.1 applicable to them:

▶ **Lemma 4.** *The curves $\bar{R}$ and $\bar{B}$ are x-monotone.*

We have shown that the algorithm developed in Section 3.1 can be applied to the curves $\bar{R}$ and $\bar{B}$. Next we show that $\bar{R}$ and $\bar{B}$ have low complexity. Let R have n vertices and B have m vertices. We show that $\bar{R}$ has $\mathcal{O}(n)$ vertices and $\bar{B}$ has $\mathcal{O}(m)$ vertices. While doing so, we develop a data structure that can construct $\bar{R}$ and $\bar{B}$ efficiently.

We construct pieces of $\bar{R}$ and $\bar{B}$ by computing the functions φ and ψ over individual edges of R and B. When restricted to a single line segment e, the function ψ is similar to the "hourglass function" H_{e,e^*} introduced by Cook and Wenk [9]. The hourglass function uses the L_2-norm to measure lengths of shortest paths, and can have $\mathcal{O}(k)$ complexity [9]. Perhaps surprisingly however, we show that since we measure lengths with the L_1-norm, the function ψ has only constant complexity when applied to a single edge. Moreover, the function φ has constant complexity as well.

▶ **Lemma 5.** *Let $\overline{uv} \subseteq P$ be a line segment interior-disjoint from e^*. The functions φ and ψ, restricted to $\overline{uv}$, are piecewise-linear with $\mathcal{O}(1)$ pieces. After preprocessing P in $\mathcal{O}(k)$ time, the functions can be computed over $\overline{uv}$ in $\mathcal{O}(\log k)$ time.*

By querying the data structure of Lemma 5 with every edge of R and B individually and combining the results, we obtain the functions φ and ψ over R and B in $\mathcal{O}((n + m) \log k)$ time, after $\mathcal{O}(k)$-time preprocessing. From this, we extract $\bar{R}$ and $\bar{B}$ in $\mathcal{O}(n + m)$ additional time. The complexities of $\bar{R}$ and $\bar{B}$ are $\mathcal{O}(n)$ and $\mathcal{O}(m)$, respectively. Thus we obtain:

▶ **Lemma 6.** *Let P be a simple polygon with k vertices. Let R and B be two simple curves with n and m vertices on the boundary of P, separated by a horizontal line segment in P. After preprocessing P in $\mathcal{O}(k)$ time, we can construct a pair of vertically-separated x-monotone curves $\bar{R}$ and $\bar{B}$ with $\mathcal{F}_\delta(R, B) = \mathcal{F}_\delta(\bar{R}, \bar{B})$ in $\mathcal{O}((n + m) \log k)$ time. The curves $\bar{R}$ and $\bar{B}$ have $\mathcal{O}(n)$ and $\mathcal{O}(m)$ vertices, respectively.*

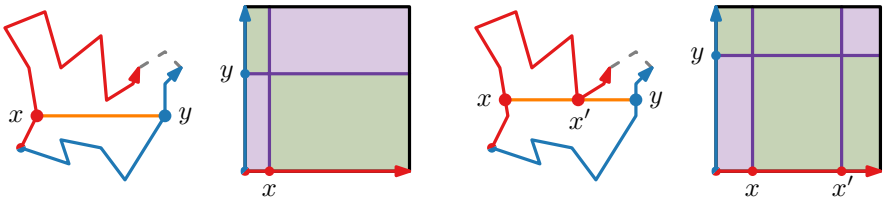


Figure 5 Two types of bichromatic chords and the partitions of the parameter space that they induce. All green regions correspond to separated subcurves; the purple regions are split recursively.

▶ **Corollary 7.** *Let P be a simple polygon with k vertices. Let R and B be two simple curves with n and m vertices on the boundary of P, separated by a horizontal line segment in P. After preprocessing P in $\mathcal{O}(k)$ time, we can compute a doubly-separated partition of the parameter space of R and B that has cost $\mathcal{O}((n+m)\log nm)$ in $\mathcal{O}((n+m)\log nm)$ time.*

3.3 The general case

In this section, we start out in the general setting, where R and B are simple, interior-disjoint curves on the boundary of a simple polygon P, and construct a doubly-separated partition of the parameter space of R and B.

We follow the approach of Section 3.1, in that we partition the parameter space recursively, though this time based on horizontal *chords* of P. A chord is a maximal segment that does not go outside P. By our assumptions, a chord can have at most three points in common with the boundary of P. If possible, we will use bichromatic chords, which have a point in common with both curves. Each such chord e^* splits each curve into at most three subcurves: If e^* intersects R only at an endpoint of R, the chord "trivially" splits R into the curve R itself. Otherwise, R is split into the maximal prefix $R[0,x]$ whose intersection with e^* is only the point $R(x)$, the maximal suffix $R[x',1]$ whose intersection with e^* is only the point $R(x')$, and the maximal subcurve $R[x,x']$ bounded by e^*. See Figure 5. The split of B induced by e^* is defined analogously, but if R is split into three subcurves, then B can be split into at most two subcurves (and vice versa). A chord corresponds to a partition of the parameter space into the axis-aligned rectangles whose corresponding subcurves are induced by the splits of R and B. Thus, a chord corresponds to a partition into at most six regions.

If a bichromatic chord does not exist, then R and B can be separated trivially by a horizontal chord. Our partition algorithm works recursively, like the algorithm of Lemma 3. To bound the recursive depth, we use a chord that splits R and B into at most two and three subcurves each, where the first and last subcurves of each have a total of at most $(n+m)/c$ vertices at either side of the chord, for some constant $c > 1$.

We prove the existence of a horizontal chord with the desired properties, and give a construction algorithm. The result follows from a relaxed version of the "polygon-cutting theorem" of Chazelle [6].

▶ **Lemma 8.** *If R and B are not already separated by a horizontal chord, there exists a horizontal chord that intersects both R and B. Furthermore, if $n+m \geq 3$, such a chord exists that splits R and B into at most three subcurves each, where for each pair (R_i, B_j) of subcurves not separated by the chord, $|R_i| + |B_j| \leq 2(n+m)/3$. Such a chord can be found in $\mathcal{O}(k)$ time.*

With the existence and construction result for good chords at hand, we make an initial partition of the parameter space. The partition is not yet a doubly-separated partition, but we refine it afterwards into one that is.

We recursively partition the parameter space as follows. If $|R| \leq 2$ and $|B| \leq 2$ then we stop partitioning further. Otherwise, we compute a horizontal chord e^* of P with the algorithm of Lemma 8, taking $\mathcal{O}(k)$ time.

Assuming P is in general position, the chord intersects one curve in at most two points and the other in at most one point. Suppose that e^* intersects R in two points, splitting it into the subcurves $R_1 = R[0, x]$, $R_2 = R[x, x']$ and $R_3 = R[x', 1]$, and that e^* intersects B in one point, splitting it into the subcurves $B_1 = B[0, y]$ and $B_2 = B[y, 1]$. These five subcurves together define six axis-parallel rectangles in the parameter space, which is the partition corresponding to e^*.

Let $\mathcal{P}$ be this partition. The subcurve R_2 is bounded by e^*, and thus is separated from all of B by e^*. Further, the subcurve R_1 is separated from B_2, and the subcurve R_3 from B_1. Thus the only regions in the partition whose corresponding subcurves are not separated by a horizontal chord are the bottom-left region $[0, x] \times [0, y]$ and the top-right region $[x', 1] \times [y, 1]$; we recursively partition $\mathcal{P}$ in these regions.

We analyze the partition in the same manner as we do for doubly-separated partitions. That is, we refer to the cost of an axis-aligned rectangle $\mathcal{R} = [x, x'] \times [y, y']$ as $\text{Cost}(\mathcal{R}) = \big|R[x, x']\big| + \big|B[y, y']\big|$, the same as for doubly-separated regions, and refer to the cost of the partition $\mathcal{P}$ as $\text{Cost}(\mathcal{P}) = \sum_{\mathcal{R} \in \mathcal{P}} \text{Cost}(\mathcal{R})$.

▶ **Lemma 9.** *$\mathcal{P}$ has cost $\mathcal{O}((n + m) \log nm)$ and can be computed in $\mathcal{O}(k \log nm)$ time.*

We refine $\mathcal{P}$ into a doubly-separated partition. Some regions in $\mathcal{P}$ already correspond to line segments on R and B, and so are trivial. The other regions correspond to subcurves of R and B that are separated by a horizontal chord of P. We partition these regions further using the result of Corollary 7.

▶ **Lemma 10.** *Let P be a simple polygon with k vertices. Let R and B be two simple, interior-disjoint curves on with n and m vertices on the boundary of P. We can compute a doubly-separated partition of the parameter space of R and B that has cost $\mathcal{O}((n+m) \log^2 nm)$ in $\mathcal{O}(k \log nm + (n+m) \log^2 nm)$ time. Additionally, in the same time bound, we can compute, for each region $[x, x'] \times [y, y']$ in the partition, a pair of doubly-separated x-monotone curves $\bar{R}$ and $\bar{B}$ with $\mathcal{F}_\delta(R[x, x'], B[y, y']) = \mathcal{F}_\delta(\bar{R}, \bar{B})$ for all δ. The curves $\bar{R}$ and $\bar{B}$ have $\mathcal{O}(\big|R[x, x']\big|)$ and $\mathcal{O}(\big|B[y, y']\big|)$ vertices, respectively.*

4 Computing the Fréchet distance

In this section, we discuss the main problem setting of this work. Let P be a simple polygon and let R and B be two simple, interior-disjoint curves on the boundary of P. We assume R is oriented clockwise with respect to P, and B counter-clockwise. We present a near-linear time algorithm for computing $\overline{d}_{\text{F}}(R, B)$.

Our algorithm makes use of a *decision* algorithm that, given a parameter $\delta \geq 0$, reports whether $\overline{d}_{\text{F}}(R, B) \leq \delta$ or $\overline{d}_{\text{F}}(R, B) > \delta$. Recall from the preliminaries that Alt and Godau [2] showed that $\overline{d}_{\text{F}}(R, B) \leq \delta$ if and only if there exists a bimonotone path in $\mathcal{F}_\delta(R, B)$ from $(0, 0)$ to $(1, 1)$. We develop an algorithm that decides whether such a path exists.

In Section 4.1, we use the doubly-separated partition constructed in Section 3 to efficiently propagate reachability information through the various regions. Recall that there are two types of regions in a doubly-separated partition: trivial regions and doubly-separated regions.

For trivial regions, we give a simple algorithm for propagating reachability information through them, using that these regions correspond to pairs of segments on R and B. For the other regions, we use the result of Section 2 instead, to also propagate reachability information through them. This leads to our decision algorithm. In Section 4.2, we turn our decision algorithm into an optimization algorithm.

4.1 The decision algorithm

We present an algorithm for deciding whether $\overline{d}_{\mathrm{F}}(R, B) \leq \delta$ for a given parameter $\delta \geq 0$. Our algorithm decides whether a bimonotone path in $\mathcal{F}_\delta(R, B)$ exists from $(0, 0)$ to $(1, 1)$. Let $\mathcal{P}^*$ be a doubly-separated partition of the parameter space of R and B computed with the algorithm of Lemma 10. We propagate reachability information through each region in $\mathcal{P}^*$ separately.

Before we do so, we need to address one issue. For the regions $\mathcal{R} \in \mathcal{P}^*$ where we have a pair of doubly-separated x-monotone curves $\bar{R}$ and $\bar{B}$ whose free space is identical to that inside $\mathcal{R}$, we wish to apply the result of Lemma 2. However, this algorithm is for propagating reachability from a given discrete set of points to a given discrete set of points. Hence we first construct sets $S_\mathcal{R}$ and $T_\mathcal{R}$ to which we apply the result of Lemma 2.

We define the sets $S_\mathcal{R}$ and $T_\mathcal{R}$. For this, we say that a point $B(x)$ is *locally closest* to $R(y)$ if an infinitesimal perturbation of $B(x)$ while staying on B increases its distance to $R(y)$. We call points on R locally closest to points on B if a symmetric condition holds. The set $T_\mathcal{R}$ is the set of all points (x^*, y^*) on the top and right sides of $\mathcal{R}$ for which at least one of $R(x^*)$ and $B(y^*)$ is a vertex of R or B, or locally closest to the other point.

Let $\mathcal{R} = [x, x'] \times [y, y'] \in \mathcal{P}^*$. The set $T_\mathcal{R}$ is defined as follows. For each edge e of R, we add the point (x^*, y') to $T_\mathcal{R}$ if $x^* \in [x, x']$ and $R(x^*)$ is either a vertex of R or the point on e closest to $B(y')$. Symmetrically, for each edge e of B, we add the point (x', y^*) to $T_\mathcal{R}$ if $y^* \in [y, y']$ and $B(y^*)$ is either a vertex of B or the point on e closest to $R(x')$.

The set $S_\mathcal{R}$ is defined as all points in $S \cap \mathcal{R}$, as well as all points on the bottom and left sides of $\mathcal{R}$ that are in a set $T_{\mathcal{R}'}$ (for an adjacent region $\mathcal{R}'$) and are δ-reachable from $S_{\mathcal{R}'}$. The set $S_\mathcal{R}$ is defined as all points on the bottom and left sides of $\mathcal{R}$ that are in a set $T_{\mathcal{R}'}$ (for an adjacent region $\mathcal{R}'$) and are δ-reachable from $S_{\mathcal{R}'}$. If $\mathcal{R}$ contains the point $(0, 0)$, we set $S_\mathcal{R} = \{(0, 0)\}$ instead.

The usefulness of these sets comes from Lemma 11, from which it follows that matchings can be assumed to enter and leave a region $\mathcal{R}$ through points in $S_\mathcal{R}$ and $T_\mathcal{R}$, respectively.

▶ **Lemma 11.** *There exists a Fréchet matching between R and B where for every matched pair of points (r, b), at least one of r and b is a vertex or locally closest to the other point.*

Naturally, for our decision algorithm, we need to compute these sets efficiently:

▶ **Lemma 12.** *We can compute the set $T_\mathcal{R}$ for all $\mathcal{R} \in \mathcal{P}^*$ in $\mathcal{O}(k + (n + m) \log^2 nm \log k)$ time altogether.*

We are now ready to give our decision algorithm, which decides whether a bimonotone path in $\mathcal{F}_\delta(R, B)$ from $(0, 0)$ to $(1, 1)$ exists, for a given $\delta \geq 0$. First, we compute the doubly-separated partition $\mathcal{P}^*$ of Lemma 10, taking $\mathcal{O}(k \log nm + (n+m) \log^2 nm)$ time. The partition has cost $\mathcal{O}((n+m) \log^2 nm)$, and for each region $[x, x'] \times [y, y']$ in this partition, either $R[x, x']$ and $B[y, y']$ are line segments, or we have additionally computed a pair of doubly-separated x-monotone curves $\bar{R}$ and $\bar{B}$ with $\mathcal{O}(\big|R[x, x']\big|)$ and $\mathcal{O}(\big|B[y, y']\big|)$ vertices, respectively. We compute the sets $T_\mathcal{R}$ for all regions $\mathcal{R} \in \mathcal{P}^*$, taking $\mathcal{O}(k + (n + m) \log^2 nm \log k)$ time. The goal is then to compute, for each set $T_\mathcal{R}$, the subset of points that are δ-reachable from $(0, 0)$.

Consider a region $\mathcal{R} = [x, x'] \times [y, y'] \in \mathcal{P}^*$ and suppose that for each region $\mathcal{R}'$ incident to the bottom or left side of $\mathcal{R}$ we have already computed the subset of $T_{\mathcal{R}'}$ of points that are δ-reachable from $(0, 0)$. The union of these subsets forms the set $S_{\mathcal{R}}$ that serves as the set of "entrances" for $\mathcal{R}$; a point in $T_{\mathcal{R}}$ is δ-reachable from $(0, 0)$ if and only if it is δ-reachable from a point in $S_{\mathcal{R}}$.

If $R[x, x']$ or $B[y, y']$ has more than one edge, we have available a pair of doubly-separated x-monotone curves $\bar{R}$ and $\bar{B}$ with $\mathcal{F}_\delta(R[x, x'], B[y, y']) = \mathcal{F}_\delta(\bar{R}, \bar{B})$. We apply the algorithm of Lemma 2 to these curves, using the set $S_{\mathcal{R}}$ for the "entrances" and the set $T_{\mathcal{R}}$ for the "potential exits." This algorithm reports all points in $T_{\mathcal{R}}$ that are δ-reachable from at least one point in $S_{\mathcal{R}}$ in $\mathcal{O}((|\bar{R}| + |\bar{B}|) \log |\bar{R}||\bar{B}|) = \mathcal{O}(\text{Cost}(\mathcal{R}) \log \text{Cost}(\mathcal{R}))$ time.

If both $R[x, x']$ and $B[y, y']$ are line segments, then we can check in constant time whether a given point in $T_{\mathcal{R}}$ is reachable from a given point in $S_{\mathcal{R}}$, so we can report the set of points in $T_{\mathcal{R}}$ that are δ-reachable from at least one point in $S_{\mathcal{R}}$ in $\mathcal{O}(|S_{\mathcal{R}}| \cdot |T_{\mathcal{R}}|) = \mathcal{O}(1)$ time.

▶ **Lemma 13.** *If both $R[x, x']$ and $B[y, y']$ are line segments, then a point $t \in T_{\mathcal{R}}$ is δ-reachable from a point $s \in S_{\mathcal{R}}$ if and only if $t \in \mathcal{F}_\delta(R[x, x'], B[y, y'])$ and t lies above and to the right of s.*

We now have an algorithm that, given the sets $S_{\mathcal{R}}$ and $T_{\mathcal{R}}$, reports the points in $T_{\mathcal{R}}$ that are δ-reachable in $\mathcal{O}(\text{Cost}(\mathcal{R}) \log \text{Cost}(\mathcal{R}))$ time. Applying this algorithm to all regions in $\mathcal{P}^*$, in some topological order, we eventually decide whether $(1, 1)$ is δ-reachable from $(0, 0)$, since it is included in a set $T_{\mathcal{R}}$. This takes

$$\sum_{\mathcal{R} \in \mathcal{P}^*} \mathcal{O}(\text{Cost}(\mathcal{R}) \log \text{Cost}(\mathcal{R})) = \mathcal{O}(\text{Cost}(\mathcal{P}^*) \log nm) = \mathcal{O}((n + m) \log^3 nm)$$

time. The computations of $\mathcal{P}^*$ and the sets $T_{\mathcal{R}}$, which took $\mathcal{O}(k \log nm + (n+m) \log^2 nm \log k)$ time in total, dominate the running time. However, these computations are independent of the decision parameter δ, and so we view these computations as preprocessing. This gives the following result:

▶ **Lemma 14.** *Let P be a polygon with k vertices. Let R and B be two simple, interior disjoint curves on the boundary of P, with n and m vertices. After preprocessing P, R and B in $\mathcal{O}(k \log nm + (n + m) \log^2 nm \log k)$ time, we can decide, given $\delta \geq 0$, whether $\overline{d}_{\text{F}}(R, B) \leq \delta$ in $\mathcal{O}((n + m) \log^3 nm)$ time.*

4.2 Obtaining an optimization algorithm

In this section, we turn the decision algorithm of the previous section into an algorithm that computes $\overline{d}_{\text{F}}(R, B)$, at the cost of a logarithmic factor in the running time when compared to the decision algorithm. For this, we derive a polynomial number of candidate distances, one of which is the actual Fréchet distance $\overline{d}_{\text{F}}(R, B)$. Specifically, for two sets of real numbers X and Y, their *Cartesian sum*, denoted $X \oplus Y$, is the multiset $X \oplus Y = \{\{x+y \mid x \in X, y \in Y\}\}$. We will show that we can compute two sets X and Y of $\mathcal{O}((n + m) \log^2 nm)$ real numbers each, such that $\overline{d}_{\text{F}}(R, B) \in X \oplus Y$. After computing X and Y in near-linear time, we can use existing techniques [8, 10] for selection in Cartesian sums to binary search over the Cartesian sum $X \oplus Y$ without explicitly computing its $|X| \cdot |Y|$ many elements.

We again make use of the doubly-separated partition $\mathcal{P}^*$ computed with the algorithm of Lemma 10. Additionally, we again use the sets $T_{\mathcal{R}}$ for regions $\mathcal{R} \in \mathcal{P}^*$. Recall that there exists a Fréchet matching between R and B that corresponds to a bimonotone path in the parameter space that, for each region $\mathcal{R} \in \mathcal{P}^*$ that it intersects, goes through a point in $T_{\mathcal{R}}$.

Thus, there exists a pair of points $s \in T_{\mathcal{R}'}$ and $t \in T_{\mathcal{R}}$, for a region $\mathcal{R}'$ and region $\mathcal{R}$ incident to the top or right side of $\mathcal{R}'$, such that the minimum parameter δ for which there exists a δ-matching from s to t is the Fréchet distance.

Take regions $\mathcal{R}'$ and $\mathcal{R}$, with $\mathcal{R}$ incident to the top or right side of $\mathcal{R}'$. Let $s \in T_{\mathcal{R}'}$ and let $t \in T_{\mathcal{R}}$ be above and to the right of s. Let δ^* be the minimum parameter for which there exists a δ^*-matching from s to t. We construct sets $X_{\mathcal{R}}$ and $Y_{\mathcal{R}}$ such that $\delta^* \in X_{\mathcal{R}} \oplus Y_{\mathcal{R}}$.

If the subcurves corresponding to $\mathcal{R}$ both are line segments, then from Lemma 13 we obtain that t is δ-reachable from s, for some $\delta \geq 0$, if and only if both points lie in $\mathcal{F}_\delta(R, B)$. We set $X_{\mathcal{R}}$ to be the set minimum values δ for which points $t \in T_{\mathcal{R}}$ lie in $\mathcal{F}_\delta(R, B)$. The set $Y_{\mathcal{R}}$ is simply set to $\{0\}$. Taken over all regions in $\mathcal{P}^*$, the sets have a total cardinality of $\mathcal{O}(\text{Cost}(\mathcal{P}^*)) = \mathcal{O}((n+m)\log^2 nm)$.

If one of the subcurves corresponding to $\mathcal{R}$ has more than one edge, then we have access to a pair of doubly-separated x-monotone curves whose δ-free space is identical to the δ-free space of R and B inside $\mathcal{R}$. Moreover, by Lemma 1, we can transform these curves into a pair of curves in $\mathbb{R}$ that are separated by the point 0, such that the δ-free space remains the same. Let $\bar{R}$ and $\bar{B}$ be these curves. Alt and Godau [2] observe that δ^* must be one of the following values:

- the minimum value for which s and t lie in $\mathcal{F}_{\delta^*}(\bar{R}, \bar{B})$, or
- the distance between a vertex of $\bar{R}$ (resp. $\bar{B}$) and an edge of $\bar{B}$ (resp. $\bar{R}$), or
- the distance between two vertices of $\bar{R}$ (resp. $\bar{B}$) to an edge of $\bar{B}$ (resp. $\bar{R}$), if the vertices lie at equal distance to the edge.

Since $\bar{R}$ and $\bar{B}$ are curves in $\mathbb{R}$ that are separated by a point, the candidates for δ^* of second and third type coincide. Moreover, the distance from a vertex p of one curve to an edge e of the other curve is also the distance from p to the endpoint of e closest to p, which must be a vertex. Thus, the latter two types of candidates for δ^* can be summarized as all pairwise distances between vertices of $\bar{R}$ and vertices of $\bar{B}$. Exploiting the separation of $\bar{R}$ and $\bar{B}$ further, the pairwise distances between vertices of $\bar{R}$ and vertices of $\bar{B}$ can be represented by the Cartesian sum of two sets $\bar{X}$ and $\bar{Y}$, containing the absolute values of the vertex values of $\bar{R}$ and $\bar{B}$, respectively. We set $X_{\mathcal{R}} \leftarrow \bar{X}$ and $Y_{\mathcal{R}} \leftarrow \bar{Y}$. Taken over all regions in $\mathcal{P}^*$, the sets have a total cardinality of $\mathcal{O}(\text{Cost}(\mathcal{P}^*)) = \mathcal{O}((n+m)\log^2 nm)$.

Let $X = \bigcup_{\mathcal{R}\in\mathcal{P}^*} X_{\mathcal{R}}$ and $Y = \bigcup_{\mathcal{R}\in\mathcal{P}^*} Y_{\mathcal{R}}$. These sets have a total cardinality of $\mathcal{O}((n+m)\log^2 nm)$, and contain values $x^* \in X$ and $y^* \in Y$ such that $\overline{d}_{\text{F}}(R, B) = x^* + y^*$. Next we search over $X \oplus Y$ to find the exact value of $\overline{d}_{\text{F}}(R, B)$.

After sorting X and Y, we can compute the i^{th} smallest value in $X \oplus Y$, for any given i, in $\mathcal{O}(|X| + |Y|) = \mathcal{O}((n+m)\log^2 nm)$ time [8, 10]. There are $|X| \cdot |Y|$ values in $X \oplus Y$. We binary search over the integers $1, \ldots, |X| \cdot |Y|$, and at each considered integer i, we compute the i^{th} smallest value δ in $X \oplus Y$. Then we use our decision algorithm to decide whether $\overline{d}_{\text{F}}(R, B) \leq \delta$ and to guide the search to the value $\delta^* \in X \oplus Y$ with $\delta^* = \overline{d}_{\text{F}}(R, B)$.

In the above search procedure, each step takes $\mathcal{O}((n+m)\log^3 nm)$ time after preprocessing P and the curves R and B for decision queries, which takes $\mathcal{O}(k \log nm + (n+m)\log^2 nm \log k)$ time (Lemma 14). We perform $\mathcal{O}(\log(|X| \cdot |Y|)) = \mathcal{O}(\log nm)$ steps. Thus we obtain our main result:

▶ **Theorem 15.** *Let P be a polygon with k vertices. Let R and B be two simple, interior disjoint curves on the boundary of P, with n and m vertices. We can compute $\overline{d}_{\text{F}}(R, B)$ in $\mathcal{O}(k \log nm + (n+m)(\log^2 nm \log k + \log^4 nm))$ time.*

References

1 Helmut Alt and Michael Godau. Measuring the resemblance of polygonal curves. In *Proc. Eighth Annual Symposium on Computational Geometry (SoCG)*, pages 102–109. ACM, 1992. doi:10.1145/142675.142699.

2 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5:75–91, 1995. doi:10.1142/S0218195995000064.

3 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proc. 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 661–670, 2014. doi:10.1109/FOCS.2014.76.

4 Karl Bringmann and Marvin Künnemann. Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds. *International Journal of Computational Geometry & Applications*, 27(1-2):85–120, 2017. doi:10.1142/S0218195917600056.

5 Kevin Buchin, Tim Ophelders, and Bettina Speckmann. SETH says: Weak Fréchet distance is faster, but only if it is continuous and in one dimension. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2887–2901, 2019. doi:10.1137/1.9781611975482.179.

6 Bernard Chazelle. A theorem on polygon cutting with applications. In *proc. 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 339–349. IEEE Computer Society, 1982. doi:10.1109/SFCS.1982.58.

7 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, 2012. doi:10.1007/s00454-012-9402-z.

8 Greg N. Frederickson and Donald B. Johnson. Generalized selection and ranking: Sorted matrices. *SIAM Journal of Computing*, 13(1):14–30, 1984. doi:10.1137/0213002.

9 Atlas F. Cook IV and Carola Wenk. Geodesic Fréchet distance inside a simple polygon. *ACM Transactions on Algorithms*, 7(1):9:1–9:19, 2010. doi:10.1145/1868237.1868247.

10 Andranik Mirzaian and Eshrat Arjomandi. Selection in X+Y and matrices with sorted rows and columns. *Information Processing Letters*, 20(1):13–17, 1985. doi:10.1016/0020-0190(85)90123-1.

11 Thijs van der Horst, Marc van Kreveld, Tim Ophelders, and Bettina Speckmann. The geodesic Fréchet distance between two curves bounding a simple polygon. *CoRR*, abs/2501.03834, 2025. doi:10.48550/arXiv.2501.03834.

Clustering Point Sets Revisited

Md. Billal Hossain
Department of Computer Science, University of Texas at Dallas, TX, USA

Benjamin Raichel
Department of Computer Science, University of Texas at Dallas, TX, USA

Abstract

In the sets clustering problem one is given a collection of point sets $\mathcal{P} = \{P_1, \dots P_m\}$ in $\mathbb{R}^d$, where for any set of k centers in $\mathbb{R}^d$, each P_i is assigned to its nearest center as determine by some local cost functions. The goal is then to select a set of k centers to minimize some global cost function of the corresponding local assignment costs. Specifically, we consider either summing or taking the maximum cost over all P_i, where for each P_i the cost of assigning it to a center c is either $\max_{p \in P_i} \|c - p\|$, $\sum_{p \in P_i} \|c - p\|$, or $\sum_{p \in P_i} \|c - p\|^2$.

Different combinations of the global and local cost functions naturally generalize the k-center, k-median, and k-means clustering problems. In this paper, we improve the prior results for the natural generalization of k-center, give the first result for the natural generalization of k-means, and give results for generalizations of k-median and k-center which differ from those previously studied.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Clustering, k-center, k-median, k-means

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.38

Funding Work on this paper was partially supported by NSF CAREER AF Award CCF-1750780 and NSF AF Award CCF-2311179.

1 Introduction

Clustering is one of the most well studied problems in computational geometry and computer science as a whole, with a variety of applications. For center based clustering, given a point set $P \subset \mathbb{R}^d$ and integer parameter $k > 0$, the goal is to select a set $C \subset \mathbb{R}^d$ of k *centers*, so as to minimize some cost function of the distances determined by assigning each point in P to its nearest center in C. Specifically, for any $p \in P$, let $\|p - C\|$ denote the distance from p to its nearest center in C. Then the *k-center*, *k-median*, and *k-means* objectives are to find the set C minimizing $\max_{p \in P} \|p - C\|$, $\sum_{p \in P} \|p - C\|$, and $\sum_{p \in P} \|p - C\|^2$, respectively.

k-center, k-median, and k-means clustering are all known to be NP-hard. k-center is NP-hard to approximate within any factor less than 2 in general metric spaces [13], and even in the plane is still hard to approximate within a factor of roughly 1.82 [7]. Conversely, both the standard greedy algorithm by Gonzalez [10] and the alternative scooping method by Hochbaum and Shmoys [12] achieve a 2-approximation for k-center. The k-median and k-means problems are both known to be hard to approximate within a $1 + \gamma$ factor for some constant $\gamma > 0$, even in $O(\log n)$ dimensional Euclidean space. Different values of γ are known depending on whether the points are in low or high dimensional Euclidean space, some other metric norm, or a general metric space. Conversely, there are several constant factor approximation algorithms for both problems. For a more in depth discussion see [5] and references therein.

For points in $\mathbb{R}^d$, PTAS's exist for these center based clustering problems when k and d are bounded. For k-center, Agarwal and Procopiuc [1] gave a $(1 + \varepsilon)$-approximation in $O(n \log k) + (k/\varepsilon)^{O(k^{1-1/d})}$ time. For k-median and k-means, Har-Peled and Mazumdar [11] used coresets to achieve a $(1 + \varepsilon)$-approximation algorithm whose running time is linear in n.

© Md. Billal Hossain and Benjamin Raichel;
licensed under Creative Commons License CC-BY 4.0

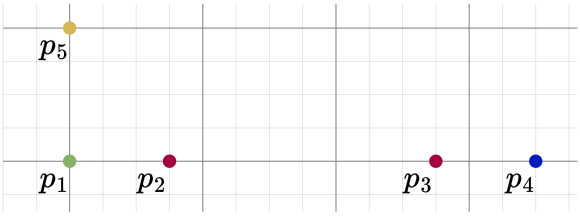


Figure 1.1 Example showing sets clustering differs from regular clustering, where $\mathcal{P} = \{\{p_1\}, \{p_2, p_3\}, \{p_4\}, \{p_5\}\}$, $P = \{p_1, p_2, p_3, p_4, p_5\}$, and $k = 3$. For k-center/means/median clustering on P, the optimal centers are p_5, the midpoint of (p_1, p_2) and the midpoint of (p_3, p_4). However, the optimal centers for the k-center/means/median sets clustering variants on $\mathcal{P}$ that we consider (that is, $cost_{\infty,\infty}/cost_{1,2}/cost_{1,\infty}$) are the midpoint of (p_1, p_5), the midpoint of (p_2, p_3), and p_4.

Subsequent coreset based papers improve the time dependency on k, d, and ε [4, 8]. Using a sampling based approach, [17] provided a $(1+\varepsilon)$-approximation with probability $\geq 1/2$ in $O(2^{(k/\varepsilon)^{O(1)}} dn)$ time, where the probability can be improved by boosting. In general, there are many approximation schemes for k-median and k-means in Euclidean settings, though we note a linear dependence on d and n is possible (e.g. [17]), and for k-median specifically a polynomial dependence on k is possible for bounded d (e.g. [11]).

Sets Clustering

While standard clustering focuses on clustering points, naturally one can consider clustering more general geometric objects. In this paper, we consider the *sets clustering problem* [20], where one must cluster a collection of sets, $\mathcal{P} = \{P_1, \ldots P_m\}$, of total size $n = \sum_i |P_i|$. Here each P_i can for example be viewed as a sample from a given object of study, whether say a physical object in 3d, or some class in a high dimensional feature space. With the interpretation that all points in P_i arise from the same object, we thus naturally require they all be covered by the same center.

Given a center $c \in C$, we let $f(c, P_i)$ denote the cost of assigning P_i to c, and let $f(C, P_i) = \min_{c \in C} f(c, P_i)$. Here we consider the cost functions $f_\infty(c, P_i) = \max_{p \in P_i} \|c - p\|$, $f_1(c, P_i) = \sum_{p \in P_i} \|c - p\|$, and $f_2(c, P_i) = \sum_{p \in P_i} \|c - p\|^2$. Note these cost functions are the 1-center, 1-median, and 1-means costs, and thus are motivated by the applications of those respective problems. However, unlike the 1-center/median/means problems, here we must cluster multiple sets and thus we need an aggregate cost. We consider two possibilities for the cost of clustering $\mathcal{P}$, namely $cost_{\infty,\beta}(C, \mathcal{P}) = \max_i f_\beta(C, P_i)$ and $cost_{1,\beta}(C, \mathcal{P}) = \sum_i f_\beta(C, P_i)$. Observe that when $\mathcal{P}$ is a collection of singleton points that $cost_{\infty,\infty} = cost_{\infty,1}$, $cost_{1,\infty} = cost_{1,1}$, and $cost_{1,2}$ respectively capture the k-center, k-median, and k-means objectives on the point set consisting of these singletons. Conversely, Figure 1.1 shows how when the sets in $\mathcal{P}$ are not singletons, the optimal solutions of the variants of sets clustering that we consider differ from the k-center, k-median, and k-means solutions on $\cup_i P_i$.

Several prior works considered different variants of the sets clustering problem, most notably [20]. Our aim is both to improve prior results, as well as to consider new variants of the sets clustering problem. We break our discussion into whether a particular sets clustering problem is most naturally viewed as a generalization of k-center, k-means, or k-median.

- **k-center:** Minimizing $cost_{\infty,\infty}(C, \mathcal{P})$ naturally generalizes the k-center problem. Previously, for points in constant dimensions, [20] provided an $O(n + m \log k)$ time 3-approximation as well as an $O(nk)$ time $(1 + \sqrt{3})$-approximation. These results require minimum enclosing ball (MEB) computations, and thus have running times with hidden

constants depending on d, though they remark that for higher dimensions they can use a $(1+\varepsilon)$-approximate MEB at the cost of $+\varepsilon$ in the approximation quality.

In Section 3, for points in any dimension d, we show that a natural adaptation of the standard $O(dnk)$ time greedy 2-approximation algorithm for k-center remains an $O(dnk)$ time 2-approximation for the sets clustering problem, a fact which surprisingly appears to have been overlooked in prior work. Moreover, our 2-approximation algorithm works in any metric space, and thus is optimal for general metric spaces (as it captures k-center, which is hard to approximate within a factor of 2). Additionally, for any constant d, using a standard grid based approach we provide an $O(nk^{k+1}/\varepsilon^{dk})$ time $(1+\varepsilon)$-approximation, that is, a linear time algorithm when k, d, and ε are constants.

- **k-means:** Minimizing $cost_{1,2}(C,\mathcal{P})$ naturally generalizes the k-means problem, and we study this objective in Section 4. Surprisingly, it appears this cost function for sets clustering has not been studied in prior works, despite the ubiquity of k-means clustering. Consider the point set P obtained by replacing each set P_i with $|P_i|$ copies of its centroid. Using previously observed facts about the geometry of the k-means objective, we prove the strong statement that an α-approximation to the k-means clustering problem on P is also an α-approximation for the sets clustering problem.
- **k-median:** Minimizing $cost_{1,1}(C,\mathcal{P})$ generalizes the k-median problem. Previously, [20] provided a polynomial time $(3+\varepsilon)$ approximation in $\mathbb{R}^d$ for constant d, by replacing each set P_i with $|P_i|$ copies of its $(1+\varepsilon)$-approximate 1-median, and then applying a $(1+\varepsilon)$-approximation to k-median for constant d (such as [11]). In Section 5, we instead consider the problem of minimizing $cost_{1,\infty}(C,\mathcal{P})$, which is also equivalent to k-median when $\mathcal{P}$ consists of singletons.[1] Minimizing $cost_{1,\infty}(C,\mathcal{P})$ models the case where how well a center covers a set is determined by the furthest point in the set from the center (i.e. the minimum radius ball at the center enclosing the whole set). To address this problem we apply a similar approach as [20], replacing each set with the center of its minimum enclosing ball. We argue that this provides a $(1+\alpha)$-approximation where α is the approximation quality of the k-median subroutine.
- **k-center Alternative:** In Section 6 we considered the problem of minimizing $cost_{\infty,1}(C,\mathcal{P})$. Similar to the $cost_{\infty,\infty}$ considered in Section 3 and prior work, this problem also generalizes standard k-center clustering. This problem is also related to the $cost_{1,\infty}$ problem considered in Section 5, but where we inverted the max and sum operators. This problem is more challenging, though we still can provide a polynomial time $(3+\varepsilon)$-approximation for point sets in the plane and for any constant $0<\varepsilon\leq 1$.

We emphasize that the algorithms in sections 3, 4, and 5 for the $cost_{\infty,\infty}, cost_{1,2}$, and $cost_{1,\infty}$ objectives, either modify or directly reduce to the algorithms respectively used for k-center, k-means, and k-median. Thus our running times are virtually equivalent to the algorithms used for these standard clustering problems, which is the best one could hope for as these standard clustering problems are special cases of our sets clustering problems. On the other hand, our polynomial time approximation for the more challenging $cost_{1,\infty}(C,\mathcal{P})$ in Section 6 is slower, and we leave it as an open problem for future work to optimize the time.

We study the combinations of maximums, sums, and sums of squares which we believe to be the most natural, though other combinations exist that we do not study, such as squaring sums of squares. Several prior works also considered clustering based on the closest point in a

[1] There is also a loose connection between $cost_{1,\infty}(C,\mathcal{P})$ and clustering to minimize the sum of radii [3], though unlike k-median there is no immediate reduction.

set to its nearest center. Specifically, let $f_0(C,P) = \min_{c\in C, p\in P} \|c-p\|$, then [15] considered minimizing $\sum_{P_i\in\mathcal{P}} f_0(C,P_i)^2$, providing a near linear time $(1+\varepsilon)$-approximation for the case when k, d, and z are constants, where $z = \max_i |P_i|$. [14] considered the problem of minimizing $\max_{P_i\in\mathcal{P}} f_0(C,P_i)$, where the P_i sets are continuous convex regions, providing a number of results such as a $(5+2\sqrt{3})$-approximation for the case of disks. Many other papers have considered clustering other continuous geometric objects, and in particular unbounded objects such as lines or hyperplanes [9, 18, 6], though such results are less relevant than the above mentioned papers on clustering discrete sets.

2 Setup

For two points $p, q \in \mathbb{R}^d$, we use $\|p-q\|$ to denote the Euclidean distance between p and q. Similarly, for a point $q\in\mathbb{R}^d$ and a finite point set $P\subset\mathbb{R}^d$ we have $\|q-P\| = \min_{p\in P}\|q-p\|$.

Given a point set $P\subset\mathbb{R}^d$ and a set of k centers $C\subset\mathbb{R}^d$, the standard cost functions for k-center, k-median, and k-means clustering are as follows.

- $kcenter(C,P) = \max_{p\in P}\|p-C\|$
- $kmedian(C,P) = \sum_{p\in P}\|p-C\|$
- $kmeans(C,P) = \sum_{p\in P}\|p-C\|^2$

Given a parameter k and point set $P\subset\mathbb{R}^d$, the goal of k-center, k-median, or k-means clustering is then to find a set C of k centers minimizing the respective cost function.

We now generalize these notions to collections of point sets. So let $\mathcal{P} = \{P_1, \ldots P_m\}$ be a collection of m sets of points, where $P_i\subset\mathbb{R}^d$ for all i, and $n = \sum_i |P_i|$ is the total size. For a point $c\in\mathbb{R}^d$, let $f_\beta(c,P_i)$ be some non-negative function, representing the cost of assigning point set P_i to a center c. We consider three cost functions denoted by $\beta = \infty$, 1, or 2.[2]

- $f_\infty(c,P_i) = \max_{p\in P_i}\|c-p\|$
- $f_1(c,P_i) = \sum_{p\in P_i}\|c-p\|$
- $f_2(c,P_i) = \sum_{p\in P_i}\|c-p\|^2$

For a single point p we write $f_\beta(c,p) = f_\beta(c,\{p\})$, where in particular we have $f_\infty(c,p) = f_1(c,p) = \|c-p\|$ and $f_2(c,p) = \|c-p\|^2$

Given a set $C = \{c_1,\ldots,c_k\}\subset\mathbb{R}^d$ of k centers, define $f_\beta(C,P_i) = \min_{c\in C} f_\beta(c,P_i)$, that is P_i is assigned to its nearest center under the function f_β. Now the $f_\beta(C,P_i)$ values over all i define a vector of length m, and we consider either the ℓ_∞ or ℓ_1 norm. Namely, we define the cost functions, $cost_{\alpha,\beta}$, for $\alpha = \infty$ or 1 as follows:

- $cost_{\infty,\beta}(C,\mathcal{P}) = \max_i f_\beta(C,P_i)$
- $cost_{1,\beta}(C,\mathcal{P}) = \sum_i f_\beta(C,P_i)$

Finally, let $optcost_{\alpha,\beta}(k,\mathcal{P}) = \min_{C\subseteq\mathbb{R}^d,|C|=k} cost_{\alpha,\beta}(C,\mathcal{P})$, and let $opt_{\alpha,\beta}(k,\mathcal{P})$ denote a set C of k centers achieving $optcost_{\alpha,\beta}(k,\mathcal{P})$. For some $\gamma\geq 1$, a set $C\subset\mathbb{R}^d$ with $|C| = k$ is referred to as a *γ-approximation* to $opt_{\alpha,\beta}(k,\mathcal{P})$ if $cost_{\alpha,\beta}(C,\mathcal{P}) \leq \gamma\cdot optcost_{\alpha,\beta}(k,\mathcal{P})$. (That is, k is a hard constraint and the approximation is on the cost, as is standard for clustering.)

▶ **Observation 1.** *Given a point set P, let $\mathcal{P}(P)$ denote the collection of $|P|$ singleton sets $\{p\}$ for each $p\in P$. Then $kcenter(C,P) = cost_{\infty,\infty}(C,\mathcal{P}(P)) = cost_{\infty,1}(C,\mathcal{P}(P))$, $kmedian(C,P) = cost_{1,\infty}(C,\mathcal{P}(P)) = cost_{1,1}(C,\mathcal{P}(P))$, and $kmeans(C,P) = cost_{1,2}(C,\mathcal{P}(P))$.*

[2] The different β values are supposed to be vaguely reminiscent of corresponding vector norms.

3 k-center Sets Clustering

Given a collection of point sets $\mathcal{P} = \{P_1, \dots P_m\}$ in $\mathbb{R}^d$ and a parameter k, [20] considered the problem of covering $\mathcal{P}$ with k equal radius balls of minimum radius such that each P_i is entirely contained in one of the balls. Using the terminology defined above, this is equivalent to computing $opt_{\infty,\infty}(k, \mathcal{P})$. For this problem, [20] gave a $(1 + \sqrt{3})$-approximation. Here we show that the standard greedy Gonzalez algorithm for k-center clustering can be adapted to yield a 2-approximation. Not only is this a better approximation ratio, but this algorithm will in fact work for any metric space. Moreover, for general metrics it is not possible to get a $2 - \varepsilon$ approximation for k-center clustering for any $\varepsilon > 0$, unless P=NP [13]. As k-center is a special case of our problem, our approximation algorithm is thus optimal.

Given a point set P and a parameter k, we recall that the Gonzalez algorithm [10] greedily outputs a set of k center $c_1, \dots, c_k$, where c_1 is an arbitrary point in P, and for $i > 1$, $c_i = \arg\max_{p \in P} \|p - \{c_1, \dots, c_{i-1}\}\|$. We argue the following natural adaptation of this algorithm to our sets clustering problem, which greedily picks the furthest set rather than the furthest point, also yields a 2-approximation. (Figure 1.1 shows that naively running Gonzalez on $P = \cup_i P_i$ does not achieve a 2-approximation. Namely, Gonzalez on P might pick p_1, p_4, and p_5 as the centers, resulting in a cost of 11 for sets clustering on $\mathcal{P}$, however, choosing p_1, the midpoint of (p_2, p_3), and p_4, gives a cost of 4.)

Algorithm 1 **greedy**$(k, \mathcal{P} = \{P_1, \dots, P_m\})$.

```
Initialize all point sets P_1, P_2, ..., P_m as unmarked.
Initialize C = {c_1}, where c_1 is an arbitrary point from P_1, and mark P_1
for i = 2 to k do
    P' = arg max_{unmarked P ∈ P} f_∞(C, P)
    Set C = C ∪ {c'} where c' is an arbitrary point from P', and mark P'.
return C
```

▶ **Theorem 2.** *Given a parameter k and a collection of point sets $\mathcal{P} = \{P_1, \dots P_m\}$ in $\mathbb{R}^d$,* **greedy**$(k, \mathcal{P})$ *gives an $O(dnk)$ time 2-approximation for computing $opt_{\infty,\infty}(k, \mathcal{P})$, where $n = \sum_i |P_i|$.*

Proof. Let $C = \{c_1, \dots, c_k\}$ be the output of **greedy**$(k, \mathcal{P})$ and let $C^* = \{c_1^*, c_2^*, \dots, c_k^*\} = opt_{\infty,\infty}(k, \mathcal{P})$ with cost $r^* = optcost_{\infty,\infty}(k, \mathcal{P}) = \min_{C \subseteq \mathbb{R}^d, |C|=k} cost_{\infty,\infty}(C, \mathcal{P})$, where recall $cost_{\infty,\infty}(C, \mathcal{P}) = \max_i f_\infty(C, P_i)$.

There are two possible cases. In the first case, for all $c^* \in C^*$, there exists some $c \in C$ such that $\|c^* - c\| \leq r^*$. Consider any set P_i with c^* as its nearest optimal center under f_∞, that is $f_\infty(c^*, P_i) = f_\infty(C^*, P_i)$. By definition, $f_\infty(c^*, P_i) = \max_{p \in P_i} \|c^* - p\| \leq r^*$. Thus for any point $p \in P_i$, by the triangle inequality, $\|p - c\| \leq \|p - c^*\| + \|c^* - c\| \leq 2r^*$. Therefore, $f_\infty(c, P_i) = \max_{p \in P_i} \|c - p\| \leq 2r^*$. This in turn implies $f_\infty(C, P_i) \leq 2r^*$, and as this holds for all i it implies C is a 2-approximation.

In the second case, there exists some $c^* \in C^*$, such that for any $c \in C$ we have $\|c^* - c\| > r^*$. For a given $c \in C$, let $P(c)$ denote the set $P_i \in \mathcal{P}$ from which c was selected in **greedy**$(k, \mathcal{P})$. Observe that if $\|c^* - c\| > r^*$ then, $f_\infty(c^*, P(c)) > r^*$, that is $P(c)$ is not covered by c^* in the optimal solution. However, $f_\infty(C^*, P_i) \leq r^*$ for all i, and so by the pigeon hole principle if such a $c^* \in C^*$ occurs then there must exist some other $c_j^* \in C^*$ and centers $c_\alpha, c_\beta \in C$ with $\alpha < \beta$ such that $f_\infty(c_j^*, P(c_\alpha)), f_\infty(c_j^*, P(c_\beta)) \leq r^*$. This implies $\|c_j^* - p\| \leq r^*$ for any $p \in P(c_\alpha) \cup P(c_\beta)$. Thus by the triangle inequality, for any $p \in P(c_\beta)$ we have $\|c_\alpha - p\| \leq \|c_\alpha - c_j^*\| + \|c_j^* - p\| \leq 2r^*$, which implies $f_\infty(c_\alpha, P(c_\beta)) \leq 2r^*$.

For any $1 \leq i \leq k$, let $C_i = \{c_1, \ldots, c_i\}$, i.e. the first i centers chosen by $\mathbf{greedy}(k, \mathcal{P})$, and let $\delta_i = f_\infty(C_{i-1}, P(c_i))$ (where $\delta_1 = \infty$). Denote the cost of the final solution C output by $\mathbf{greedy}(k, \mathcal{P})$ as $\delta_{k+1} = \max_{P \in \mathcal{P}} f_\infty(C, P)$. Observe that $\delta_1 \geq \delta_2 \geq \ldots \geq \delta_k \geq \delta_{k+1}$. Above we argued that $f_\infty(c_\alpha, P(c_\beta)) \leq 2r^*$ where $\alpha < \beta$, thus $\delta_{k+1} \leq \delta_\beta = f_\infty(C_{\beta-1}, P(c_\beta)) \leq f_\infty(C_\alpha, P(c_\beta)) \leq f_\infty(c_\alpha, P(c_\beta)) \leq 2r^*$, and thus C is a 2-approximation.

As for the running time, we can achieve $O(dnk)$ time in a similar fashion to the standard k-center Gonzalez algorithm. Specifically, let C_i be the set of centers after some i iterations of the algorithm, and assume for all unmarked $P \in \mathcal{P}$ we maintain $f_\infty(C_i, P)$. Then in the $i+1$ round, c_{i+1} is some arbitrary point from $\arg\max_{\text{unmarked } P \in \mathcal{P}} f_\infty(C_i, P)$, which can thus be found by a linear scan of the $f_\infty(C_i, P)$ values. We also need to update the $f_\infty(C_i, P)$ values to $f_\infty(C_{i+1}, P)$. To do so observe that for any $P \in \mathcal{P}$, $f_\infty(C_{i+1}, P) = \min\{f_\infty(C_i, P), f_\infty(c_{i+1}, P)\}$. As $f_\infty(c_{i+1}, P)$ can be computed in $O(d|P|)$ time, we can thus update all of these values in $O(dn)$ time. As the algorithm performs k iterations, it thus takes $O(dnk)$ time overall. ◀

The algorithm used in [20] for achieving the $O(1+\sqrt{3})$-approximation requires computing the minimum enclosing ball (MEB) for each point set, and thus has a hidden constant in the running time depending on d. Thus for high dimensions, [20] use a $(1+\varepsilon)$-approximate MEB (which can be computed in $O(dn/\varepsilon + 1/\varepsilon^5)$ time [2]), resulting in a $O(1+\sqrt{3}+\varepsilon)$-approximation factor. In comparison, our algorithm avoids computing the MEB altogether, and as the above proof only relied on the triangle inequality, the same proof verbatim yields a 2-approximation for any metric space.[3]

k-center clustering is known to be hard to approximate within a factor of roughly 1.82 even in the plane [7], however, a $(1+\varepsilon)$-approximation is possible in constant dimensions, though at the cost of having a time exponential in k. We now show this standard algorithm also applies to our problem.

We will make use of the following standard observation (adapted for our settings).

▶ **Observation 3.** *Let $C = \{c_1, \ldots, c_k\} \subset \mathbb{R}^d$ be a set of k points, and let $C' = \{c'_1, \ldots, c'_k\}$ be a set of k points such that $\|c_i - c'_i\| \leq x$. Then for any collection of point sets $\mathcal{P} = \{P_1, \ldots P_m\}$, we have that $cost_{\infty,\infty}(C', \mathcal{P}) \leq cost_{\infty,\infty}(C, \mathcal{P}) + x$. This follows since if for a given set $P_j \in \mathcal{P}$ its nearest center under f_∞ was c_i, then by the triangle inequality we have that $f_\infty(c'_i, P_j) \leq f_\infty(c_i, P_i) + x$.*

▶ **Theorem 4.** *Given a parameter k and a collection of point sets $\mathcal{P} = \{P_1, \ldots P_m\}$ in $\mathbb{R}^d$, for constant d, then for any $\varepsilon > 0$ there is an $O(nk^{k+1}/\varepsilon^{dk})$ time $(1+\varepsilon)$-approximation for computing $opt_{\infty,\infty}(k, \mathcal{P})$, where $n = \sum_i |P_i|$.*

Proof. Let $C = \{c_1, c_2, \ldots, c_k\}$ be the centers output by $\mathbf{greedy}(k, \mathcal{P})$, and let $r = cost_{\infty,\infty}(C, \mathcal{P})$. Also let $C^* = \{c^*_1, c^*_2, \ldots, c^*_k\} = opt_{\infty,\infty}(k, \mathcal{P})$ where $r^* = cost_{\infty,\infty}(C^*, \mathcal{P})$. By Theorem 2 we know that $r^* \leq r \leq 2r^*$.

Let $\mathcal{B}(x) = \cup_i B(c_i, x)$ where $B(c_i, x)$ is the ball of radius x centered at c_i. By definition of C and r, we know $\cup_i P_i \subseteq \mathcal{B}(r)$. We can assume that every center $c^* \in C^*$ is within distance $r^* \leq r$ of some point in $\cup_i P_i$, as otherwise c^* can be deleted without affecting the optimal solution quality. Thus we have that $C^* \subseteq \mathcal{B}(2r)$.

Now consider the axis aligned grid over $\mathbb{R}^d$ with cell side length $\delta = \varepsilon r/2\sqrt{d}$. Any ball of radius $2r$ intersects at most $O((2r/\delta)^d) = O(1/\varepsilon^d)$ cells, for constant d. Thus $\mathcal{B}(2r)$ intersects $O(k/\varepsilon^d)$ grid cells, and as $C^* \subseteq \mathcal{B}(2r)$, there are thus $O(k/\varepsilon^d)$ possible cells for each center

[3] This is our only result that holds for arbitrary metric spaces, which is why $opt_{\alpha,\beta}$ was defined for points specifically in $\mathbb{R}^d$.

in C^*. Now for any point $q = (q_1, \ldots, q_d) \in \mathbb{R}^d$, let $grid_\delta(q) = (\delta\lfloor q_1/\delta \rfloor, \ldots, \delta\lfloor q_d/\delta \rfloor)$ be the lowest corner of the grid cell containing q (specifically the coordinate-wise minimal corner). Let $grid_\delta(C^*) = \{grid_\delta(c_1^*), \ldots, grid_\delta(c_k^*)\}$. As the diameter of each cell is $\varepsilon r/2$, by Observation 3 we have that $cost_{\infty,\infty}(grid_\delta(C^*), \mathcal{P}) \leq cost_{\infty,\infty}(C^*, \mathcal{P}) + \varepsilon r/2 = r^* + \varepsilon r/2 \leq r^* + \varepsilon r^* = (1+\varepsilon)r^*$.

The above motivates the following algorithm. First compute $C = \{c_1, c_2, \ldots, c_k\}$ using **greedy**$(k, \mathcal{P})$. This determines a set of $O(k/\varepsilon^d)$ grid cells around the centers of C which must contain C^*. We try all possible $O((k/\varepsilon^d)^k)$ possibilities for these k cells and for each possibility we use the lowest corners of the respective cells as our candidate set of centers. We then return as our solution the set of such centers C' which had the minimum $cost_{\infty,\infty}(C', \mathcal{P})$ value. As $grid_\delta(C^*)$ will be one of the possible sets of centers considered, where we already argued that $cost_{\infty,\infty}(grid_\delta(C^*), \mathcal{P}) \leq (1+\varepsilon)r^*$, this procedure is guaranteed to output a $(1+\varepsilon)$-approximation.

As for the running time, the initial call to **greedy**$(k, \mathcal{P})$ takes $O(nk)$ time. There are $O((k/\varepsilon^d)^k)$ subsets of k cells we consider, and for each one we first compute the set C' of lower corners in $O(k)$ time. Then we compute $cost_{\infty,\infty}(C', \mathcal{P})$, which can be done in $O(nk)$ time. Thus the overall time is $O(nk(k/\varepsilon^d)^k) = O(nk^{k+1}/\varepsilon^{dk})$ ◀

▶ Remark 5. For k-center clustering it is common to consider the discrete version of the problem where the centers must come from the input point set. One can easily argue that both Theorem 2 and Theorem 4 extend to this variant. (In the proof of Theorem 4, we would restrict to non-empty grid cells for potential center locations.)

4 k-means Sets Clustering

Given $\mathcal{P} = \{P_1, \ldots P_m\}$ and a parameter k, for any set $C \subset \mathbb{R}^d$ of k centers recall that

$$cost_{1,2}(C, \mathcal{P}) = \sum_{i=1}^{m} f_2(C, P_i) = \sum_{i=1}^{m} \min_{c \in C} f_2(c, P_i) = \sum_{i=1}^{m} \min_{c \in C} \sum_{p \in P_i} \|c - p\|^2.$$

Note that this is the natural analogue of the k-means problem to clustering point sets. Specifically, when each set is a single point, that is $P_i = \{p_i\}$ for all i, then this is equivalent to the standard k-means clustering cost on the point set $P = \{p_1, \ldots, p_m\}$, as then $cost_{1,2}(C, \mathcal{P}) = \sum_{i=1}^{m} \|p_i - C\|^2$

▶ **Definition 6.** *For a given point set $P \subset \mathbb{R}^d$, the* centroid *of P is defined as $\bar{p} = \bar{p}(P) = \frac{\sum_{p \in P} p}{|P|}$.*

Given a collection of sets $\mathcal{P} = \{P_1, \ldots, P_m\}$ in $\mathbb{R}^d$, let $\bar{P}_i = \{\bar{p}_{i1}, \ldots, \bar{p}_{i|P_i|}\}$ be the set of $|P_i|$ distinct copies of $\bar{p}_i = \bar{p}(P_i)$, and let $\bar{P}(\mathcal{P}) = \cup_i \bar{P}_i$. Note, while $\bar{p}_{ij}$ and $\bar{p}_{ik}$ are collocated they are viewed as distinct points, and thus $|\bar{P}_i| = |P_i|$ and $|\bar{P}(\mathcal{P})| = \sum_i |P_i|$.

It is well known that the optimal solution to the 1-mean problem with respect to a point set P is the centroid $C = \{\bar{p}(P)\}$. We require the following standard lemma, whose proof can be found in [16].

▶ **Lemma 7.** *For a point set $P \subset \mathbb{R}^d$ with centroid $\bar{p} = \bar{p}(P)$ and any point $x \in \mathbb{R}^d$, we have, $f_2(x, P) = f_2(\bar{p}, P) + |P| \cdot f_2(x, \bar{p})$.*

Given a collection of point sets, we now argue that one can solve the sets clustering problem by solving the k-means problem on the centroids of the sets.

▶ **Theorem 8.** *For a parameter k and a point set $P \subset \mathbb{R}^d$, let* **kmeansAlg**(k, P) *denote any algorithm which achieves an α-approximation for the k-means problem.*

Given a parameter k and collection of sets $\mathcal{P} = \{P_1, \ldots, P_m\}$ in $\mathbb{R}^d$, **kmeansAlg**$(k, \bar{P}(\mathcal{P}))$ *is an α-approximation for computing $opt_{1,2}(k, \mathcal{P})$.*

Proof. For any two point sets $C, P \subset \mathbb{R}^d$, by lemma 7 we have:

$$f_2(C, P) = \min_{c \in C} f_2(c, P) = \min_{c \in C}(f_2(\bar{p}, P) + |P| \cdot f_2(c, \bar{p})) = f_2(\bar{p}, P) + |P| \cdot \min_{c \in C} f_2(c, \bar{p}).$$

Recall from Section 2 that $kmeans(C, P) = \sum_{p \in P} \|p - C\|^2 = \sum_{p \in P} \min_{c \in C} f_2(c, p)$. Thus summing the above equation for P_i over all i gives:

$$\begin{aligned} cost_{1,2}(C, \mathcal{P}) &= \sum_{i=1}^{m} f_2(C, P_i) = \sum_{i=1}^{m} f_2(\bar{p}_i, P_i) + \sum_{i=1}^{m} |P_i| \cdot \min_{c \in C} f_2(c, \bar{p}_i) \\ &= \sum_{i=1}^{m} f_2(\bar{p}_i, P_i) + \sum_{p \in \bar{P}(\mathcal{P})} \min_{c \in C} f_2(c, p) = \sum_{i=1}^{m} f_2(\bar{p}_i, P_i) + kmeans(C, \bar{P}(\mathcal{P})). \end{aligned}$$

Thus $cost_{1,2}(C, \mathcal{P}) = \sum_{i=1}^{m} f_2(\bar{p}_i, P_i) + kmeans(C, \bar{P}(\mathcal{P}))$. So let C^* denote the optimal k-means solution on $\bar{P}(\mathcal{P})$, that is $C^* = \arg\min_{C \subset \mathbb{R}^d, |C|=k} kmeans(C, \bar{P}(\mathcal{P}))$. As $\sum_{i=1}^{m} f_2(\bar{p}_i, P_i)$ does not depend on C, the above equation implies that minimizing $kmeans(C, \bar{P}(\mathcal{P}))$ also minimizes $cost_{1,2}(C, \mathcal{P})$, i.e. $C^* = \arg\min_{C \subset \mathbb{R}^d, |C|=k} cost_{1,2}(C, \mathcal{P})$.

So let C' be the $\alpha \geq 1$ approximation returned by **kmeansAlg**$(k, \bar{P}(\mathcal{P}))$, meaning $kmeans(C', \bar{P}(\mathcal{P})) \leq \alpha \cdot kmeans(C^*, \bar{P}(\mathcal{P}))$. Then we have

$$\begin{aligned} cost_{1,2}(C', \mathcal{P}) &= \sum_{i=1}^{m} f_2(\bar{p}_i, P_i) + kmeans(C', \bar{P}(\mathcal{P})) \\ &\leq \sum_{i=1}^{m} f_2(\bar{p}_i, P_i) + \alpha \cdot kmeans(C^*, \bar{P}(\mathcal{P})) \\ &\leq \alpha(\sum_{i=1}^{m} f_2(\bar{p}_i, P_i) + kmeans(C^*, \bar{P}(\mathcal{P}))) = \alpha \cdot cost_{1,2}(C^*, \mathcal{P}) \end{aligned}$$ ◀

As discussed in the introduction, there are many known $(1+\varepsilon)$-approximation algorithms for k-means, with various trade-offs in the running times depending on the parameters n, d, k, and ε. Moreover, as k-means is a special case of our problem, we should not expect our times to be faster than what is possible for k-means.

Before we can apply such algorithms the above theorem requires we compute $\bar{P}(\mathcal{P})$. However, this takes only $O(dn)$ time, where $n = \sum_i |P_i|$, as the centroid of a set is simply the average of the points. Thus the above theorem immediately implies that we have the following.

▶ **Corollary 9.** *For a parameter k and set $P \subset \mathbb{R}^d$ of n points, let $T(n, k, d, \varepsilon)$ denote the running time of any algorithm which achieves a $(1+\varepsilon)$-approximation for k-means.*

Given a parameter k and collection of sets $\mathcal{P} = \{P_1, \ldots, P_m\}$ in $\mathbb{R}^d$, where $n = \sum_i |P_i|$, there is a $(1+\varepsilon)$-approximation for computing $opt_{1,2}(k, \mathcal{P})$ with $O(dn + T(n, k, d, \varepsilon))$ running time.

5 k-median Sets Clustering

Given $\mathcal{P} = \{P_1, \dots P_m\}$ and a parameter k, for any set $C \subset \mathbb{R}^d$ of k centers recall that

$$cost_{1,\infty}(C, \mathcal{P}) = \sum_{i=1}^{m} f_\infty(C, P_i) = \sum_{i=1}^{m} \min_{c \in C} f_\infty(c, P_i) = \sum_{i=1}^{m} \min_{c \in C} \max_{p \in P_i} \|c - p\|.$$

As discussed in the introduction, the above cost (as well as $cost_{1,1}(C, \mathcal{P})$ studied in [20]) generalizes the k-median problem to clustering point sets. Specifically, when each set consists of a single point, i.e., $P_i = \{p_i\}$ for all i, the problem reduces to the standard k-median clustering cost on the point set $P = \{p_1, \dots, p_m\}$, as then $cost_{1,\infty}(C, \mathcal{P}) = \sum_{i=1}^{m} \|p_i - C\|$.

▶ **Lemma 10.** *Let $\mathcal{P} = \{P_1, \dots, P_m\}$ be a collection of sets and $B = \{b_1, \dots, b_m\}$ where b_i is the center of the minimum enclosing ball of P_i, then $\sum_{i=1}^{m} \max_{p \in P_i} \|p - b_i\| \leq optcost_{1,\infty}(k, \mathcal{P})$.*

Proof. Since b_i is the center of the minimum enclosing ball of P_i, by definition we have that $\max_{p \in P_i} \|p - b_i\| = \min_{b \in \mathbb{R}^d} \max_{p \in P_i} \|p - b\| = \min_{b \in \mathbb{R}^d} f_\infty(b, P_i)$. So if $C^* = \{c_1^*, c_2^*, \dots, c_k^*\} = opt_{1,\infty}(k, \mathcal{P})$, then,

$$\sum_{i=1}^{m} \max_{p \in P_i} \|p - b_i\| = \sum_{i=1}^{m} \min_{b \in \mathbb{R}^d} f_\infty(b, P_i) \leq \sum_{i=1}^{m} \min_{c \in C^*} f_\infty(c, P_i) = \sum_{i=1}^{m} f_\infty(C^*, P_i) = optcost_{1,\infty}(k, \mathcal{P})$$

◀

▶ **Lemma 11.** *Let $\mathcal{P} = \{P_1, \dots, P_m\}$ be a collection of point sets from $\mathbb{R}^d$, let $B = \{b_1, \dots, b_m\}$ where b_i is the center of the minimum enclosing ball of P_i, and let $C \subset \mathbb{R}^d$ be any set of k centers, then $kmedian(C, B) \leq cost_{1,\infty}(C, \mathcal{P})$.*

Proof. First, we make the standard observation that for any point $q \in \mathbb{R}^d$, $\|q - b_i\| \leq \max_{p \in P_i} \|p - q\|$. (Assume $q \neq b_i$ as otherwise the inequality trivially holds.) This holds by considering the hyperplane passing through b_i whose normal is in the direction from b_i to q. As b_i is the center of the minimum enclosing ball of P_i, the closed halfspace defined by this hyperplane and not containing q must contain a point from P_i, and thus this point is at least as far as b_i from q. Thus we have,

$$\begin{aligned} kmedian(C, B) &= \sum_{i=1}^{m} \|b_i - C\| = \sum_{i=1}^{m} \min_{c \in C} \|b_i - c\| \leq \sum_{i=1}^{m} \min_{c \in C} \max_{p \in P_i} \|p - c\| \\ &= \sum_{i=1}^{m} f_\infty(C, P_i) = cost_{1,\infty}(C, \mathcal{P}) \end{aligned}$$

◀

▶ **Theorem 12.** *For a parameter k and a point set $P \subset \mathbb{R}^d$, let* **kmedianAlg**(k, P) *denote any algorithm which achieves an α-approximation for the k-median problem.*

Given a parameter k and collection of sets $\mathcal{P} = \{P_1, \dots, P_m\}$ in $\mathbb{R}^d$, let $B = \{b_1, \dots, b_m\}$ where b_i is the center of the minimum enclosing ball of P_i. Then **kmedianAlg**(k, B) *is a $(1 + \alpha)$-approximation for computing $opt_{1,\infty}(k, \mathcal{P})$.*

Proof. Let $C = \{c_1, \dots, c_k\} \subset \mathbb{R}^d$ be the set of centers returned by **kmedianAlg**(k, B), for which we have $kmedian(C, B) \leq \alpha \cdot \min_{C' \subseteq \mathbb{R}^d, |C'| = k} kmedian(C', B)$. Also let $C^* = opt_{1,\infty}(k, \mathcal{P})$. Then using triangle inequality,

$$
\begin{aligned}
cost_{1,\infty}(C,\mathcal{P}) &= \sum_{i=1}^{m} \min_{c\in C}\max_{p\in P_i}\|c-p\| \leq \sum_{i=1}^{m} \min_{c\in C}\max_{p\in P_i}(\|p-b_i\|+\|c-b_i\|) \\
&= \sum_{i=1}^{m}((\max_{p\in P_i}\|p-b_i\|)+(\min_{c\in C}\|c-b_i\|)) \\
&= \sum_{i=1}^{m}\max_{p\in P_i}\|p-b_i\| + \sum_{i=1}^{m}\min_{c\in C}\|c-b_i\| \\
&= \sum_{i=1}^{m}\max_{p\in P_i}\|p-b_i\| + kmedian(C,B) \\
&\leq optcost_{1,\infty}(k,\mathcal{P}) + kmedian(C,B) && \text{Lemma 10} \\
&\leq optcost_{1,\infty}(k,\mathcal{P}) + \alpha\cdot\min_{C'\subseteq\mathbb{R}^d,|C'|=k} kmedian(C',B) \\
&\leq optcost_{1,\infty}(k,\mathcal{P}) + \alpha\cdot kmedian(C^*,B) \\
&\leq optcost_{1,\infty}(k,\mathcal{P}) + \alpha\cdot cost_{1,\infty}(C^*,\mathcal{P}) && \text{Lemma 11} \\
&= optcost_{1,\infty}(k,\mathcal{P}) + \alpha\cdot optcost_{1,\infty}(k,\mathcal{P}) \\
&= (1+\alpha)optcost_{1,\infty}(k,\mathcal{P}) && \blacktriangleleft
\end{aligned}
$$

As discussed in the introduction, there are many known $(1+\varepsilon)$-approximation algorithms for k-median, with various trade-offs in the running times depending on the parameters n, d, k, and ε. Moreover, as k-median is a special case of our problem, we should not expect our times to be faster than what is possible for k-median.

Before we can apply such algorithms, the above theorem requires computing the minimum enclosing ball of each set P_i, for which we can use $(1+\varepsilon)$-approximate minimum enclosing ball center, which can be computed in $O(dn/\varepsilon+1/\varepsilon^5)$ time [2]. Doing so would introduce a $(1+\varepsilon)$ factor in Lemma 10, and thus using this together with a $(1+\varepsilon)$-approximation for k-median, the same analysis from the proof of Theorem 12, would yield a $(1+\varepsilon)^2$ approximation to computing $opt_{1,\infty}(k,\mathcal{P})$. By observing that $(1+\varepsilon/4)^2 \leq (1+\varepsilon)$ for any $0<\varepsilon\leq 1$ we immediately have the following.

▶ **Corollary 13.** *For a parameter k and set $P\subset\mathbb{R}^d$ of n points, let $T(n,k,d,\varepsilon)$ denote the running time of any algorithm which achieves a $(1+\varepsilon)$-approximation for k-median. Moreover, let $MEB(n,d,\varepsilon)$ denote the time to compute a $(1+\varepsilon)$-approximate minimum enclosing ball of P.*

Given a parameter k and collection of sets $\mathcal{P}=\{P_1,\ldots,P_m\}$ in $\mathbb{R}^d$, where $n=\sum_i|P_i|$, then for any $0<\varepsilon\leq 1$, there is a $(2+\varepsilon)$-approximation for computing $opt_{1,\infty}(k,\mathcal{P})$ with $O(MEB(n,d,\varepsilon/4)+T(n,k,d,\varepsilon/4))$ running time.

6 Another Variant of k-center Sets Clustering

Above we gave improved approximation algorithms for the $cost_{\infty,\infty}(C,\mathcal{P})$ objective, previously considered by [20], and which naturally generalizes the k-center problem to sets. Here we consider an alternative generalization of k-center to sets clustering. Namely, given $\mathcal{P}=\{P_1,...,P_m\}$ and a parameter k, find the set $C\subset\mathbb{R}^d$ of k centers that minimizes:

$$cost_{\infty,1}(\mathcal{P},C) = \max_{P\in\mathcal{P}} f_1(C,P) = \max_{P\in\mathcal{P}}\min_{c\in C} f_1(c,P) = \max_{P\in\mathcal{P}}\min_{c\in C}\sum_{p\in P}\|c-p\|.$$

This variant poses additional challenges, so for simplicity we will assume that the sets in $\mathcal{P}$ lie in $\mathbb{R}^2$, though a similar approach should extend to any constant dimension. Moreover, in this section we state our running times simply as being polynomial in n, as the precise constant in the exponent is large, as opposed to earlier sections which may be linear in n (depending on the subroutines used).

▶ **Definition 14.** *In the* weighted k-center problem*, we are given a parameter k, a set of points $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$ and a set of non-negative weights $W = \{w_1, \ldots, w_n\}$, where w_i is the weight associated with point p_i, and the goal is to find the set $C \subset \mathbb{R}^2$ of k centers which minimizes the cost function: $wkcenter(C, P) = \max_{p_i \in P} \min_{c \in C} w_i \cdot \|p_i - c\|$.*

Observe that the weighted k-center objective can be viewed as a special instance of our $cost_{\infty,1}$ objective, as a point with weight $|P_i|$ models the case when all points in P_i are collocated, for each $P_i \in \mathcal{P}$. Thus intuitively, to approximate $cost_{\infty,1}$ will require approximating $wkcenter$.

[19] considered the weighted k-center problem where the input is instead an edge-weighted graph $G = (V, E)$, where each vertex v has an associated non-negative weight w_v. Specifically, let $d(u, v)$ denote the shortest path distance between u and v (with respect to edge weights, not vertex weights), then they seek the set $C \subseteq V$ of k centers which minimizes: $wkcenter(C, V) = \max_{v \in V} \min_{c \in C} w_v \cdot d(v, c)$. For this problem [19] give a 2-approximation, and we now describe how this implies a 2-approximation for the case in plane.

▶ **Lemma 15.** *There exists a polynomial time 2-approximation algorithm for the weighted k-center problem in the plane. That is, given a parameter k, a set of points $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$, and a set of non-negative weights $W = \{w_1, \ldots, w_n\}$ where w_i is the weight associated with point p_i, there is a polynomial time 2-approximation to $\min_{C \subset \mathbb{R}^2, |C|=k} wkcenter(C, P)$.*

Proof. As we allow centers to be located anywhere in the plane, we first describe how to find a polynomial sized set Γ which contains the optimal set of centers as a subset.

For any two points $p_i, p_j \in P$, define their weighted bisector, $\beta(p_i, p_j)$, as the subset of points $q \in \mathbb{R}^2$ such that $w_i\|p_i - q\| = w_j\|p_j - q\|$. It is well known that $\beta(p_i, p_j)$ is a line when $w_i = w_j$, and otherwise is a circle (called the Apollonius circle) containing the heavier weight point. Consider the arrangement of all such weighted bisectors over all pairs of points in P. The vertices of the arrangement occur at intersections of bisectors, and the edges are circular arcs between vertices (or entire bisectors, when the bisector does not intersect any other bisector). Let $\Gamma \subset \mathbb{R}^2$ be the set containing all points in P, all vertices in the arrangement, and for every edge on a bisector $\beta(p_i, p_j)$, a point on that edge achieving the minimum value of $w_i\|p_i - q\| = w_j\|p_j - q\|$ (which may occur at a vertex of the arrangement). Observe that $|\Gamma| = O(n^4)$ as this is the complexity of the arrangement.

So let C^* be an optimal set of centers, and consider some $c^* \in C^*$. We now argue that if $c^* \notin \Gamma$, then it can be moved to a point in Γ in such a way that the weighted k-center objective does not increase. First, let P' be the subset of points from P which call c^* their nearest center in C^* and let $q = \arg\max_{p_i \in P'} w_i\|p_i - c^*\|$. Consider moving c^* continuously from its initial location towards q (i.e. along the segment c^*q). As we move along this segment the weighted distance to q strictly decreases, and moreover until we cross a bisector $\beta(q, p_i)$ for some p_i, q remains the point in P' furthest from c^* (i.e. the point determining c^*'s contribution to the overall objective). First suppose that c^* is moved all the way to q without crossing a bisector involving q. In this case q remained the furthest point from c^*, and so the cost of assigning P' to c^* did not increase (implying in fact that $c^* = q$ initially). In the second case, we run into some bisector $\beta(q, p_i)$ for some p_i, at which point we stop

moving c^* towards q (to ensure q remains the furthest). Now if c^* is at a vertex of the arrangement of bisectors, then we are done as we included all vertices in Γ. Otherwise, c^* is on the interior of an edge in the arrangement. Now the point achieving the minimum value of the weighted distance from q (or equivalently p_i) to c^* along this edge is in included in Γ (and may in fact be a vertex), and so we can move c^* to this point without increasing the cost of assigning P' to c^*, as we do not cross any other bisector while moving along this edge.

Now we construct a weighted graph instance $G = (V, E)$ of weighted k-center as follows. First, set $V = \Gamma$ (recall $P \subseteq \Gamma$). Now for every pair of points in V set its edge weight equal to the Euclidean distance between the corresponding set of points. Finally, for the vertex weights, set the weight of all vertices in $\Gamma \setminus P$ equal to 0 (i.e. they do not have to be clustered), and for each $p_i \in P$ its weight remains w_i (i.e. the weight from the weighted k-center instance in the plane). Note that by construction, the cost of any solution $C \subseteq V$ to this graph based weighted k-center problem is equivalent to the cost of C for the corresponding weighted k-center problem in the plane. Thus, as $V = \Gamma$ is guaranteed to contain an optimal solution to our instance in the plane, if we simply call the polynomial time 2-approximation algorithm from [19] on this graph instance, then the returned solution is also a 2-approximation for our instance in the plane. ◀

Our goal now is to use the above lemma to get a constant factor approximation to $opt_{\infty,1}(k, \mathcal{P})$. First we need the following lemma, which is the analogue of Lemma 10 from the prior section.

▶ **Lemma 16.** *Let $\mathcal{P} = \{P_1, ..., P_m\}$ be a collection of sets. Then, for any point-set $P_i \in \mathcal{P}$,*

$$\min_{\mu \in \mathbb{R}^d} \sum_{p \in P_i} \|\mu - p\| \leq optcost_{\infty,1}(k, \mathcal{P}).$$

Proof. Let $C^* = opt_{\infty,1}(k, \mathcal{P})$. Then for any point set $P_i \in \mathcal{P}$,

$$\min_{\mu \in \mathbb{R}^d} \sum_{p \in P_i} \|\mu - p\| \leq \min_{c^* \in C^*} \sum_{p \in P_i} \|c^* - p\| \leq \max_{P_i \in \mathcal{P}} \min_{c^* \in C^*} \sum_{p \in P_i} \|c^* - p\| = optcost_{\infty,1}(k, \mathcal{P}).$$ ◀

▶ **Theorem 17.** *For a parameter k, a set of points $P = \{p_1, \ldots, p_l\} \subset \mathbb{R}^2$, and weights $W = \{w_1, \ldots, w_l\}$ where $w_i \geq 0$ is the weight associated with point p_i, let* **weightedKCenter**(k, P, W) *denote any algorithm which achieves an α_1-approximation for the weighted k-center problem. Also, for a point set $P \subset \mathbb{R}^2$, let* **1medianAlg**(P) *denote any algorithm that achieves an α_2-approximation for the 1-median problem.*

Given a parameter k and collection of sets $\mathcal{P} = \{P_1, \ldots, P_m\}$ in $\mathbb{R}^2$, there is a polynomial time $(\alpha_1 + \alpha_2 + \alpha_1 \cdot \alpha_2)$-approximation for computing $opt_{\infty,1}(k, \mathcal{P})$.

Proof. For any set $C \subset \mathbb{R}^2$ of k centers, and any set of points $X = \{x_1, \ldots, x_m\}$, by the triangle inequality we have

$$\begin{aligned} cost_{\infty,1}(C, \mathcal{P}) &= \max_{P_i \in \mathcal{P}} \min_{c \in C} \sum_{p \in P_i} \|c - p\| \leq \max_{P_i \in \mathcal{P}} \min_{c \in C} \Big(\sum_{p \in P_i} \|x_i - p\| + \sum_{p \in P_i} \|c - x_i\| \Big) \\ &\leq \max_{P_i \in \mathcal{P}} \sum_{p \in P_i} \|x_i - p\| + \max_{P_i \in \mathcal{P}} \min_{c \in C} |P_i| \cdot \|c - x_i\| \end{aligned}$$

Now, let the optimal set of centers for our problem be $C^* = opt_{\infty,1}(k, \mathcal{P})$ and let C' be the α_1-approximation returned by **weightedKCenter**(k, X, W) where $X = \{x_1, \ldots, x_m\}$ is the set of α_2-approximate 1-medians from the P_i sets, i.e. $x_i =$ **1medianAlg**(P_i), and $W = \{|P_1|, \ldots, |P_m|\}$. Then,

$$
\begin{aligned}
&cost_{\infty,1}(C', \mathcal{P}) \\
&\quad \leq \max_{P_i \in \mathcal{P}} \sum_{p \in P_i} \|x_i - p\| + \max_{P_i \in \mathcal{P}} \min_{c \in C'} |P_i| \cdot \|c - x_i\| \\
&\quad \leq \max_{P_i \in \mathcal{P}} (\alpha_2 \cdot \min_{\mu \in \mathbb{R}^2} \sum_{p \in P_i} \|\mu - p\|) + \alpha_1 \cdot \min_{\bar{C} \subset \mathbb{R}^2, |\bar{C}| = k} (\max_{P_i \in \mathcal{P}} \min_{c \in \bar{C}} |P_i| \cdot \|c - x_i\|) \\
&\quad \leq \alpha_2 \cdot \max_{P_i \in \mathcal{P}} \min_{\mu \in \mathbb{R}^2} \sum_{p \in P_i} \|\mu - p\| + \alpha_1 \cdot \max_{P_i \in \mathcal{P}} \min_{c \in C^*} |P_i| \cdot \|c - x_i\| \\
&\quad \leq \alpha_2 \cdot costopt_{\infty,1}(k, \mathcal{P}) + \alpha_1 \cdot \max_{P_i \in \mathcal{P}} \min_{c \in C^*} \sum_{p \in P_i} (\|x_i - p\| + \|c - p\|) \qquad \text{Lemmma 16} \\
&\quad \leq \alpha_2 \cdot costopt_{\infty,1}(k, \mathcal{P}) + \alpha_1 \cdot (\max_{P_i \in \mathcal{P}} \sum_{p \in P_i} \|x_i - p\| + \max_{P_i \in \mathcal{P}} \min_{c \in C^*} \sum_{p \in P_i} \|c - p\|) \\
&\quad \leq \alpha_2 \cdot costopt_{\infty,1}(k, \mathcal{P}) + \alpha_1 \cdot (\max_{P_i \in \mathcal{P}} (\alpha_2 \cdot \min_{\mu \in \mathbb{R}^2} \sum_{p \in P_i} \|\mu - p\|) + costopt_{\infty,1}(k, \mathcal{P})) \\
&\quad \leq \alpha_2 \cdot costopt_{\infty,1}(k, \mathcal{P}) + \alpha_1 \cdot (\alpha_2 \cdot costopt_{\infty,1}(k, \mathcal{P}) + costopt_{\infty,1}(k, \mathcal{P})) \\
&\quad = (\alpha_1 + \alpha_2 + \alpha_1 \cdot \alpha_2) \cdot costopt_{\infty,1}(k, \mathcal{P}) \qquad \blacktriangleleft
\end{aligned}
$$

▶ **Corollary 18.** *Given a parameter k and collection of sets $\mathcal{P} = \{P_1, \ldots, P_m\}$ in $\mathbb{R}^2$, there is a polynomial time $(5+\varepsilon)$-approximation for computing $opt_{\infty,1}(k, \mathcal{P})$, for any constant $\varepsilon > 0$.*

Proof. Using [11], we can get a polynomial time $(1+\varepsilon/3)$-approximation for the 1-median, for any constant $\varepsilon > 0$. Lemma 15 gives us a polynomial time 2-approximation for weighted k-center. So we have $\alpha_1 = 2$, $\alpha_2 = (1+\varepsilon/3)$, and $(\alpha_1+\alpha_2+\alpha_1\alpha_2) = (2+(1+\varepsilon/3)+2(1+\varepsilon/3)) = (5+\varepsilon)$. Thus, applying Theorem 17 with these algorithms as subroutines gives a polynomial time $(5+\varepsilon)$-approximation for $opt_{\infty,1}(k, \mathcal{P})$. ◀

6.1 Approximation Improvement

We now show how to improve the above $(5+\varepsilon)$-approximation into a $(3+\varepsilon)$-approximation, by constructing and searching with a $(3+\varepsilon)$-approximate decision procedure.

Algorithm 2 **decider**$(\mathcal{P} = \{P_1, \ldots P_m\}, k, r)$.

```
Initialize all point sets P_1, P_2, ..., P_m as unmarked.
Compute M = {μ_1, ..., μ_m} where μ_i = 1medianAlg(P_i) is an α-approximate median
Initialize C = {}.
repeat
    Let P_i = arg max_{unmarked P∈P} |P|.
    Mark point-set P_j if Σ_{p∈P_j} ||p − μ_i|| ≤ r.
    Set C = C ∪ {μ_i}.
until all sets are marked;
if |C| > k then
    return "optcost_{∞,1}(k, P) > r/(2 + α)"
return C
```

▶ **Lemma 19.** *For a point set $P \subset \mathbb{R}^2$, let* **1medianAlg**(P) *denote any algorithm that achieves an α-approximation for the 1-median problem.*

Given a collection of sets $\mathcal{P} = \{P_1, ..., P_m\}$ in $\mathbb{R}^2$, a parameter k, and a radius $r \in \mathbb{R}$, then **decider**$(\mathcal{P}, k, r)$ *either returns a set $C \subset \mathbb{R}^2$ of k centers such that $cost_{\infty,1}(C, \mathcal{P}) \leq r$, or correctly returns that $optcost_{\infty,1}(k, \mathcal{P}) > r/(2+\alpha)$.*

Proof. **decider**$(\mathcal{P}, k, r)$ starts by initializing all point sets in $\mathcal{P}$ as unmarked, and computing the approximate medians of each point set using **1medianAlg**. Next, the algorithm picks the approximate median of the unmarked point set which has the most points, adds it to the set C of centers, and mark all the sets that are $\leq r$ from this approximate median. This process is repeated until all sets are marked. At the end of the loop, if $|C| \leq k$, then the algorithm returns C, as clearly we have obtained a valid clustering with cost $\leq r$.

Otherwise, $|C| \geq k+1$, in which case the algorithm returns "$optcost_{\infty,1}(k, \mathcal{P}) > r/(2+\alpha)$", which we now argue is correct. Let $\{c_1, ..., c_{k+1}\}$ be the first $k+1$ centers selected by algorithm, where $c_i \in M$ was selected in the ith iteration, and let $\mathcal{S} = \{S_1, S_2, ..., S_{k+1}\}$, where $S_i \in \mathcal{P}$ is the set such that $c_i =$ **1medianAlg**(S_i). Let $C^* = opt_{\infty,1}(k, \mathcal{P})$. Then by the pigeonhole principle, there must be some $c^* \in C^*$ and sets $S_i, S_j \in \mathcal{S}$ with $i < j$ such that $c^* = \arg\min_{c \in C^*} \sum_{s \in S_i} \|c - s\| = \arg\min_{c \in C^*} \sum_{s \in S_j} \|c - s\|$. Note that as $i < j$, by line 5 of the algorithm we know $|S_i| \geq |S_j|$. Then,

$$\begin{aligned}
\sum_{s \in S_j} \|c_i - s\| &\leq \sum_{s \in S_j} (\|c_i - c^*\| + \|c^* - s\|) = \sum_{s \in S_j} \|c_i - c^*\| + \sum_{s \in S_j} \|c^* - s\| \\
&\leq |S_j| \cdot \|c_i - c^*\| + optcost_{\infty,1}(k, \mathcal{P}) \leq |S_i| \cdot \|c_i - c^*\| + optcost_{\infty,1}(k, \mathcal{P}) \\
&\leq \sum_{s \in S_i} (\|c_i - s\| + \|c^* - s\|) + optcost_{\infty,1}(k, \mathcal{P}) \\
&= \sum_{s \in S_i} \|c_i - s\| + \sum_{s \in S_i} \|c^* - s\| + optcost_{\infty,1}(k, \mathcal{P}) \\
&\leq \alpha \cdot \min_{c' \in \mathbb{R}^2} \sum_{s \in S_i} \|c' - s\| + optcost_{\infty,1}(k, \mathcal{P}) + optcost_{\infty,1}(k, \mathcal{P}) \\
&\leq \alpha \cdot optcost_{\infty,1}(k, \mathcal{P}) + 2 \cdot optcost_{\infty,1}(k, \mathcal{P}) \\
&= (2+\alpha) \cdot optcost_{\infty,1}(k, \mathcal{P})
\end{aligned}$$

Since $i < j$, we know that c_i did not cover S_j within radius r, that is $r < \sum_{s \in S_j} \|c_i - s\|$. Thus by the above, $r < \sum_{s \in S_j} \|c_i - s\| \leq (2+\alpha) \cdot optcost_{\infty,1}(k, \mathcal{P})$. Therefore, the algorithm correctly returns that $optcost_{\infty,1}(k, \mathcal{P}) > \frac{r}{2+\alpha}$. ◀

▶ **Theorem 20.** *Given a parameter k and a collection of point sets $\mathcal{P} = \{P_1, \ldots, P_m\}$ in $\mathbb{R}^2$, for any constant $0 < \varepsilon \leq 1$, there exists a polynomial time $(3+\varepsilon)$-approximation algorithm for computing $opt_{\infty,1}(k, \mathcal{P})$.*

Proof. Let C' be the $(5+\varepsilon)$-approximate set of centers returned by Corollary 18. Let $r = cost_{\infty,1}(C', \mathcal{P})$ and let $r^* = opt_{\infty,1}(k, \mathcal{P})$, where by Corollary 18 we know that $r^* \in [\frac{r}{5+\varepsilon}, r]$. Consider the candidate radii set $R = \{r_0, r_1, \ldots, r_z\}$, where $r_i = \frac{r}{5+\varepsilon}(1+\varepsilon/4)^i$ and $z = \lceil \log_{1+\varepsilon/4}(5+\varepsilon) \rceil = O(1)$ for any constant $\varepsilon > 0$. Note that for any i we have $r_{i+1}/r_i = (1+\varepsilon/4)$, and z was chosen such that $r_z \geq r$.

Now we iterate through the candidate radii $r_i \in R$, in order from $i = 0$ to z. In the ith iteration we call Algorithm 2 with radius $(2+\alpha) \cdot r_i$ where α is the approximation ratio of the subroutine used to compute the approximate 1-medians. We can use [11] to get an $\alpha = (1+\varepsilon/8)$-approximate 1-median, which runs in polynomial time for any

constant $\varepsilon > 0$. Let j be the first iteration where Algorithm 2 returns a set C rather than "$optcost_{\infty,1}(k,\mathcal{P}) > (2+\alpha)\cdot r_j/(2+\alpha)$". We claim that this set C is the desired approximation.

First, observe that this procedure is guaranteed to return some set C, that is j is well defined. Specifically, when $j = z$, the call to Algorithm 2 with radius $(2+\alpha)\cdot r_z$, must return a set C since otherwise by Lemma 19 the optimal radius $r^* > \frac{(2+\alpha)\cdot r_z}{2+\alpha} = r_z \geq r$ which is a contradiction to the fact that $r^* \in [\frac{r}{5+\varepsilon}, r]$.

Lemma 19 guarantees that $cost_{\infty,1}(C,\mathcal{P}) \leq (2+\alpha)\cdot r_j$. If $j = 0$, then we are done as we know $(2+\alpha)\cdot r_j = (3+\varepsilon/8)\cdot r_0 = (3+\varepsilon/8)\cdot \frac{r}{5+\varepsilon} \leq (3+\varepsilon/8)\cdot r^* \leq (3+\varepsilon)\cdot r^*$. Otherwise, $j > 0$ in which case we know r_{j-1} returned "$optcost_{\infty,1}(k,\mathcal{P}) > (2+\alpha)\cdot r_{j-1}/(2+\alpha)$", and so again by Lemma 19 we then know that $r^* > \frac{(2+\alpha)\cdot r_{j-1}}{2+\alpha} = r_{j-1}$. Summarizing, $r_{j-1} < r^* \leq cost_{\infty,1}(C,\mathcal{P}) \leq (2+\alpha)\cdot r_j$, and so the approximation quality of the solution C is bounded by the ratio:

$$\frac{(2+\alpha)\cdot r_j}{r_{j-1}} = (2+\alpha)(1+\varepsilon/4) = (3+\varepsilon/8)(1+\varepsilon/4) = 3+(7/8)\varepsilon+\varepsilon^2/32 \leq 3+\varepsilon$$

Overall this is a polynomial time algorithm, as we made $O(1)$ calls to Algorithm 2, which itself is polynomial time when using a polynomial time subroutine to compute the approximate 1-medians. ◀

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. Mihai Badoiu and Kenneth L. Clarkson. Smaller core-sets for balls. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 801–802. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644240.
3. Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. *J. Comput. Syst. Sci.*, 68(2):417–441, 2004. doi:10.1016/J.JCSS.2003.07.014.
4. Ke Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM J. Comput.*, 39(3):923–947, 2009. doi:10.1137/070699007.
5. Vincent Cohen-Addad, Karthik C. S., and Euiwoong Lee. Johnson coverage hypothesis: Inapproximability of k-means and k-median in l_p-metrics. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1493–1530. SIAM, 2022. doi:10.1137/1.9781611977073.63.
6. Eduard Eiben, Fedor V. Fomin, Petr A. Golovach, William Lochet, Fahad Panolan, and Kirill Simonov. EPTAS for k-means clustering of affine subspaces. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2649–2659. SIAM, 2021. doi:10.1137/1.9781611976465.157.
7. Tomás Feder and Daniel H. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 434–444. ACM, 1988. doi:10.1145/62212.62255.
8. Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for k-means clustering based on weak coresets. In *Proceedings of the 23rd ACM Symposium on Computational Geometry (SOCG)*, pages 11–18. ACM, 2007. doi:10.1145/1247069.1247072.
9. Jie Gao, Michael Langberg, and Leonard J. Schulman. Clustering lines in high-dimensional space: Classification of incomplete data. *ACM Trans. Algorithms*, 7(1), December 2010. doi:10.1145/1868237.1868246.
10. T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.

11 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 291–300. ACM, 2004. doi:10.1145/1007352.1007400.

12 D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985. doi:10.1287/moor.10.2.180.

13 W. Hsu and G. L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979. doi:10.1016/0166-218X(79)90044-1.

14 Hongyao Huang, Georgiy Klimenko, and Benjamin Raichel. Clustering with neighborhoods. In *32nd International Symposium on Algorithms and Computation (ISAAC)*, volume 212 of *LIPIcs*, pages 6:1–6:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ISAAC.2021.6.

15 Ibrahim Jubran, Murad Tukan, Alaa Maalouf, and Dan Feldman. Sets clustering. In *Proceedings of the 37th International Conference on Machine Learning, (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 4994–5005. PMLR, 2020. URL: http://proceedings.mlr.press/v119/jubran20a.html.

16 Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. *Comput. Geom.*, 28(2-3):89–112, 2004. doi:10.1016/J.COMGEO.2004.03.003.

17 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM*, 57(2), February 2010. doi:10.1145/1667053.1667054.

18 Sagi Lotan, Ernesto Evgeniy Sanches Shayda, and Dan Feldman. Coreset for line-sets clustering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL: http://papers.nips.cc/paper_files/paper/2022/hash/f2ce95887c34393af4eb240d60017860-Abstract-Conference.html.

19 Qingzhou Wang and Kam-Hoi Cheng. A heuristic algorithm for the k-center problem with vertex weight. In *International Symposium on Algorithms (SIGAL)*, volume 450 of *Lecture Notes in Computer Science*, pages 388–396. Springer, 1990. doi:10.1007/3-540-52921-7_88.

20 Guang Xu and Jinhui Xu. Efficient approximation algorithms for clustering point-sets. *Computational Geometry*, 43(1):59–66, 2010. Special Issue on the 14th Annual Fall Workshop. doi:10.1016/j.comgeo.2007.12.002.

Sweeping a Domain with Line-Of-Sight Between Covisible Agents

Kien C. Huynh
Communications and Transport Systems, ITN, Linköping University, Sweden

Joseph S. B. Mitchell
Department of Applied Mathematics and Statistics, Stony Brook University, NY, USA

Valentin Polishchuk
Communications and Transport Systems, ITN, Linköping University, Sweden

Abstract

We consider sweeping a polygonal domain using variable-length segments whose endpoints can be considered to be mobile agents moving with bounded speeds; a point in the domain is swept when it belongs to one of the segments. The objective is to sweep the domain as quickly as possible. We show that the problem is NP-hard even in simple polygons and even for a single segment (two agents), and give constant-factor approximation algorithms, both for simple polygons and polygons with holes.

Our approximations are obtained by introducing a new type of "window partition" of the polygon, which may find other applications. For domains with holes, our results are based on a non-trivial topological argument proving a surprising fact: a connected subset of the domain, whose points are swept but not directly touched by the agents, may contain at most one hole.

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

Keywords and phrases Polygon sweeping, collaborating agents, motion coordination, makespan optimization

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.39

Supplementary Material
Software (Source Code): `https://github.com/KienHuynh/polygon_sweeping` [20]

Funding This work is partially supported by the Swedish Research Council and the Swedish Transport Administration.

Acknowledgements We thank the anonymous reviewers for their helpful comments.

1 Introduction

We study sweeping a polygonal domain using mobile line segments whose lengths can change. Consider $2k$ agents starting at a given point s (the depot or the base, or the door through which the agents enter) on the boundary of a polygonal domain P. Each agent moves within P, with maximum speed 1. (We do not impose an acceleration bound.) The agents form pairs and each pair a, b defines a sweeping segment whenever the agents see one another; when this is the case, the points along the segment ab are seen by both agents, and we say that the points of the segment ab have been *swept*. The goal is to compute motions (trajectories) for the agents, such that all of P is swept and the agents return to the depot as soon as possible, i.e., minimizing the *makespan* of the sweep, or the time when the agents return to the base after all points of P are swept. See Figure 1 and videos [1, 2].

Search and surveillance are recurring applications of computational geometry: sweeping geometric domains is both a fundamental (in particular, minimum-length sweeping with a line is Problem CG-Open26 in the $\exists\mathbb{R}$ complexity class compendium [34]) and a practical

© Kien C. Huynh, Joseph S. B. Mitchell, and Valentin Polishchuk;
licensed under Creative Commons License CC-BY 4.0

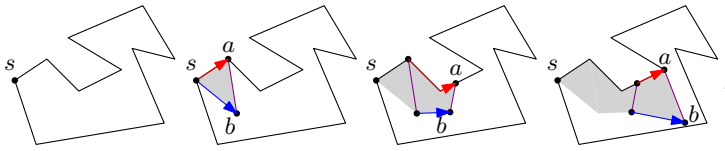


Figure 1 A few steps in the sweeping: white is unswept, gray is swept. Agents a and b start at a point $s \in \partial P$; during the sweep, the agents are not required to stay on the boundary. The red and blue arrows show the trajectories of a and b, respectively.

(e.g., shaving is sweeping the skin area with a segment) problem, which has been studied for decades. Sweeping with a disk or a square is known as milling/lawnmowing [5, 8, 18], and sweeping with a visibility polygon of a mobile observer(s) is the *watchman route problem* (WRP) [10, 29, 30]. Both in WRP and in our problem, the sweep can catch (see) any static target in P, while a mobile target can escape being seen. This is different from, perhaps, a more standard notion of sweeping (by agents with no speed bounds), which ensures that swept points cannot be "recontaminated": here, the archetypal problem is 2-walkability [9,17,21,37] (Can 2 guards sweep a simple polygon by walking along its boundary while always seeing each other?), which was generalized to sweeping with a curve [25, 28, 33] and a chain of guards [12], also in terrains [13, 14]. (Problems in which the target, usually non-static, needs to be "touched," not only seen, in order to be caught, are known as pursuit-evasion games, cops and robbers, lion and man, etc. [3, 11, 16, 22–24, 26, 32, 36].) Sweeping problems may be viewed as far-going extensions of computing the Frechét distance [4] – a classical motion coordination problem. Our solutions apply also to the Laser Tripwire Sensor Deployment for indoor localization and tracking (tripwire localization methods are common and well studied [6, 27]): suppose you have 2 robots that want to deploy a set of tripwire sensors (pairs of points that see each other) – the robots stop every so often (at step length δ) to place the pairs of sensors, and the goal is to complete the deployment as soon as possible.

Our Contributions

We formulate and give the first study of the algorithmic complexity and approximation of the sweeping problem with variable-length line segment(s):

- Problem 1. Given a polygonal domain P, an agent depot $s \in \partial P$, and k pairs of agents: find a motion plan to sweep all points in P and minimize the total time (makespan). Here, each pair of agents have to be covisible at all times (the line segment connecting the pair must stay within P). We call this the "unbreakable" constraint. This problem has two versions:
  - SweepWithReturn: The agents have to return to s (we call this the default version).
  - SweepWithNoReturn: The agents can end anywhere in P.
- Problem 2. The setting and objective are similar to Problem 1. However, in this problem, we do not require the agents to always be covisible but they can only sweep something in P if they become covisible. In addition, the agents themselves also need to stay within P. We call this the BreakableSegmentSweeping problem.

The results in this paper can be seen as contributing to the broader class of optimal coordinated motion planning problems involving mobile agents moving within a domain, with constraints (e.g., covisibility), and an objective to accomplish a covering task.

Specifically,

- Section 2 provides a proof of NP-hardness (from 3-Partition) for all versions, even in the case of an orthogonal, x-monotone simple polygon P (and even if the sweeping segment is required to be vertical at all times).
- Section 3 presents a constant-factor approximation for sweeping simple polygons for Problem 1. Our main technical contribution is the solution for the case of $k = 1$ segment, based on a "window" partition of P that we work out specifically for our purposes (we believe the worked-out decomposition may find also other applications, e.g., when considering similar search problems in terrains); the solution for $k > 1$ (Section 3.2) is obtained by assigning different parts of P to be swept by different pairs of agents.
- In Section 4 the approximation for 1 segment is extended to polygons with holes. The algorithm is simple: turn the domain into a simple polygon by bridging together the holes and the outer boundary of the domain; then apply the algorithm for simple polygons. However, proving that this is an $O(1)$-approximation (i.e., giving a lower bound) is highly non-trivial (and crucially uses the fact that the segment is unbreakable). We leave open sweeping with $k > 1$ segments in polygons with holes.
- Section 5 considers Problem 2. We prove that in a simple polygon relaxing the covisibility constraint does not improve the makespan by more than a factor of 2, and we show that this bound is tight; thus, our constant-factor approximation algorithm for the unbreakable case applies for the breakable case as well. (We note that our NP-hardness proof (Section 2) for unbreakable sweeping of a simple polygon P applies as well to breakable sweeping of P.) In the case that P has holes, we give a polylog approximation algorithm in the special case that P is a polyomino and we sweep with a vertical segment.

In addition to the above results, we also implemented the algorithm described in Section 3 and compared it with a heuristic algorithm on randomly generated data which we show in Appendix D. The codes of the algorithms can be found at [20].

2 NP-Hardness

We show that even in the very special case of sweeping an orthogonal, monotone simple polygon with 2 agents a and b, computing an optimal sweeping schedule is strongly NP-hard. This is in contrast with sweeping a simple polygon with a visibility polygon (WRP), which has a polynomial-time exact algorithm [10, 30], and the sweeping of a simple polygon with a disk or square (the milling problem), which is conjectured to have a polynomial-time exact algorithm [8], as the closely related problems of determining Hamiltonicity of a simple triangular or square grid graph are decidable in polynomial time [7, 40].

▶ **Theorem 1.** *It is NP-hard to compute an optimal sweeping for SweepWithNoReturn, even if P is an x-monotone, orthogonal simple polygon.*

Proof. We first consider the SweepWithNoReturn version where the agents do not have to return to the depot s (so we are minimizing the time when the last point of P is swept). We then show how to modify the reduction to prove hardness of the SweepWithReturn version in which the agents must return to s (and the time of the return defines the makespan). Given an instance of 3-Partition (Can given integers $x_1 \dots x_{3m}$ be partitioned into m triples with equal sums?), we construct a polygon P with $O(m)$ vertices, as shown in Figure 2, consisting of a tall and very narrow rectangular room (whose width is exaggerated in the figure, to show details) with a staircase on the right and with two corridors (each of which

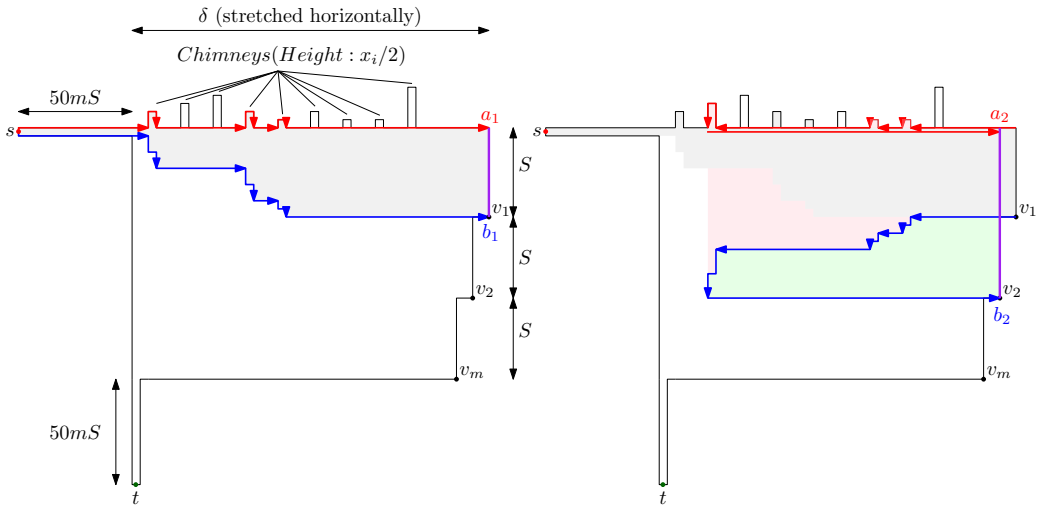


Figure 2 Reducing 3-Partition to sweeping. On the left: the trajectories of the two agents (red and blue arrows) depicting their motions to sweep the first staircase step. After the top agent has swept 3 chimneys corresponding to an arbitrary triple in the 3-Partition solution, they will sweep the first step (including v_1) and end at a_1b_1 (purple). On the right: the motions of the agents in order to sweep the next step. First, the agents will move from a_1b_1 to the left to sweep another triple of chimneys (new areas swept during this motion are painted pink). Then, the agents move to the right to sweep the second staircase step (and v_2) and stop at a_2b_2, the new area swept this time is colored green. Similarly, step m is swept after the top agent has visited the m-th triple.

has length $50mS$) extending from the room as shown. The height of the room is mS where $S = \sum x_i/m$ is the target sum in each of the sought triples of the given numbers; the width, δ, is very small (e.g., $\delta < 1/m^{10}$).

The corridors ending with s and t are very thin and long ($50mS$). Thus, in any optimal sweeping schedule each of the corridors is swept exactly once: the sweep starts at s (by definition) and ends with one of the agents at t (recall that for now we do not restrict where the agents are at the conclusion of the sweep). The top of the room has a chimney for each number in the set. The chimneys are very narrow (width $< \frac{\delta}{3m} < \frac{1}{3m^{11}}$) and chimney i has height $x_i/2$. Finally, the height of each step in the staircase is S.

If the 3-Partition instance is feasible, then P can be swept as follows (with both a and b moving at the maximum, unit speed, and the segment ab staying vertical throughout the motion): the top agent a will visit and ascend chimneys corresponding to triples in the solution. While a is in chimney i (either ascending or descending), the bottom agent b moves down by x_i. After a finishes a triple that sums to S, b has moved down by S. (Note that apart from the duration during which the segment sweeps the horizontal corridor, horizontal movements are not particularly relevant for the analysis, since the room is so narrow, and chimneys are even narrower, implying that the sweep segment must be nearly vertical at all times.) Finally, in this first pass, ab moves all the way to the right, arriving with length S so that it is just long enough to sweep the top step of the staircase (including v_1) without spending further time having agents move vertically. See the left of Figure 2 for an example of this motion to sweep the first step. Next, ab will move to the left of the polygon and sweep more chimneys before moving back to the right to sweep v_2 (Figure 2, right). Such sideways sweeps are repeated until all m of the stairs are swept. (Again, all horizontal motion of agents is not relevant to the overall makespan, since m left-to-right motions are still much less than length 1.) The time to sweep the room, along with the staircases, is at most $mS + 2m\delta$. The total time to sweep everything is $101mS + 2m\delta$.

Conversely, suppose the stairs are swept in any other way, then either b has to wait for at least a duration of 1 before aligning with a stair tread, or a has to waste time by not going into a chimney with a single up-and-down pass. The first case occurs when the agents pick a triplet of chimneys (x, x', x'') with a sum larger than S, then agent a still needs to trace the boundary of the chosen chimneys to sweep them completely. Even if b can already reach the depth of the next staircase vertex (say, v_j), it must still wait for a to finish sweeping the last chimney in the triplet so that it can finally move to sweep v_j, which takes longer than S. In the second case, the chosen triplet of chimneys has a sum smaller than S. In this case, agent a will have to wait for agent b to reach the depth of the next staircase vertex. This still costs the agents a time of S while leaving taller chimneys for other triplets (which leads back to the previous case). Note that there is no reward for a to partially sweep a chimney, since a must exit the chimney and align vertically with b when b touches v_j so that the both of them can sweep the vertical edge incident to v_j. Agent b can technically move diagonally downward to save time, but in that case, the most it can save is δ, which is chosen to be much smaller than $1/m^{10}$ (so in total it can save at most $m \cdot 1/m^{10} = 1/m^9 \ll 1$).

We now modify the above gadget to show the hardness of our SweepWithReturn version. See Appendix A for the modified gadget. As in the proof for the version without return to s, the segment will sweep the top chimneys (solving the 3-Partition) and sweep the convex vertices on the right in parallel. Once the segment is done with the right convex vertices and the top chimneys, the segment sweeps the inverted chimneys (also solving the same 3-Partition instance); each time it sweeps 3 bottom chimneys that make a sum S, the segment will retract itself (top endpoint moving down) so that it could reach the convex vertices on the left $(v'_1, v'_2, \dots)$. The overall makespan will then be $3mS + O(\delta)$. As in the reduction for the version without return to s, any deviation from the described schedule incurs a wait of at least 1 by at least one of the agents. ◀

3 Approximation Algorithm for Simple Polygons

We begin with a simple observation (see Appendix B for the proof):

▶ **Lemma 2.** *Every convex vertex of P must be visited by at least one agent.*

We first consider the case when there are two agents, a and b, starting at the same point $s \in P$: we present an $O(1)$-approximation algorithm to sweep any simple polygon using an unbreakable segment. As with our hardness proof (Section 2), we will first provide algorithmic results for the SweepWithNoReturn version. We show that in linear time, one can find a schedule, for a single unbreakable segment, with makespan $16 \cdot OPT$ where OPT is the optimal makespan. Let $|x|$ denote the Euclidean length of any 1D structure. From Lemma 2,

▶ **Lemma 3.** $OPT \geq \frac{1}{4}|\partial P|$

Proof. By Lemma 2, the union of paths for a and b visits all convex vertices; thus, the union is at least as long as any path within P visiting all convex vertices, implying that the union cannot be shorter than $\frac{1}{2}|\partial P|$. This is because in a simple polygon, ∂P is the shortest tour that visits all convex vertices, and any path that does so must be at least half as long. ◀

Our algorithm partitions the polygon into subpolygons, each of which has a simple sweeping strategy. The dual of the partition is a tree, which allows the line segment to sweep all the subpolygons, traversing the tree recursively.

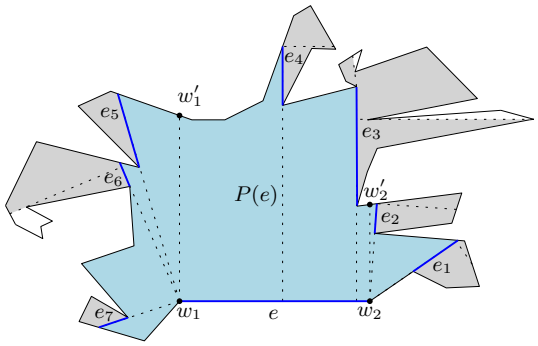


Figure 3 The light blue region is the special visibility polygon $P(e)$, with the base edge $e = w_1w_2$. The additional chords, $e_1, e_2, e_3, \ldots$ also have visibility polygons (shaded gray) associated with them.

The details of the partition are as follows. Consider any edge $e = w_1w_2$ of the polygon. We first compute the histogram polygon with base e, $HP(e)$, which is defined to be the union of all interior chords perpendicular to e and having an endpoint on e (a chord is a line segment connecting two points of ∂P and stays completely within P). Let (w_1, w_1') be such a chord anchored at w_1. We then compute the visibility polygon of w_1 restricted to points that are left of (w_1, w_1'); call the polygon $VP(w_1)$. Similarly, we compute another visibility polygon to the right of (w_2, w_2'), called $VP(w_2)$. Then, $P(e) = HP(e) \cup VP(w_1) \cup VP(w_2)$ is one subpolygon in our partitioning scheme. See Figure 3 for an example of $P(e)$. Notice that the boundary of $P(e)$ includes some new "shadow chords," each of which becomes a histogram base for subsequent subpolygons in the partition. We will order these chords counterclockwise. The process is complete when every shadow chord that has been generated has been utilized as a histogram base for a subpolygon adjacent to the subpolygon that generated the shadow chord. For the first subpolygon, we will simply use the visibility polygon of s. The overall partitioning of P can be done in linear time, utilizing a triangulation of P (similar to computing window partition trees [35] for link distances in simple polygons).

We now describe the recursive process to sweep a subpolygon $P(e)$ and its descendants, for an edge $e = w_1w_2$ of P (refer to Figure 3). The agents start at the endpoint w_1 of e; once the process is completed, i.e. when the segment finishes sweeping $P(e)$ and its descendants, both agents return to w_1. First, a sits at w_1 while b travels clockwise around the boundary of $VP(w_1)$ until reaching w_1'; this way the rotating segment ab sweeps $VP(w_1)$. Next, $HP(e)$ is swept by the segment perpendicular to e: b travels along the top polygonal path of the histogram while a moves along e (a slows itself in order that ab remains perpendicular to e). Then, the segment rotates clockwise around w_2 to sweep $VP(w_2)$; at this point $P(e)$ is completely swept and the segment will move to the furthest (w.r.t. w_2) endpoint of the first "shadow chord" (see e_1 in Figure 3). After the segment completely sweeps $P(e_1)$ (and its descendants), it will move to the next child of $P(e)$. This will be repeated until all descendants of $P(e)$ are swept. Notice that while sweeping the children, the segment will at times move counterclockwise along the boundary of $P(e)$. Once the segment is done with all children of $P(e)$, both of its endpoints will regroup at w_1.

▶ **Lemma 4.** *The sweeping algorithm above produces a schedule with makespan* $\leq 16 \cdot OPT$.

Proof. We will first perform analysis on any subpolygon $P(e)$ other than the root. Without loss of generality, we assume that the segment always starts sweeping from the left vertex of e (i.e., w_1 in the figure). First, the endpoints sweep $P(e)$ by traveling from w_1 to w_2, this will incur a cost of $|\partial P(e)| - |e|$. This is because a will move along e with a speed at most

that of b. Second, the endpoints travel counterclockwise to visit all children of $P(e)$. The cost of this counterclockwise traversal is at most $|\partial P(e)| - |e|$. Overall, the cost of sweeping $P(e)$ and moving counterclockwise to visit all of its children is

$$2(|\partial P(e)| - |e|) = 2\Big(|\partial P(e) \cap \partial P| + \sum_{e_i \in C(e)} |e_i|\Big),$$

where $\partial P(e) \cap \partial P$ are the portions of $\partial P(e)$ that belong to the boundary of P, and $C(e)$ are the base edges of children of $P(e)$. For the root subpolygon $P(s)$, the cost to sweep it and visit all of its children is at most

$$2|\partial P(s)| = 2\Big(|\partial P(s) \cap \partial P| + \sum_{e_i \in C(s)} |e_i|\Big).$$

Here, $C(s)$ is the set of all chords generated and associated with $P(s)$.

Let E be the set of all additional chords created during the partition process. The total cost to sweep P will be the sum of the two terms above:

$$2\Big(\sum_{e \in E}(|\partial P(e) \cap \partial P| + |e|) + |\partial P(s) \cap \partial P|\Big) \leq 4\sum_{e \in E} |\partial P(e) \cap \partial P| + 2(|\partial P(s) \cap \partial P|).$$

We have $|e| \leq |\partial P(e) \cap \partial P|$ for each $e \in E$ because $\partial HP(e) \cap \partial P \subseteq \partial P(e) \cap \partial P$ is an orthogonal projection of e onto ∂P. Thus overall, the sweeping cost of P is at most

$$4\Big(\sum_{e \in E} |\partial P(e) \cap \partial P| + |\partial P(s) \cap \partial P|\Big) \leq 4|\partial P| \leq 16 \cdot OPT.$$

The first inequality holds because the portions of each $P(e)$ that belong to ∂P, i.e. $\partial P(e) \cap \partial P$, are pairwise disjoint for different $e \in E$. Thus, $\sum_{e \in E} |\partial P(e) \cap \partial P| + |\partial P(s) \cap \partial P| = |\partial P|$. The second inequality follows directly from Lemma 3. ◀

3.1 Sweeping with return (default version)

The algorithm above works also when a and b must return to s. The only modification is that at the end of the schedule, both agents must return to s. The analysis remains the same, except that the lower bound of Lemma 3 becomes $OPT \geq \frac{1}{2}|\partial P|$.

▶ **Theorem 5.** *A sweeping schedule, for a single unbreakable segment, with makespan $8 \cdot OPT$ can be found in linear time.*

3.2 Sweeping with multiple segments with return

Suppose now that we have $2k > 2$ agents who start and return to the depot s (the makespan is when the last agent returns to the depot; for $k > 1$ we do not have a solution for the version in which the agents are not required to return to s). Here we have two lower bounds:

- $OPT \geq |\partial P|/(2k)$ where $|\partial P|$ is the perimeter – this follows from Lemma 2 (the $2k$ agents have to visit all convex vertices).
- Let $d_{\max}$ be the geodesic distance from the depot s to the convex vertex furthest from s; then $OPT \geq 2d_{\max}$ – an agent has to visit this furthest convex vertex and go back to the depot.

Partition P into k parts as follows: Pick $k-1$ points on the boundary of P so that these $k-1$ points, together with s, split the boundary of P into k parts of equal perimeter. From the depot, draw the geodesic shortest path to each of these $k-1$ points; the paths decompose P into subpolygons. (Some subpolygons may have very small areas, since the geodesic paths might collapse onto themselves and share edges.)

Because each drawn shortest path is no longer than $d_{\max}$, each subpolygon has perimeter at most $2d_{\max} + |\partial P|/k$. We now simply reuse the approximation algorithm for the single segment case: there, the makespan is at most 4 times the perimeter of the polygon, so for k segments, the makespan will be $4(2d_{\max} + |\partial P|/k) \leq 4 \cdot OPT + 8 \cdot OPT = 12 \cdot OPT$.

4 Approximation Algorithm for Polygonal Domains with Holes

If the domain P has holes, for sweeping with 1 segment ab, we can leverage the result for simple polygons from the preceding section, provided we can reduce the multiply connected case to the simply connected case, via an appropriate bridging of holes to each other and to the outer boundary of P. The main technical challenge is proving lower bounds (Lemma 10) on OPT that enable this approach to yield the desired result – a constant-factor approximation. The version that we shall describe in this section is SweepWithReturn, but the approximation algorithm will also work for SweepWithNoReturn (with a larger multiplicative factor).

First, we compute an approximation of the minimum Steiner tree, MST^*, within P that spans all connected components of the boundary ∂P of P (both the holes and the outer boundary). This can be done using the PTAS in [31]. The computed tree and the boundaries can be viewed as bounding a degenerate simple polygon P' (these polygons are sometimes called "weakly" simple; the boundary cycle of such polygons may go through some points more than once). In the second phase, we apply the algorithm from Section 3 to compute a sweeping solution for P' with makespan $O(|\partial P'|)$, where $\partial P'$ includes the boundaries of the holes and the outer boundary of P, as well as the edges of the spanning tree.

To prove a lower bound of $\Omega(|\partial P'|)$ on OPT, we will need some definitions. A vertex of P is *convex* or *reflex* depending on whether its angle *interior to* P is less than or greater than π. Note that all vertices of a convex hole are actually reflex vertices of P. A *maximal reflex chain* is a chain on ∂P between two consecutive convex vertices; in particular, any edge with two convex endpoints is a maximal reflex chain. Unless otherwise stated, all reflex chains we consider will be maximal reflex chains. Note that a convex hole has no reflex chain.

Given a polygonal domain F, a subset $R \subseteq F$ is *geodesically (or relatively) convex* if for any points $s, t \in R$ the shortest s-t path (within F) lies fully within R. The geodesic (or relative) convex hull R of a set S in F is the smallest geodesically convex set containing S. We will use GCH as a shorthand for geodesic convex hull. Reflex vertices of the hull are reflex vertices of F.

We use *obstacle* to denote a convex hole or a reflex chain. If we need to indicate that we are working with holes or reflex chains, we will use the specific term for them. We say that a set $S(\subseteq F)$ *contains* some obstacles $H_1, \ldots H_m$, if $S \cup (\bigcup_{i=1}^{m} H_i) \subseteq S$ (This containment definition is relaxed from the usual set inclusion definition: since H_i could be a hole and the interior of it does not belong to F, S is not actually a superset of H_i). Similarly, we say $R(\subseteq F)$ is a GCH of some obstacles if R is the smallest geodesically convex set containing all of these obstacles. Note that if H_i is a reflex chain and $H_i \subseteq R$ then $H_i \subseteq \partial R$.

The interior of a GCH needs not to be connected: the boundary of the hull may visit a reflex vertex x of F more than once (so the GCH is a weakly simple polygon) – at some of the visits x is a convex vertex of the hull, while at others it is a reflex vertex (Figure 4,

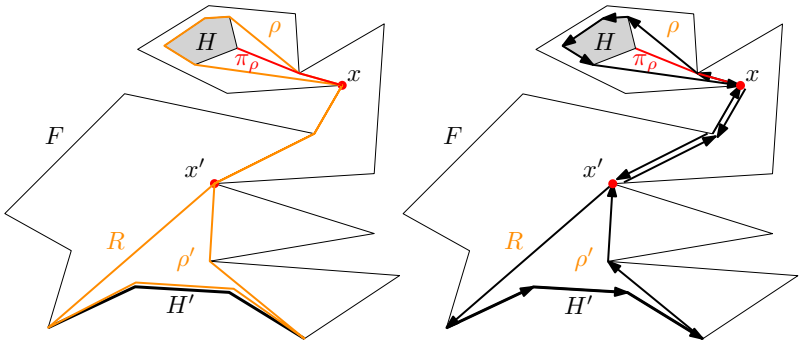


Figure 4 Left: the geodesic convex hull R (drawn in orange) of two obstacles H and H' in a polygonal domain F. H is a hole and H' is a reflex chain. x and x' are collapsing vertices of R. Right: the PSLG constructed by walking around R counterclockwise has two non-empty cycles, which we label as pockets ρ and ρ'.

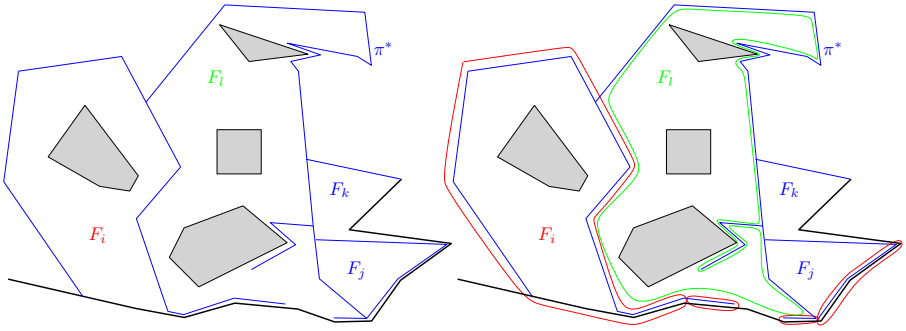


Figure 5 Left: Black is part of ∂P (outer boundary or a non-convex hole), blue is π^*, and gray are convex holes; where blue touches black or gray, they are shown slightly separated for visual clarity. Right: Degenerate faces are circled with red: some faces, e.g., those consisting of a reflex chain followed by π^* have area 0; others, like F_i, have some degenerate parts where π^* follows ∂P. The boundary of F_ℓ (also a degenerate face) is delineated with green; the face contains 3 convex holes, but only one of them (the square) is not touched by blue (for the other two, the blue is slightly pulled away from the holes, for visual clarity).

left). When this occurs, we say that the hull *collapses* onto itself at x and x is a *collapsing* vertex. Consider the directed, planar straight-line graph (PSLG) formed by walking around the boundary of a GCH counterclockwise (Figure 4, right). When the GCH collapses, it will form more than one cycle in the PSLG (since at least one vertex is repeated during the walk). A cycle in this graph can be degenerate (i.e. a cycle from x to x' to x which has an empty area) or non-degenerate. If a region bounded by a cycle contains an obstacle, we will call it a pocket, denoted by ρ. Note that a degenerate cycle can actually be a pocket: it can contain an edge of some obstacle.

Let π^* be the union of the paths for the two segment endpoints, a and b, in an optimal solution. By Lemma 2, the union π^* of the two agents' paths touches all convex vertices of P. In particular, π^*, together with the reflex chains, partitions P into faces $F_1, F_2, \ldots$ (Figure 5, left) such that every convex vertex of every face has at least one incident edge from π^*. Note that in places where π^* follows (hugs) ∂P, some faces are degenerate; a degenerate face (a weakly simple polygon) is created also where π^* touches a convex hole or – more generally – a point of ∂P other than a convex vertex (See Figure 5, right).

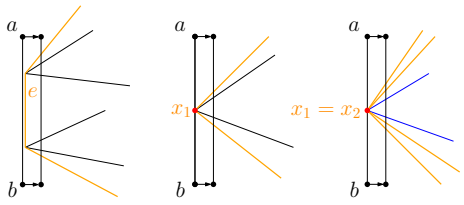


Figure 6 The (interior points of the) segment either enter the geodesic convex hull by aligning with its edge e (in which case the segment starts intersecting a hole when moving into the hull because e is supported by hole vertices), or by moving over a convex vertex x of R_i^1. In the second case, if x is not a collapsing point of the hull, then it must be supported by a hole that ab will intersect after passing over x (middle figure).

The boundary ∂F_i of any face F_i consists of π^*, the reflex chains of ∂P, and convex holes of P; the first two are the outer boundary of F_i. Note that only convex holes of P, if any, remain holes inside the faces; any non-convex hole is split into reflex chains that form boundaries of the faces (if a hole has only one convex vertex we treat it as two convex vertices connected by a degenerate, 0-length edge – so the hole contributes two reflex chains to the partition). For any maximal contiguous portion of ∂P in ∂F_i, it must be a reflex chain: if it contains a convex vertex of P then π^* would need to touch that vertex in order to sweep it.

Starting from here, when we say obstacles in F_i, we refer to convex holes or reflex chains of P in F_i. Let R_i be the GCH of all obstacles in a face F_i, we want to prove the following property of this GCH:

▶ **Lemma 6.** *If R_i contains at least 2 obstacles and does not collapse onto itself anywhere, the segment ab cannot sweep the interior of R_i.*

Proof. By definition of F_i, both endpoints of ab never enter the interior of F_i and R_i; which means the interior of R_i must be swept by interior points of ab. Since the beginning of the schedule, a and b are both at s so they are outside of R_i (the intersection between the segment ab and the interior of R_i is empty). Consider the moment when some interior points of ab enter R_i (Figure 6): the segment is either aligned with an edge of the hull or touches a convex vertex x_1 of the hull. If ab is supported by a (necessarily reflex) vertex of P, then the segment will start intersecting the exterior of P as it moves. In order to *not* intersect with the exterior, the vertex x_1 must be supported by a reflex vertex of F_i, which may only happen if there is another (reflex) vertex x_2 of the hull collocated with x_1. This is due to the fact that any convex vertex of a GCH is either a vertex of P, or is supported by F_i padded by another (reflex) vertex. Therefore, R_i must collapse at some vertex. ◀

Now we have the following result based on the fact that R_i has to collapse:

▶ **Lemma 7.** *If R_i has at least two obstacles and collapses onto itself at some vertices, then it will create at least two pockets.*

Proof. Note that by definition, a pocket must contain at least one obstacle. Let $\mathcal{H}$ be the set of obstacles in R_i, and let ρ be the only pocket that R_i has, then ρ contains all obstacles in $\mathcal{H}$. Consider the PSLG created by walking along ∂R_i counterclockwise: since R_i collapses onto itself at some vertices, it will create more than one cycle in this graph. However, only one cycle (the one bounding ρ) has any obstacle in it, therefore the rest of the cycles in the PSLG are degenerate and empty. But if the PSLG contains all of these degenerate cycles, it means that R_i was not an actual GCH of the obstacles, since we can get rid of all of these degenerate cycles and only keep ρ to get a smaller set that contains all obstacles, which is a contradiction. Therefore, ρ cannot be the only pocket. ◀

Note that the above lemmas only state that R_i will collapse and create at least two pockets, each having at least one obstacle. Now, for each pocket ρ in R_i, we can reapply the same arguments: let R_ρ be the GCH containing all obstacles in ρ then R_ρ must also collapse onto itself somewhere, and create at least two pockets each containing some obstacles. We can keep repeating the same process and each time a GCH collapses it splits the set of obstacles into at least two subsets, each subset belonging to a unique pocket. Since there are a finite amount of obstacles, after a finite amount of these collapsing and splitting steps, each obstacle will be contained in its own pocket (*not necessarily* a pocket of the first GCH R_i):

▶ **Corollary 8.** *Each obstacle in the face F_i is contained in its own pocket of some GCH.*

Next, we obtain the following upper bound on the total perimeter of the pockets:

▶ **Lemma 9.** *Let Φ_i be the set of all pockets in F_i such that each pocket contains one obstacle:*

$$\sum_{\rho \in \Phi_i} |\partial \rho| \leq |R_i|.$$

Proof. To prove this, we define a tree structure where each node is either a GCH or a pocket. We call R_i the root GCH, and the pockets resulted from the collapsing of R_i its child pockets. For each pocket, we call the GCH containing all of its obstacles its child GCH. Similarly, each child GCH also has its child pockets. Note that a GCH node could have multiple children (pockets) but a pocket can only have one child. In this tree, only pocket nodes can be leaf nodes. A leaf pocket node is one in which there is exactly one obstacle in it. We use $child(\cdot)$ to denote the set of children of a node. For any GCH node R, we have the inequality

$$\sum_{\rho \in child(R)} |\partial \rho| \leq |\partial R|.$$

This is because each pocket is a different part of R (two pockets share at most one vertex in R), therefore their total perimeter must be less than $|\partial R|$. Next, for any pocket node:

$$|\partial child(\rho)| \leq |\partial \rho|.$$

This is because $child(\rho)$ is a GCH in ρ therefore its perimeter cannot be larger than $|\partial \rho|$. The tree we have now is one such that the perimeter of each node is greater than or equal to the total perimeter of its children, therefore we can conclude that the perimeter of any GCH node is greater than or equal to the total perimeter of all of its descendant leaf pocket nodes. This applies to the root GCH R_i as well, which proves the lemma. ◀

We finally prove the lower bound for the approximation algorithm:

▶ **Lemma 10.** *We have $|\partial P| \leq 2|\pi^*| \leq 4 \cdot OPT$ and $|MST^*| \leq 2|\pi^*| \leq 4 \cdot OPT$.*

(Recall that MST^* is the minimum spanning tree with Steiner points that spans all holes and the outer boundary of P.)

Proof. Let Φ_i be the set of pockets where each pocket contains exactly one obstacle in F_i. If an obstacle H is a convex hole and ρ is its pocket, then $|\partial H| \leq |\partial \rho|$. However, if H is a reflex chain, then $2|\partial H| \leq |\partial \rho|$: this is because the perimeter of the GCH R of H in ρ is twice longer than ∂H (a GCH of a reflex chain is simply the chain traversed back and forth). We will denote $\mathcal{H}_h$ (resp. $\mathcal{H}_r$) as the set of convex hole (resp. reflex chain) obstacles in F_i. We acquire the following using Lemma 9:

$$\sum_{H \in \mathcal{H}_h} |\partial H| + 2 \sum_{H \in \mathcal{H}_r} |\partial H| \leq \sum_{\rho \in \Phi_i} |\partial \rho| \leq |\partial R_i| \leq |\partial F_i|. \tag{1}$$

Let $\pi_i = \pi^* \cap \partial F_i$ (the portion of π^* in ∂F_i), then by definition $\partial F_i = \pi_i \cup (\bigcup_{H \in \mathcal{H}_r} \partial H)$ so $|\partial F_i| = |\pi_i| + \sum_{H \in \mathcal{H}_r} |\partial H|$. Using the above inequality we obtain:

$$\sum_{H \in \mathcal{H}_h} |\partial H| + \sum_{H \in \mathcal{H}_r} |\partial H| \leq |\pi_i|.$$

Summing over all F_i, we achieve the first lower bound of the lemma:

$$|\partial P| \leq 2|\pi^*| \leq 4 \cdot OPT.$$

Note that $|\partial P| \leq 2|\pi^*|$ because any portion of π^* is counted twice during the sum: any edge of π^* is shared by at most two different faces.

To prove the second bound in the lemma, we introduce a way to connect all of the *convex holes* to their respective ∂F_i such that the total cost of these connections is bounded by $O(|\pi^*|)$. These connections, along with π^*, will serve as a network connecting all components of the boundary of P, and the total length of this network cannot be shorter than the length of MST^* (the minimum Steiner tree connecting these components), which will give us the second bound of the lemma. We only consider convex holes in this case because any reflex chain obstacle is already connected to π^* by definition.

Recall that when R_i collapses (and creates pockets), there will be at least one collapsing vertex which is supported by a reflex vertex of either ∂P or π^*. Let x be any collapsing vertex that belongs to pocket ρ and H be the convex hole in ρ, and let π_ρ be the (geodesic) shortest path from H to x (see Figure 4 for an illustration). π_ρ is the shortest path between a point $u \in \partial H$ and x. We note that u is the only point shared between π_ρ and ∂H because if there is another point $w \in \pi_\rho \cap \partial H$ then π_ρ is suboptimal (we can simply remove the subpath from u to w to get a better path). Therefore, if we remove H from ρ and compute the shortest path between u and x_ρ it will stay the same. Since π_ρ is the shortest path between two points in a simple polygon (ρ without H), it cannot be longer than a half of $\partial\rho$:

$$2|\pi_\rho| \leq |\partial\rho|. \tag{2}$$

Let Φ_h (resp. Φ_r) be the set of all pockets of the convex holes (resp. reflex chains) in F_i. Note that $\Phi_h \cup \Phi_r = \Phi_i$ and $\Phi_h \cap \Phi_r = \emptyset$. Using (1) and (2):

$$2 \sum_{\rho \in \Phi_h} |\pi_\rho| + \sum_{\rho \in \Phi_r} |\partial\rho| \leq \sum_{\rho \in \Phi_h} |\partial\rho| + \sum_{\rho \in \Phi_r} |\partial\rho| \leq |\partial F_i| = |\pi_i| + \sum_{H \in \mathcal{H}_r} |\partial H|.$$

Recall that for each pocket of a reflex chain obstacle, the perimeter of the pocket is at least double that of the reflex chain so $2 \sum_{H \in \mathcal{H}_r} |\partial H| \leq \sum_{\rho \in \Phi_r} |\partial\rho|$, therefore:

$$\begin{aligned} &2 \sum_{\rho \in \Phi_h} |\pi_\rho| + 2 \sum_{H \in \mathcal{H}_r} |\partial H| \leq 2 \sum_{\rho \in \Phi_h} |\pi_\rho| + \sum_{\rho \in \Phi_r} |\partial\rho| \leq |\pi_i| + \sum_{H \in \mathcal{H}_r} |\partial H| \\ &\Rightarrow \sum_{\rho \in \Phi_h} |\pi_\rho| \leq \frac{1}{2}\Big(|\pi_i| - \sum_{H \in \mathcal{H}_r} |\partial H|\Big) \leq \frac{1}{2}|\pi_i|. \end{aligned}$$

Let Φ be the set of all pockets containing convex holes of every face, by summing the above inequality over all F_i we obtain $\sum_{\rho \in \Phi} |\pi_\rho| \leq |\pi^*|$. Finally, the length of the network connecting all boundary components, i.e. the union $\pi^* \cup (\bigcup_{\rho \in \Phi} \pi_\rho)$, is $L = \sum_{\rho \in \Phi} |\pi_\rho| + |\pi^*| \leq 2|\pi^*|$. Since L cannot be shorter than MST^*:

$$|MST^*| \leq L \leq 2|\pi^*| \leq 4 \cdot OPT. \qquad \blacktriangleleft$$

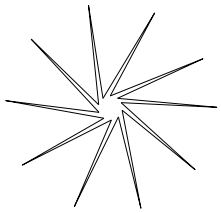


Figure 7 Pinwheel polygons show that T can be arbitrarily close to $2T'$.

With the above lemma, the perimeter of the degenerate polygon P' is $|\partial P'| \leq (12+\epsilon)\cdot OPT$ for any $\epsilon > 0$ (note that we need to double MST^* to create P'), assuming we are using a PTAS to compute the minimum Steiner tree connecting all boundaries of P. Based on Lemma 4, the schedule computed on P' using the approximation algorithm in Theorem 5 would yield a makespan within $4|\partial P'| \leq (48 + 4\epsilon) \cdot OPT$. Thus, we obtain the theorem:

▶ **Theorem 11.** *A $(48 + \epsilon)$-approximation for sweeping a polygonal domain with holes where the agents must return to their depot can be computed in polynomial time.*

5 Breakable Segment Sweeping

We consider the breakable segment sweeping problem in which the agents defining the endpoints of the sweeping segments must stay within P, but we do not require that they remain covisible at all times; however, a point $p \in P$ is swept only if p lies on the segment ab at a moment when the agents a, b are covisible (i.e., $ab \subset P$).

5.1 Comparison with the unbreakable case

Allowing the sweeping segment to break can allow for a significantly reduced makespan. We first consider the case of a simple polygon P, for which we can bound the ratio of makespans between the breakable and unbreakable segment cases.

▶ **Lemma 12.** *Given a simple polygon P, let T (resp. T') be the optimal makespan of an unbreakable (resp. breakable) sweeping schedule. Then, $T' \leq T < 2T'$.*

The proof of this lemma can be found in Appendix C.

The upper bound in the lemma is tight: consider pinwheel polygons in which the edges are angled so that no two convex vertices can see each other (Figure 7). To sweep this polygon, an unbreakable segment has to spend about $|\partial P \setminus \pi_r|$ where π_r is the longest reflex chain. This is because after sweeping one pin, one endpoint has to wait for the other before proceeding into the next pin. For a breakable segment, it only takes $0.5|\partial P \setminus \pi_r|$ to sweep all pins because once an endpoint of the segment reaches a convex vertex in a pin, the other endpoint can start moving into an adjacent pin in parallel. This leaves an unswept region in the center with the shape of a regular $\frac{n}{2}-$gon. The perimeter of this region can be made arbitrarily shorter than the length of the pins so that it only takes a tiny amount of time for the breakable segment to clean it. Overall, the ratio T/T' would approach 2 if we keep adding more pins and stretching them.

Note that Lemma 12 does not apply to polygons with holes since a breakable segment is allowed to pass through them while an unbreakable one is not. Figure 8 shows an example in which T can be worse than T' by a factor of $\Omega(|\mathcal{H}|)$ where $\mathcal{H}$ is the set of holes in the polygon. By Lemma 10, the optimal schedule of an unbreakable segment has makespan

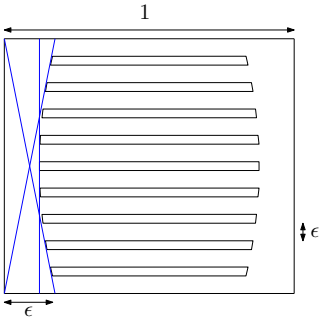


Figure 8 A square of side length 1 containing $|\mathcal{H}|$ horizontal rectangular holes of width at least $1 - \epsilon$. The length ϵ is not drawn to scale. The left (and right) edges of the bars are angled in such a way that it only takes $O(\epsilon)$ for a segment to sweep the portions left/right of the holes, if the segment begins as a unit-length segment along the left/right side of the square.

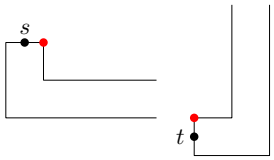


Figure 9 Two notches (small squares) are attached to the end of the thin hallways in the gadget shown in Figure 2. To sweep the red convex vertices, both agents are forced to travel to the end of each hallway, incurring a cost of about $100ms$ in the makespan.

at least $0.25 \times 2(1 - \epsilon)|\mathcal{H}| = 0.5(1 - \epsilon)|\mathcal{H}|$. However, a breakable segment can sweep this polygon in a time of at most $3.5 + O(\epsilon)$: The agents begin at the midpoint of the left side of P, move apart vertically to extend along that unit-length side, spend time $O(\epsilon)$ to sweep the portion of P to the left of the holes, including the left sides of the holes, then move to the right side (while not sweeping points, as the segment passes through holes), sweep the portion on the right in time $O(\epsilon)$, then reposition in time 1 to lie along the top edge of P, and finally sweep downwards in time 1 to complete the sweep of the corridors between holes.

5.2 Hardness and approximation

As in the case of sweeping with an unbreakable segment, the problem of makespan optimization is NP-hard for sweeping a simple polygon with a breakable segment:

▶ **Theorem 13.** *It is NP-hard to compute an optimal sweeping schedule of a simple polygon P using a breakable line segment, even if P is an orthogonal simple polygon.*

Proof. We use reduction from 3-Partition, similar to the proof of Theorem 1. We modify the gadget by adding a small notch at the end of each narrow corridors (Figure 9), forcing both agents to travel the entire lengths of these corridors in order to sweep the notches. ◀

The algorithm given in Section 3 applies in the breakable segment sweeping case, yielding a 9-approximation when P is a simple polygon, as the same lower bound applies. For the case in which P has holes, we take a different approach to obtain a weaker and more specialized approximation result than we obtained in the unbreakable case.

Consider the case in which P is a polyomino, a union of integer-coordinate unit squares whose planar dual graph, joining pairs of squares ("pixels") that share a common unit-length edge, is connected. Consider the case in which we are to sweep P with a single

breakable vertical segment. The problem is strongly NP-hard, even if P is a simply connected polyomino – the polygon in Figure 2 from our hardness proof in Section 2 can be turned into a polyomino. Let N be the number of pixels making up P:

▶ **Theorem 14.** *There exists a polynomial-time algorithm (in N) achieving polylogarithmic approximation of the minimum makespan sweep of a polyomino using a breakable vertical sweeping segment.*

See Appendix C for details of the proof. As pointed out by a reviewer of a previous version of the paper, this result can be extended to polyominos built from rectangles whose perimeter is bounded by a constant. The above algorithm also applies to sweeping a polyomino with an axis-aligned (horizontal or vertical) breakable segment: in this version, the agents can move arbitrarily but the sweep coverage happens only when the agents are covisible and aligned vertically or horizontally.

6 Conclusion

We introduced a novel motion planning problem where two (or multiple pairs of) agents can sweep a polygonal domain with their line of sight. The problem is strongly NP-hard, and we designed constant-factor approximation algorithms for both simple domains and domains with holes. We intend to improve the constants in these algorithms in our future work, either by a tighter analysis or better methods to decompose the polygon. Many other variants of the problem are to be considered as well, e.g., the starting point can be in the interior of P or no starting point was given. One may also want to limit the maximum distance of the pair, as this is a realistic assumption. In the multiple-segment case, if the agents are allowed to change the pairing, the problem will become much more complex as well as interesting (we thank our reviewer for raising this question).

References

1 https://emalm.com/?v=38JGp.

2 https://emalm.com/?v=8JzrX.

3 Mikkel Abrahamsen, Jacob Holm, Eva Rotenberg, and Christian Wulff-Nilsen. Best laid plans of lions and men. In Boris Aronov and Matthew J. Katz, editors, *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, volume 77 of *LIPIcs*, pages 6:1–6:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.SoCG.2017.6.

4 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5:75–91, 1995. doi:10.1142/S0218195995000064.

5 Esther M Arkin, Michael A Bender, Erik D Demaine, Sándor P Fekete, Joseph S. B. Mitchell, and Saurabh Sethia. Optimal covering tours with turn costs. *SIAM Journal on Computing*, 35(3):531–566, 2005. doi:10.1137/S0097539703434267.

6 Esther M Arkin, Rathish Das, Jie Gao, Mayank Goswami, Joseph SB Mitchell, Valentin Polishchuk, and Csaba D Tóth. Cutting polygons into small pieces with chords: Laser-based localization. In *28th Annual European Symposium on Algorithms (ESA 2020)*. Schloss Dagstuhl – Leibniz Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.7.

7 Esther M Arkin, Sándor P Fekete, Kamrul Islam, Henk Meijer, Joseph S. B. Mitchell, Yurai Núñez-Rodríguez, Valentin Polishchuk, David Rappaport, and Henry Xiao. Not being (super) thin or solid is hard: A study of grid hamiltonicity. *Computational Geometry*, 42(6-7):582–605, 2009. doi:10.1016/J.COMGEO.2008.11.004.

8 Esther M Arkin, Sándor P Fekete, and Joseph S. B. Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17(1-2):25–50, 2000. doi:10.1016/S0925-7721(00)00015-8.

9 Binay Bhattacharya, Asish Mukhopadhyay, and Giri Narasimhan. Optimal algorithms for two-guard walkability of simple polygons. In *Workshop on Algorithms and Data Structures*, pages 438–449. Springer, 2001. doi:10.1007/3-540-44634-6_40.

10 Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph S. B. Mitchell. Touring a sequence of polygons. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 473–482, 2003. doi:10.1145/780542.780612.

11 Adrian Dumitrescu, Ichiro Suzuki, and Paweł Żyliński. Offline variants of the "lion and man" problem – Some problems and techniques for measuring crowdedness and for safe path planning. *Theoretical Computer Science*, 399(3):220–235, 2008. doi:10.1016/j.tcs.2008.02.039.

12 Alon Efrat, Leonidas J Guibas, Sariel Har-Peled, David C Lin, Joseph S. B. Mitchell, and TM Murali. Sweeping simple polygons with a chain of guards. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 927–936, 2000.

13 Alon Efrat, Joseph S. B. Mitchell, Swaminathan Sankararaman, and Parrish Myers. Efficient algorithms for pursuing moving evaders in terrains. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 33–42, 2012. doi:10.1145/2424321.2424327.

14 Alon Efrat, Mikko Nikkilä, and Valentin Polishchuk. Sweeping a terrain by collaborative aerial vehicles. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 4–13, 2013. doi:10.1145/2525314.2525355.

15 Naveen Garg, Goran Konjevod, and Ramamoorthi Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. *Journal of Algorithms*, 37(1):66–84, 2000. doi:10.1006/JAGM.2000.1096.

16 Leonidas J Guibas, Jean-Claude Latombe, Steven M LaValle, David Lin, and Rajeev Motwani. A visibility-based pursuit-evasion problem. *International Journal of Computational Geometry & Applications*, 9(04n05):471–493, 1999. doi:10.1142/S0218195999000273.

17 Paul J Heffernan. An optimal algorithm for the two-guard problem. In *Proceedings of the Ninth Annual symposium on computational geometry*, pages 348–358, 1993. doi:10.1145/160985.161163.

18 Martin Held. Survey of direction-parallel milling. In *On the Computational Geometry of Pocket Machining*, chapter 3, pages 37–52. Springer, 2005.

19 Stefan Hertel and Kurt Mehlhorn. Fast triangulation of simple polygons. In *Foundations of Computation Theory: Proceedings of the 1983 International FCT-Conference Borgholm, Sweden, August 21–27, 1983 4*, pages 207–218. Springer, 1983. doi:10.1007/3-540-12689-9_105.

20 Kien C. Huynh. Polygon sweeping with line of sight between covisible agents. Software (visited on 2025-08-19). URL: https://github.com/KienHuynh/polygon_sweeping, doi:10.4230/artifacts.24586.

21 Christian Icking and Rolf Klein. The two guards problem. *International Journal of Computational Geometry & Applications*, 2(03):257–285, 1992. doi:10.1142/S0218195992000160.

22 Rufus Isaacs. *Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization*. Courier Corporation, 1999.

23 Bo Jiang and Xuehou Tan. Searching for mobile intruders in circular corridors by two 1-searchers. *Discrete applied mathematics*, 159(16):1793–1805, 2011. doi:10.1016/J.DAM.2010.10.007.

24 Tsunehiko Kameda, Ichiro Suzuki, and Masafumi Yamashita. An alternative proof for the equivalence of ∞-searcher and 2-searcher. *Theoretical Computer Science*, 634:108–119, 2016. doi:10.1016/J.TCS.2016.04.016.

25 Borislav Karaivanov, Minko Markov, Jack Snoeyink, and Tzvetalin S Vassilev. Decontaminating planar regions by sweeping with barrier curves. In *CCCG*, 2014.

26 Steven M LaValle, Borislav H Simov, and Giora Slutzki. An algorithm for searching a polygonal region with a flashlight. In *Proceedings of the sixteenth annual symposium on Computational geometry*, pages 260–269, 2000. doi:10.1145/336154.336212.

27 Kin Sum Liu, Brent Schiller, Jie Gao, Shan Lin, and Joseph SB Mitchell. Optimizing sensor deployment with line-of-sight constraints: Theory and practice. In *EWSN*, pages 95–105, 2019.

28 Minko Markov, Vladislav Haralampiev, and Georgi Georgiev. Lower bounds on the directed sweepwidth of planar shapes. *Serdica Journal of Computing*, 9(2):151–166, 2015.

29 Joseph S. B. Mitchell. Approximating watchman routes. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 844–855. SIAM, 2013. doi:10.1137/1.9781611973105.60.

30 Bengt J Nilsson and Eli Packer. An approximation algorithm for the two-watchman route in a simple polygon. In *European Workshop on Computational Geometry, EuroCG, Lugano, Switzerland (20160330-20160401)*, pages 111–114, 2016.

31 J Scott Provan. An approximation scheme for finding steiner trees with obstacles. *SIAM Journal on Computing*, 17(5):920–934, 1988. doi:10.1137/0217057.

32 Günter Rote. Pursuit-evasion with imprecise target location. In *SODA*, pages 747–753, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644231.

33 Dorian Rudolph. Approximating the sweepwidth of polygons with holes. In *EuroCG*, 2019.

34 Marcus Schaefer, Jean Cardinal, and Tillmann Miltzow. The existential theory of the reals as a complexity class: A compendium. *arXiv preprint arXiv:2407.18006*, 2024. doi:10.48550/arXiv.2407.18006.

35 Subhash Suri. On some link distance problems in a simple polygon. *IEEE transactions on Robotics and Automation*, 6(1):108–113, 1990. doi:10.1109/70.88124.

36 Ichiro Suzuki and Masafumi Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on computing*, 21(5):863–888, 1992. doi:10.1137/0221051.

37 Xuehou Tan and Bo Jiang. Minimization of the maximum distance between the two guards patrolling a polygonal region. *Theoretical Computer Science*, 532:73–79, 2014. doi:10.1016/J.TCS.2013.03.019.

38 The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.6 edition, 2023. URL: https://doc.cgal.org/5.6/Manual/packages.html.

39 Auer Thomas and Held Martin. Heuristics for the generation of random poly-gons. In *Proceedings of the 8th Canadian Conference on Computational Geometry (CCCG'96)*, pages 38–44. Citeseer, 1996. URL: http://www.cccg.ca/proceedings/1996/cccg1996_0007.pdf.

40 Christopher Umans and William Lenhart. Hamiltonian cycles in solid grid graphs. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 496–505. IEEE, 1997. doi:10.1109/SFCS.1997.646138.

A Gadget for NP-hardness proof of SweepWithReturn

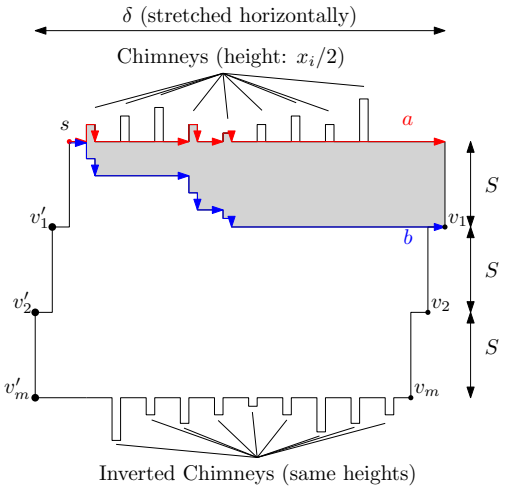


Figure 10 Modified polygonal gadget for the hardness of SweepWithReturn. s is at the top left of the polygon.

B Proof of Lemma 2

We restate the lemma here:

▶ **Lemma 2.** *Every convex vertex of P must be visited by at least one agent.*

Proof. In order for a convex vertex v to be swept, the point v must at some moment lie on the sweeping segment. If v is not visited by an endpoint of the sweeping segment, then it must be visited at some moment by a point interior to the sweeping segment. But this implies that, at that moment, in the neighborhood of v the sweeping segment, ab, enters the exterior of P, since the two collinear segments av and vb cannot both lie within the convex cone with apex v that is defined by the edges of P incident to v. ◀

C Breakable Segment Sweeping

▶ **Lemma 12.** *Given a simple polygon P, let T (resp. T') be the optimal makespan of an unbreakable (resp. breakable) sweeping schedule. Then, $T' \leq T < 2T'$.*

Proof. The first inequality ($T' \leq T$) is immediate. We will now provide proof that $T < 2T'$: given any sweeping schedule of a breakable segment, we can convert it into a schedule for an unbreakable segment, while at most doubling the makespan. Recall that the two endpoints of the breakable segment might, at some times, not see each other; however, points of P are only being swept during times when the endpoints are covisible. Therefore, a schedule of a breakable segment can be seen as a sequence of periods (intervals of time) during each of which the endpoints are either always covisible or always hidden from each other. During a covisible period, an unbreakable segment can simply follow the sweeping schedule of the breakable segment (since it is not broken during this time interval), with no change in makespan. For any hidden period, let $a'_i b'_i$ be the first position of the breakable segment and $a'_j b'_j$ be the last. If T'_{ij} is the makespan of the breakable segment in this period, then

$$T'_{ij} \geq \max\{|\pi(a'_i, a'_j)|, |\pi(b'_i, b'_j)|\}, \tag{3}$$

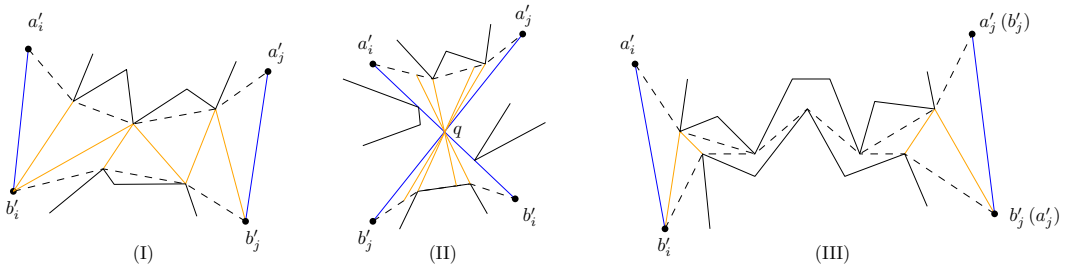


Figure 11 Three cases that can happen when $a'b'$ moves from $a'_ib'_i$ to $a'_jb'_j$. The dashed lines denote the geodesic shortest paths between the pairs of points, the black polygonal lines represent the boundary of P and the orange lines show how we decompose each case into sweepable components for the unbreakable segment.

where $\pi(a'_i, a'_j)$ and $\pi(b'_i, b'_j)$ are the geodesic shortest paths from a'_i to a'_j and b'_i to b'_j respectively.

There are now three cases (Figure 11):

(I) $a'_ib'_i$ does not intersect $a'_jb'_j$ and neither do $\pi(a'_i, a'_j)$ and $\pi(b'_i, b'_j)$. The union $a'_ib'_i \cup a'_jb'_j \cup \pi(a'_i, a'_j) \cup \pi(b'_i, b'_j)$ will bound a region P_1, which is an hourglass polygon (i.e. a polygon consisting of two reflex chains that are connected by two edges $a'_ib'_i$ and $a'_jb'_j$). Now, consider the triangulation of the hourglass polygon. Let the triangle containing $a'_ib'_i$ be $\triangle_i$ and the triangle containing $a'_jb'_j$ be $\triangle_j$. If we consider each triangle as a node of a graph and create an edge between two nodes if their corresponding triangles share an edge in the triangulation, then this graph (the dual of the triangulation) is a path from node $\triangle_i$ to node $\triangle_j$. This is always true for an hourglass polygon. Note that other than $\triangle_i$ and $\triangle_j$, a triangle in the triangulation consists of exactly two chords of P_1 and one edge on its boundary (an edge of $\pi(a'_i, a'_j)$ or $\pi(b'_i, b'_j)$). To let the unbreakable segment sweep this hourglass, we use the following motion plan:

- If ab lines up with an edge of some triangle $\triangle_k$, this edge must be $a'_ib'_i$, $a'_jb'_j$ or a chord of P_1, but not an edge of $\pi(a'_i, a'_j)$ or $\pi(b'_i, b'_j)$.
- Assume ab is lining up with one edge of an unswept triangle $\triangle_k$, either a or b must be incident to an edge of $\pi(a'_i, a'_j)$ or $\pi(b'_i, b'_j)$. Whoever is incident to such an edge will move along that edge while the other stays stationary during the whole period. This will sweep $\triangle_k$ and let ab line up with an edge of the next triangle.
- Repeat the above step until ab lines up with $a'_jb'_j$.

This whole process will cost no more than $|\pi(a'_i, a'_j)| + |\pi(b'_i, b'_j)|$.

(II) $a'_ib'_i$ intersects with $a'_jb'_j$. Let $q = a'_ib'_i \cap a'_jb'_j$. $\pi(a'_i, a'_j)$ and $\pi(b'_i, b'_j)$ will be two reflex chains visible from q. We can view $a'_ib'_i \cup \pi(a'_i, a'_j) \cup a'_jb'_j \cup \pi(b'_i, b'_j)$ as a self-intersecting polygonal chain, or the boundary of a simple polygon with a degenerate vertex q. To move from $a'_ib'_i$ to $a'_jb'_j$ using an unbreakable segment, we can rotate the segment around q while a moves along $\pi(a'_i, a'_j)$ and b moves along $\pi(b'_i, b'_j)$ in opposite directions. This can be done as follows: we project all vertices from each reflex chain onto the other chain using q as a pinhole. The projections decompose the degenerate polygon into smaller double wedges, which can be swept one by one. Each such wedge has two bases, one of which is at least as long as the other. Sweeping a double wedge will require a time equal to the length of the longer base.

(III) $\pi(a'_i, a'_j)$ and $\pi(b'_i, b'_j)$ share some vertices. In this case, the two of them will create two funnels whose bottom vertices are attached to the two ends of the shared portion. Similarly to the hourglass case in (I), we can triangulate the funnels and:

- Let the segment sweep the first funnel polygon by moving from one triangle to another. Both a and b will eventually meet at the bottom vertex of the first funnel polygon.
- Since a and b are now colocated, they can both move along the shared portion of $\pi(a'_i, a'_j)$ and $\pi(b'_i, b'_j)$ until both of them arrive at the bottom endpoint of the second funnel polygon.
- Finally, the segment ab can sweep the second funnel polygon by moving from one triangle to another until they line up with $a'_j b'_j$.

This process also costs at most $|\pi(a'_i, a'_j)| + |\pi(b'_i, b'_j)|$.

In all cases, the produced schedule for the unbreakable segment is at most $2T'_{ij}$ due to (3). ◀

▶ **Theorem 14.** *There exists a polynomial-time algorithm (in N) achieving polylogarithmic approximation of the minimum makespan sweep of a polyomino using a breakable vertical sweeping segment.*

Proof. At any moment in time, the sweeping segment ab has a *combinatorial type*, (p, q), indicating the pixel p currently containing a and the pixel q currently containing b (possibly $p = q$). At time 0, the combinatorial type is (p_s, p_s), where p_s is the pixel on whose boundary the starting point s lies. For each pixel $\gamma \in P$, in order for points in γ to be swept by ab, at least one of the set $\mathcal{C}_\gamma$ of combinatorial types must be visited, where $\mathcal{C}_\gamma$ is the set of combinatorial types (p, q) for which pq is a vertical column of pixels within P that contains γ (i.e., $\gamma \in pq$). Over the time window $[0, T]$ of an optimal sweep, the segment ab has a sequence, possibly with repeats, of combinatorial types, $(p_1, q_1), (p_2, q_2), \ldots, (p_k, q_k)$; this sequence must contain at least one combinatorial type from $\mathcal{C}_\gamma$, for every $\gamma \in P$. (Note that some of the types (p_i, q_i) may specify vertical columns of pixels that do not all lie within P; we do allow ab to lie partially outside of P in the breakable segment sweeping problem.) By continuity of motion of the agents a and b, for each i, the L_1 distance between pixel p_i and p_{i+1}, and between q_i and q_{i+1}, is at most 1 (it is 0 or 1). We define the distance between (p_i, q_i) and (p_{i+1}, q_{i+1}) to be $\max\{|p_i p_{i+1}|_1, |q_i, q_{i+1}|_1\}$. Then, in this metric, we consider the one-of-a-set (generalized) TSP path, to visit every set $\mathcal{C}_\gamma$, for $\gamma \in P$. The optimal makespan T must be at least the length of such a TSP path. Further, for any TSP path in the "combinatorial type space" that visits all of the sets $\mathcal{C}_\gamma$, we can, with only an $O(1)$ factor greater length, execute a sweep with a vertical segment ab, with $a \in p_i$ and $b \in q_i$, shifting the vertical segment (of fixed length) around the boundaries of pixels p_i and q_i, so that all points in the vertical column $p_i q_i$ of pixels within P are swept. Since the one-of-a-set TSP has a polylog factor approximation algorithm [15], applying it here will result in Theorem 14. ◀

D Experiments

Sweeping a simple polygon with 1 segment can be done with the following heuristic which, similar to our approximation algorithm (Section 3), has two phases: first, decompose P into convex pieces ($O(n)$ time [19]) – the dual graph of the decomposition is a tree; then, sweep the pieces one-by-one in DFS order around the tree, starting with the component that contains s (Figure 12 shows that the heuristic's makespan can be $\Theta(n \cdot OPT)$). We implemented one-segment sweeping with return both with our algorithm (Section 3) and the above heuristic; Python was used for the main procedures, while C++ was used as a proxy for calling CGAL modules [38]. We generated 3 sets of random simple polygons (the vertices are either within the unit square or unit disk), using the algorithm of [39] (for each

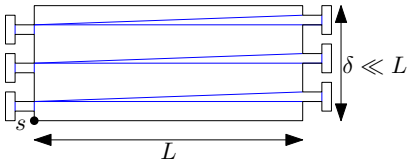


Figure 12 P is a large central room, with small rooms (nooks) attached on the left/right sides; sweeping the large central room first and then sweeping the small rooms on the left and then those on the right takes $O(L + n\delta) = O(L)$ time if $\delta << L$. The convex decomposition (blue segments) splits the middle room into convex pieces: sweeping every node in the dual tree (i.e., using the heuristic) forces the agents to move from left to right and vice versa $\Theta(n)$ times, taking $\Theta(nL)$ time.

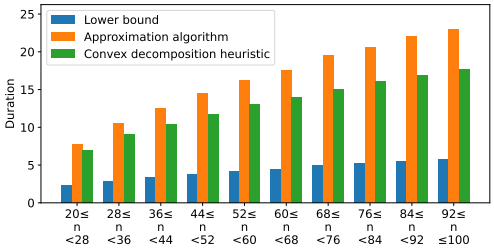
polygon, s is a random vertex): 1000 polygons with 20 to 100 vertices each, 500 polygons with 500 to 1000 each, and 50 polygons with 5000 to 10000 vertices each. The results of experimenting with the polygons are shown in Fig. 13: it can be seen that although the heuristic algorithm (HA) has no approximation guarantee, it outperforms the (provable) approximation algorithm (AA) in most instances (the heuristic takes better advantage of parallelism during the second step, by moving both agents at maximum speed when they are able to do so; in the approximation algorithm, the agents move in parallel only when sweeping histogram subpolygons $HP(e)$, and even then the bottom agent slows down in order to accommodate the top agent).

Table 1 shows the average and maximum ratios of the makespans of AA and HA; the average speedup of HA is about 1.333 and the maximum speedup observed is 1.743. The average ratio of the AA makespans and the theoretical lower bounds is between 3.903 and 4.032. In the worst instance of the experiments, the schedule computed by AA takes 5.116 more time than the proven lower bound; this observed factor is still significantly less than the factor of 8 proved in Theorem 5; we believe that the reason is that the theoretical lower bound is usually unachievable (except for some specific classes of polygons like convex).

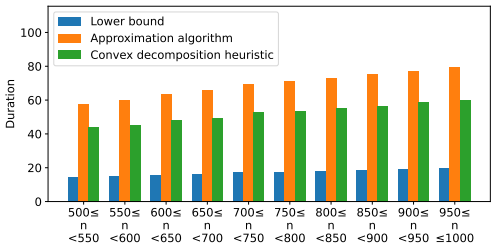
Table 1 Average and maximum makespan ratios between AA, HA, and the lower bound (LB).

Makespan ratio		AA/LB	HA/LB	AA/HA
Mean	$n \in [20, 100]$	3.903	2.933	1.337
	$n \in [500, 1000]$	4.007	3.022	1.327
	$n \in [5000, 10000]$	4.032	3.035	1.328
	All n	3.941	2.965	1.333
Max	All n	5.116	3.928	1.743

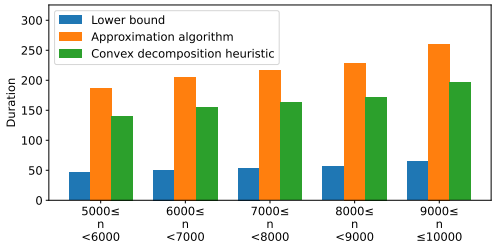
Figure 14 shows typical sweeping schedules of our algorithms; videos of our sweeps can be viewed at [1, 2]. Teal regions are swept earlier, red regions later. In some cases, red regions are isolated from those of similar color, as in the bottom left region of the top left figure, where the segment backtracks to sweep a region near s. Ideally, regions of similar color should be close to each other; however, achieving this requires a better ordering to explore the children of each node (better than the DFS).



(a) Instances of size 20 to 100.

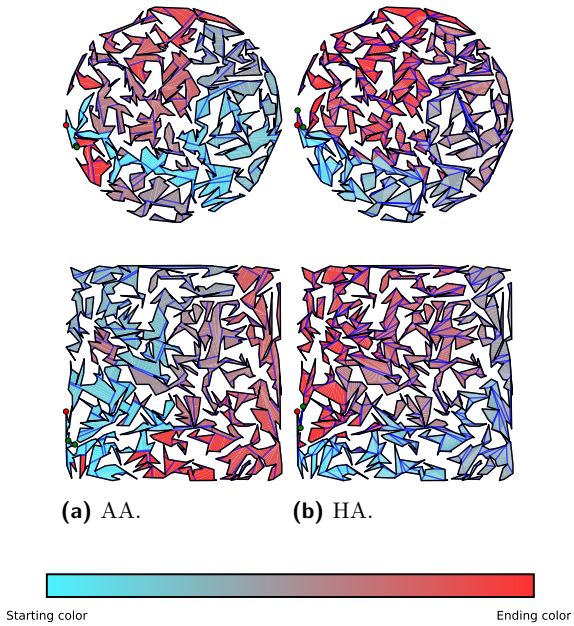


(b) Instances of size 500 to 1000.



(c) Instances of size 5000 to 10000.

Figure 13 The makespans computed using both algorithms in Section 3 for all three sets of randomized polygons. The makespan lower bound for each instance is $\frac{1}{2}|\partial P|$ as mentioned in Subsection 3.1. We grouped the results into bins by the number of vertices of the polygons (n) and calculate the average value for each algorithm as well as the lower bound per group.



(a) AA. **(b)** HA.

Figure 14 Visualization of the sweeping schedules computed by the two algorithms. The top polygon has 774 vertices and the bottom one has 895 vertices. The red dot indicates the starting location of the agents and the green dots indicate where the agents are at the end of the schedule before returning to red. Blue line segments within each polygon show how the polygon was decomposed into subpolygons for the corresponding algorithm. To show the positions of agents over time, we employ a transitioning color scheme in which the segment connecting them is teal at the beginning and red at the end.

Grandchildren-Weight-Balanced Binary Search Trees

Vincent Jugé
LIGM, Univ Gustave Eiffel & CNRS, Marne-la-Vallée, France
IRIF, Université Paris Cité & CNRS, France

Abstract

We revisit weight-balanced trees, also known as trees of bounded balance. Invented by Nievergelt and Reingold in 1972, these trees are obtained by assigning a weight to each node and requesting that the weight of each node should be quite larger than the weights of its children, the precise meaning of "quite larger" depending on a real-valued parameter γ. Blum and Mehlhorn then showed how to maintain them in a recursive (bottom-up) fashion when $2/11 \leqslant \gamma \leqslant 1-1/\sqrt{2}$, their algorithm requiring only an amortised constant number of tree rebalancing operations per update (insertion or deletion). Later, in 1993, Lai and Wood proposed a top-down procedure for updating these trees when $2/11 \leqslant \gamma \leqslant 1/4$.

Our contribution is two-fold. First, we strengthen the requirements of Nievergelt and Reingold, by also requesting that each node should have a substantially larger weight than its grandchildren, thereby obtaining what we call grandchildren-balanced trees. Grandchildren-balanced trees are not harder to maintain than weight-balanced trees, but enjoy a smaller node depth, both in the worst case (with a 6 % decrease) and on average (with a 1.6 % decrease). In particular, unlike standard weight-balanced trees, all grandchildren-balanced trees with n nodes are of height less than $2\log_2(n)$.

Second, we adapt the algorithm of Lai and Wood to all weight-balanced trees, i.e., to all parameter values γ such that $2/11 \leqslant \gamma \leqslant 1-1/\sqrt{2}$. More precisely, we adapt it to all grandchildren-balanced trees for which $1/4 < \gamma \leqslant 1-1/\sqrt{2}$. Finally, we show that, except in limit cases (where, for instance, $\gamma = 1-1/\sqrt{2}$), all these algorithms result in making a constant amortised number of tree rebalancing operations per tree update.

2012 ACM Subject Classification Theory of computation → Sorting and searching

Keywords and phrases Data structures, Balanced binary trees

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.40

Related Version *Full Version*: `https://arxiv.org/abs/2410.08825` [12]

1 Introduction

Among the most fundamental data structures are search trees. Such trees are aimed at representing sets of pairwise comparable elements of size n while allowing basic operations such as membership test, insertion and deletion, which typically require a time linear in the depth of the tree. That is why various data structures such as height-balanced (AVL) trees [1], weight-balanced trees [17], B-trees [4], red-black trees [9], splay trees [19], relaxed k-trees [8] or weak AVL trees [10] were invented since the 1960s. All of them provide worst-case or amortized $\mathcal{O}(\log(n))$ complexities for membership test, insertion and deletion, and may use local modifications of the tree shape called *rotations*.

The most desirable features of such trees include their height, their internal and external path lengths, and the amortised number of rotations triggered by an insertion or deletion. Such statistics are presented in Table 1 below for families of binary trees.

Note that, although relaxed k-trees can be made arbitrarily short (given that each tree should be of height at least $\log_2(n)$), achieving these excellent guarantees on the height of the trees requires performing significantly more rotations, and thus considering other data structures may still be relevant. This is, for instance, why Haeupler, Tarjan and Sen proposed

© Vincent Jugé;
licensed under Creative Commons License CC-BY 4.0

Table 1 Asymptotic approximate worst-case height, internal/external path length and amortised number of rotations per update in binary trees with n nodes. Next to each worst-case bound are indicated references where this bound is proved.

Tree family	Worst-case		
	height	path length	am. rotations/update
AVL	$1.4404\log_2(n)$ [1]	$1.4404n\log_2(n)$ [13]	$\Theta(\log(n))$ [2]
Weight-balanced	$2\log_2(n)$ [17]	$1.1462n\log_2(n)$ [17]	$\Theta(1)$ [5]
Red-black	$2\log_2(n)$ [9]	$2n\log_2(n)$ [7]	$\Theta(1)$ [9]
Splay	n [19]	$n^2/2$ [19]	$\Theta(1)$ [19]
Relaxed k-	$(1+\varepsilon)\log_2(n)$ [8]	$(1+\varepsilon)n\log_2(n)$ [8]	$\mathcal{O}(1/\varepsilon)$ [8]
Weak AVL	$2\log_2(n)$ [9]	$2n\log_2(n)$ [7]	$\Theta(1)$ [10]
Grandchildren-balanced	$1.8798\log_2(n)$	$1.1271n\log_2(n)$	$\Theta(1)$

weak AVL trees in 2009: these are a variant of AVL trees, isomorphic to red-black trees, whose height can be worse than that of AVL trees (but never worse than that of red-black trees), but which require only a constant number of rotations per update, whereas standard AVL trees may require up to $\log(n)$ rotations per update.

In 1972, Nievergelt and Reingold invented *weight-balanced trees*, or *trees of bounded balance* [17]. This family of trees depends on a real parameter γ, and is denoted by $\mathsf{BB}[\gamma]$. It is based on the notion of *weight* of a node n, which is the integer $|n|$ defined as one plus the number of descendants of a node n; alternatively, $|n|$ is the number of empty sub-trees descending from n. Although they might be less efficient than other families of balanced trees in general, they are a reference data structure for order-statistic trees: they are relevant in contexts where we need to store the rank of elements in a dynamic set [6] (i.e., finding efficiently the k^{th} smallest element, or counting elements smaller than a given bound). They are also widely used in libraries of mainstream languages, e.g., Haskell set or map implementations [15, 16].

The family $\mathsf{BB}[\gamma]$ consists of those binary search trees in which, for all nodes u and v, we have $|u| \geqslant \gamma|v|$ whenever u is a child of v. Nievergelt and Reingold also gave an algorithm for dynamically maintaining $\mathsf{BB}[\gamma]$-trees of size n whenever $\gamma \leqslant \sqrt{2}-1$, which required only $\mathcal{O}(\log(n))$ element comparisons and pointer moves per modification.

Blum and Mehlhorn then found that this algorithm worked only when $2/11 \leqslant \gamma \leqslant \sqrt{2}-1$. They also proved that, in that case, it required only $\mathcal{O}(1)$ amortised pointer moves per modification [5]. Lai and Wood subsequently proposed an algorithm for maintaining such trees through a single top-down pass per modification whenever $2/11 \leqslant \gamma \leqslant 1/4$, or whenever updates were guaranteed to be non-redundant [14]. This limitation on the domain of γ when tackling possibly redundant updates is somewhat unfortunate, given that those values of γ for which $\mathsf{BB}[\gamma]$ have the best upper bounds on their height and path length are those for which γ approaches $\sqrt{2}-1$.

Top-down maintenance algorithms have two main advantages over bottom-up algorithms. First, bottom-up algorithms require maintaining a link from each node to its parent or storing the list of nodes we visited along a given branch on a stack, which makes them typically slower than top-down algorithms [3]. Second, they are also a strong bottleneck in concurrent or parallel settings: going down to a leaf and then back to the root must be an atomic operation (or requires maintaining children-to-parent links), since it might finish by changing the root [9, 10].

Thus, our goal here is two-fold. First, by making small changes to a long-known data structure, we improve the guarantees it offers in terms of tree height. Second, we aim at circumventing the limitations of the algorithm from Lai and Wood, by proposing a top-down updating algorithm that will be valid in all cases, and which we also extend to our enhanced data structure.

Contributions

We propose a new variant of weight-bounded trees, which we call *grandchildren-balanced trees*. This family of trees depends on two real parameters α and β, and is denoted by $\mathsf{GCB}[\alpha, \beta]$. It consists of those binary search trees in which, for all nodes u and v, we have $|u| \leqslant \alpha|v|$ whenever u is a child of v, and $|u| \leqslant \beta|v|$ whenever u is a grandchild of v. Grandchildren-balanced trees generalise weight-bounded trees, because $\mathsf{GCB}[\alpha, \alpha^2] = \mathsf{BB}[1 - \alpha]$.

We prove that, for well-chosen values of α and β, the height and internal and external path lengths of grandchildren-balanced trees are smaller than those of weight-bounded trees; in particular, we break the $2\log_2(n)$ lower bound on the worst-case height of weight-balanced trees. These results are achieved by using Algorithm 11.

We also extend the algorithm of Lai and Wood, and propose an algorithm for maintaining grandchildren-balanced trees in a single top-down pass per modification, which requires only $\mathcal{O}(1)$ amortised pointer moves per modification; this is Algorithm 16. Our algorithm is valid for all relevant values of α and β, including those for which $\mathsf{GCB}[\alpha, \beta]$-trees enjoy the best upper bounds on their height and path length.

Most proofs are omitted in the body of this article. Those of Sections 2 and 3 can be found in the appendix; the other ones can be found in the complete version of this article [12].

2 Grandchildren-balanced trees

In this section, we describe a new family of binary search trees, called *grandchildren-balanced trees*, which depends on two real parameters α and β. We also describe the domain in which the pair (α, β) is meaningful and, for such pairs, we provide upper bounds on the height and internal and external path lengths of grandchildren-balanced trees.

Let us first recall the definition given in Section 1.

▶ **Definition 1.** *The* weight *of a binary tree $\mathcal{T}$, denoted by $|\mathcal{T}|$, is defined as the number of empty sub-trees of $\mathcal{T}$; alternatively, $|\mathcal{T}| - 1$ is the number of nodes of $\mathcal{T}$. The* weight *of a node n of $\mathcal{T}$, also denoted by $|n|$, is defined as the weight of the sub-tree of $\mathcal{T}$ rooted at n.*

In what follows, it may be convenient to see each empty tree (of weight 1) as if it had two empty children of weight $1/2$ each, themselves having two empty children of weight $1/4$ each, and so on. In other words, we may see each empty tree as an infinite complete binary tree whose nodes of depth d have weight 2^{-d}.

Let n be a node of $\mathcal{T}$, let n_1 and n_2 be its children, and let n_{11}, n_{12}, n_{21} and n_{22} be its grandchildren. The child-balance *and the* grandchild-balance *(also called C-balance and GC-balance) of n in $\mathcal{T}$ are defined as the real numbers*

$$\mathsf{bal}_{\mathsf{C}}(\mathcal{T}, n) = \frac{\max\{|n_1|, |n_2|\}}{|n|} \text{ and } \mathsf{bal}_{\mathsf{GC}}(\mathcal{T}, n) = \frac{\max\{|n_{11}|, |n_{12}|, |n_{21}|, |n_{22}|\}}{|n|}.$$

When the context is clear, we may omit referring to $\mathcal{T}$, thereby simply writing $\mathsf{bal}_{\mathsf{C}}(n)$ and $\mathsf{bal}_{\mathsf{GC}}(n)$; alternatively, when n is the root of $\mathcal{T}$, its C-balance and the GC-balance may also be directly denoted by $\mathsf{bal}_{\mathsf{C}}(\mathcal{T})$ and $\mathsf{bal}_{\mathsf{GC}}(\mathcal{T})$.

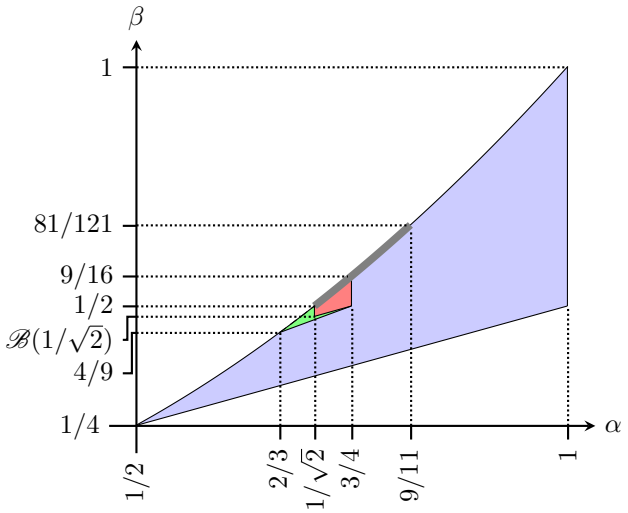


Figure 1 Two-dimensional domains $\mathcal{D}' \subseteq \mathcal{D}'' \subseteq \mathcal{D}$, and one-dimensional (thick, gray) curve $\mathcal{C}$. Whereas $\mathsf{BB}[1-\alpha]$-trees correspond to choosing (α, β) on the curve $\mathcal{C}$, our $\mathsf{GCB}[\alpha, \beta]$-trees can be created on the entire domain $\mathcal{D}'$.

Given real numbers α and β, we say that a node n of $\mathcal{T}$ is α-child balanced *(or α-C-balanced) when $\mathsf{bal}_{\mathsf{C}}(n) \leqslant \alpha$;* β-grandchild-balanced *(or β-GC-balanced) when $\mathsf{bal}_{\mathsf{GC}}(n) \leqslant \beta$; and* (α, β)-balanced *when n is both α-C-balanced and β-GC-balanced.*

Finally, we say that $\mathcal{T}$ is a $\mathsf{GCB}_{x,y}[\alpha, \beta]$-tree when its root is (x, y)-balanced and its other nodes are (α, β)-balanced. For the sake of concision, we simply say that $\mathcal{T}$ is a $\mathsf{GCB}_x[\alpha]$-tree, a $\mathsf{GCB}[\alpha, \beta]$-tree or a $\mathsf{GCB}[\alpha]$-tree when $(y, \beta) = (1, 1)$, $(x, y) = (\alpha, \beta)$ or $(x, y, \beta) = (\alpha, 1, 1)$, respectively.

Below, we will be mostly interested in the family of $\mathsf{GCB}[\alpha, \beta]$-trees. Although it generalises weight-bounded trees invented by Nievergelt and Reingold, we had to shift away from their notation. Indeed, weight-bounded trees were parametrised with a real number γ, by requesting that $|u| \geqslant \gamma|v|$ whenever u is a child of v; however, being β-GC-balanced cannot be expressed by using constraints such as "$|u| \geqslant \gamma|v|$ whenever u is a grandchild of v". In particular, in [5, 17], the *root-balance* of the node n is simply defined as the real number $\rho(n) = |n_1|/|n|$; here, we set $\mathsf{bal}_{\mathsf{C}}(n) = \max\{\rho(n), 1 - \rho(n)\}$.

The family of $\mathsf{GCB}[\alpha, \beta]$-trees coincides with

- both families of $\mathsf{GCB}[\alpha]$-trees and $\mathsf{BB}[1-\alpha]$-trees when $\alpha^2 \leqslant \beta$;
- the empty set when $\alpha < 1/2$;
- the family of $\mathsf{GCB}[1, \beta]$-trees when $1 \leqslant \alpha$;
- the family of $\mathsf{GCB}[2\beta, \beta]$-trees when $\beta < \alpha/2$;
- the family of all binary trees when $\alpha = \beta = 1$.

Thus, we assume below that the pair (α, β) belongs to the domain

$$\mathcal{D} = \{(\alpha, \beta) \colon \alpha/2 \leqslant \beta \leqslant \alpha^2 \text{ and } 1/2 \leqslant \alpha \leqslant 1\} \setminus \{(1, 1)\},$$

represented in Figure 1 along with two other domains $\mathcal{D}'$ and $\mathcal{D}''$ at the end of Section 2.

We first obtain the following upper bound on the height of $\mathsf{GCB}[\alpha, \beta]$-trees.

▶ **Theorem 2.** *When $(\alpha, \beta) \in \mathcal{D}$, each $\mathsf{GCB}[\alpha, \beta]$-tree with $\mathbf{N}$ nodes is of height*

$$h \leqslant -2\log_\beta(\mathbf{N} + 1).$$

Proof. Let $\mathcal{T}$ be a $\mathsf{GCB}[\alpha,\beta]$-tree with $\mathbf{N}$ nodes. Let h be its height, and let n be a node of $\mathcal{T}$ at depth h. An induction on k proves that $|n^{(k)}| \geqslant 2\beta^{-k/2}\alpha$ whenever $0 \leqslant k \leqslant h$, where $n^{(k)}$ denotes the k^{th} ancestor of n, and is defined by n if $k = 0$, or as the parent of $n^{(k-1)}$ if $k \geqslant 1$. It follows that $\mathbf{N}+1 \geqslant |n^{(h)}| \geqslant 2\beta^{-h/2}\alpha \geqslant \beta^{-h/2}$, i.e., that $h \leqslant -2\log_\beta(\mathbf{N}+1)$. ◀

We also focus on the internal and external path lengths of $\mathsf{GCB}[\alpha,\beta]$-trees. The *internal* path length of a tree $\mathcal{T}$ is the sum of the lengths of the paths from the root of $\mathcal{T}$ to the nodes of $\mathcal{T}$; the *external* path length of $\mathcal{T}$ is the sum of the lengths of the paths from the root of $\mathcal{T}$ to the empty sub-trees of $\mathcal{T}$. These lengths are closely related to the average number of queries required to check membership in the set of labels of $\mathcal{T}$.

Indeed, let $\mathcal{T}$ be a binary search tree whose $\mathbf{N}$ nodes are labelled by elements of a linearly ordered set $\mathcal{E}$, and let λ_{int} and λ_{ext} be its internal and external path lengths. The labels of the nodes of $\mathcal{T}$ form a linearly ordered set $\mathcal{S} \subseteq \mathcal{E}$ of size $\mathbf{N}$, which splits $\mathcal{E}$ into $\mathbf{N}+1$ intervals. Then, the average number of queries used to find an element x is:

- $\lambda_{\mathsf{int}}/\mathbf{N}$ when x is chosen uniformly at random in $\mathcal{S}$;
- $\lambda_{\mathsf{ext}}/(\mathbf{N}+1)$ when x is chosen in $\mathcal{E} \setminus \mathcal{S}$, and the interval of $\mathcal{E} \setminus \mathcal{S}$ to which x belongs is chosen uniformly at random.

By construction, each node n increases the lengths λ_{int} and λ_{ext} by $|n|-2$ and $|n|$, respectively. This means that λ_{ext} is the sum of the weights of the tree nodes, and that $\lambda_{\mathsf{int}} = \lambda_{\mathsf{ext}} - 2\mathbf{N}$.

By adapting the proof from [18], we obtain the following upper bound on the external path lengths of $\mathsf{GCB}[\alpha,\beta]$-trees.

▶ **Theorem 3.** *When $(\alpha,\beta) \in \mathcal{D}$, each $\mathsf{GCB}[\alpha,\beta]$-tree with $\mathbf{N}$ nodes is of external path length*

$$\lambda \leqslant (\mathbf{N}+1)\log_2(\mathbf{N}+1)/\Delta,$$

where $\Delta = (\mathsf{H}_2(\alpha) + \alpha\mathsf{H}_2(\beta/\alpha))/(1+\alpha)$, and $\mathsf{H}_2(x) = -x\log_2(x) - (1-x)\log_2(1-x)$ is Shannon's binary entropy.

In what follows, we will investigate more closely the family of $\mathsf{GCB}[\alpha,\beta]$-trees when (α,β) belongs to the more restricted domain

$$\mathcal{D}' = \{(\alpha,\beta)\colon 1/\sqrt{2} \leqslant \alpha < 3/4 \text{ and } \mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2\}, \text{ where } \mathscr{B}(\alpha) = \frac{\sqrt{1+4\alpha}-1}{2}.$$

In the limit cases where $\alpha = 1/\sqrt{2} \approx 0.7071$ and $\beta = \mathscr{B}(\alpha) \approx 0.4783$, Theorems 2 and 3 translate into the inequalities

$$h \leqslant 1.8798\log_2(\mathbf{N}+1) \text{ and } \lambda \leqslant 1.1271(\mathbf{N}+1)\log_2(\mathbf{N}+1),$$

thereby improving the inequalities

$$h \leqslant 2\log_2(\mathbf{N}+1) \text{ and } \lambda \leqslant 1.1462(\mathbf{N}+1)\log_2(\mathbf{N}+1)$$

obtained in the best case (i.e., in the limit case where $\alpha = 1/\sqrt{2}$ and $\beta = 1/2$) for $\mathsf{BB}[1-\alpha]$-trees.

On the domain $\mathcal{D}'$, or even on the larger domain $\mathcal{D}'' = \{(\alpha,\beta)\colon 2/3 \leqslant \beta/\alpha \leqslant \alpha < 3/4\}$, the upper bounds presented in Theorem 2 and 3 are (unsurprisingly) quite tight.

▶ **Proposition 4.** *Let (α,β) be a pair belonging to $\mathcal{D}''$, and let $\mathbf{N} \geqslant 0$ be an integer. There exists a $\mathsf{GCB}[\alpha,\beta]$-tree with $\mathbf{N}$ nodes, of height $h \geqslant -2\log_2(\mathbf{N}+1)/\log_2(\beta) - 7$ and external path length $\lambda \geqslant (\mathbf{N}+1)\log_2(\mathbf{N}+1)/\Delta - 4\mathbf{N}$.*

3 Bottom-up algorithm

In this section, we propose a simple algorithm for rebalancing a $\mathsf{GCB}[\alpha, \beta]$ tree $\mathcal{T}$ after having added a leaf to $\mathcal{T}$ or deleted a node from $\mathcal{T}$, when the pair (α, β) belongs to the domain $\mathcal{D}'$.

Rebalancing $\mathsf{GCB}[\alpha, \beta]$ trees might also be possible when $3/4 \leqslant \alpha \leqslant 1$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$ by using similar ideas, but doing so may require more complicated algorithms, in particular for dealing with base cases, which is not necessarily worth the effort since such parameters provide us with trees of larger height and path length.

In case of a deletion, as illustrated in the three cases of Figure 2, we can safely assume that the node to be deleted is a leaf s, by using Hibbard's deletion technique [11]. Indeed, if the targeted node a has just one child, since a is α-C-balanced and $\alpha < 3/4$, we know that $|a| \leqslant 3$, which means that this child is a leaf; thus, we can just focus on deleting that child and, *a posteriori*, replace the key of a with the key of this just-deleted child. Similarly, if the targeted node a has two children, we can first identify the successor of a as the deepest node b of the left branch stemming from the right child of a; b has no left child, which means that we can just focus on deleting it and, *a posteriori*, replace the key of a with the key of this just-deleted node b.

In both cases, let r be the root of $\mathcal{T}$, and let s be the node that must be inserted or deleted. The weights of the nodes from r to s need to be incremented by 1 in case of a non-redundant insertion (since s just replaced an empty tree, we can pretend that its weight was 1 before it was inserted), or decremented by 1 in case of a non-redundant deletion (since s was a leaf and is replaced by an empty tree, we can pretend that its new weight is 1). However, if the update is redundant, i.e., if we tried to insert a key that was already present in $\mathcal{T}$, or to delete an absent key, these weights will not be changed.

In practice, each tree node will contain two fields for storing explicitly its label and its weight. Moreover, in the description below, we simply say " if s lies in a sub-tree $\mathcal{T}'$ " as a place-holder for " determine, based on the key x that you want to insert or delete and on the key y stored in the root of the current tree, which sub-tree $\mathcal{T}'$ should contain s ".

Our algorithm is based on two algorithmic building blocks, called C-balancing (Algorithm 5) and GC-balancing (Algorithm 6). They consist in locally performing rotations in order to make a given node better balanced when needed, without damaging the balance of its parent and children too much.

In a nutshell, the idea of our bottom-up updating algorithm, which will be made more precise in Algorithm 11, is as follows:

1. We recursively update the sub-tree that needs to be updated.
2. Using the C-balancing algorithm ensures our root and its children are suitably child-balanced.
3. Using the GC-balancing algorithm then ensures our root is suitably child- and grandchild-balanced.

This is the same general idea as the original algorithm from Nievergelt and Reingold [5, 17], adapted to take grandchild balances into account: one should first make our tree child-balanced, and only then can one make it grandchild-balanced too.

One difficulty is that, when we perform a rotation to make our root child- or grandchild-balanced, this rotation should be worth its cost: we expect that, for the next few updates (a quantity that can be made precise, and will be studied in Section 5), no update will be required at all. This is what will make the amortised number of rotations per update a constant.

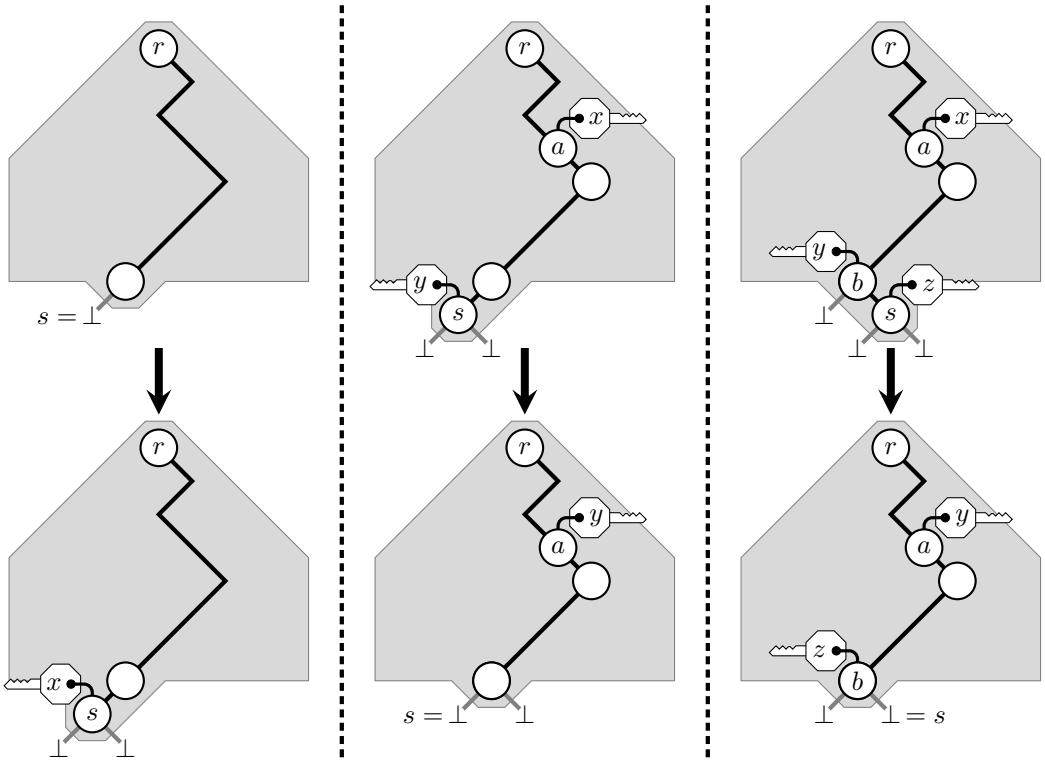


Figure 2 Inserting or deleting a key x: either an empty sub-tree grows into a leaf or a leaf shrinks to an empty sub-tree. When we wish to delete an internal node, we just focus on deleting a leaf descending from that node, and then swap node keys.

In what follows, $\mathcal{T}$ denotes a binary tree that may need to be rebalanced. Its root is denoted by n and, for every node x, the left and right children of x are denoted by x_1 and x_2, respectively. Furthermore, we consider that nodes *move* when rotations are performed. For instance, when either rotation represented in Figure 3 is performed, the node n, which used to be the root of the tree, becomes the right child of the root of the tree resulting from the rotation.

Below, we also say that a node whose children list changed during the rotation was *affected* by the rotation. We will always make sure that, when a simple or double rotation is performed on a tree $\mathcal{T}$, the resulting tree $\mathcal{T}'$ satisfies the inequality $\mathsf{bal}_{\mathsf{C}}(\mathcal{T}') \leqslant \mathsf{bal}_{\mathsf{C}}(\mathcal{T})$. That way, if $\mathcal{T}$ was in fact rooted at some child of a node x, neither the C-balance nor the GC-balance of x, or of any unaffected node, will increase as a result of the rotation.

▶ **Algorithm 5** (C-balancing). *Given real numbers u, v, w and x, the $\mathsf{C}(u,v,w,x)$-balancing algorithm operates as follows on the binary tree $\mathcal{T}$ it receives as input:*

1. *if $|n_1| > u|n| + w$ and $|n_{11}| \geqslant (1-v)|n| - x$, perform the simple rotation of Figure 3;*
2. *if $|n_2| > u|n| + w$ and $|n_{22}| \geqslant (1-v)|n| - x$, perform the mirror image of that rotation;*
3. *if $|n_1| > u|n| + w$ and $|n_{11}| < (1-v)|n| - x$, perform the double rotation of Figure 3;*
4. *if $|n_2| > u|n| + w$ and $|n_{22}| < (1-v)|n| - x$, perform the mirror image of that rotation;*
5. *if $\max\{|n_1|, |n_2|\} \leqslant u|n| + w$, do not modify $\mathcal{T}$.*

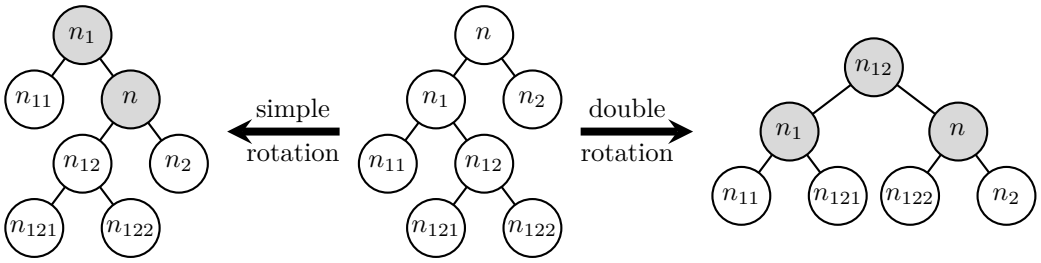


Figure 3 Performing a simple or double rotation. Affected nodes are coloured in grey.

▶ **Algorithm 6** (GC-balancing). *Given real numbers y and z, the $\mathsf{GC}(y,z)$-balancing algorithm operates as follows on the binary tree $\mathcal{T}$ it receives as input:*

1. *if $|n_{11}| > y|n| + z$, perform the simple rotation shown in Figure 3;*
2. *if $|n_{22}| > y|n| + z$, perform the mirror image of that rotation;*
3. *if $|n_{12}| > y|n| + z$, perform the double rotation shown in Figure 3;*
4. *if $|n_{21}| > y|n| + z$, perform the mirror image of that rotation;*
5. *if $\max\{|n_{11}|, |n_{12}|, |n_{21}|, |n_{22}|\} \leqslant y|n| + z$, do not modify $\mathcal{T}$.*

Parameters w, x and z can be seen as terms governing some error margin, that we will set equal to zero in Section 3 and will be non-zero in Section 4. Indeed, in the former case, we develop bottom-up updating algorithms, and thus have a perfect knowledge of $\mathcal{T}$. By contrast, in the latter case, we develop top-down algorithms and thus cannot yet know where the leaf s should be inserted or deleted: depending on the answer, the tree $\mathcal{T}$ may have different shapes, and we cannot anticipate which will be chosen.

The idea of the C-balancing algorithm is to either check that our tree root is u-C-balanced or to transform it into a v-C-balanced node; in the latter case, each affected node will also be made v-C-balanced, which will improve the C-balance of our tree root by at least $u - v$ without damaging the balances of other nodes too much. If we perform a simple rotation, n_{11} will be a new root child, and it will definitely not be too large, but its new sibling, of weight $|n| - |n_{11}|$, might prevent our new root from being v-C-balanced; this undesirable case should occur when $|n_{11}| < (1-v)|n|$, which is why, in that case, we perform a double rotation instead of a simple one. More precisely, here is a result that can be stated about the C-balancing algorithm.

▶ **Lemma 7.** *Let α be a real number such that $1/\sqrt{2} \leqslant \alpha < 3/4$. Then, let $\alpha^\bullet = 19/24$ and*

$$\hat{\alpha} = \frac{1 + \alpha - \sqrt{(1-\alpha)(5-\alpha)}}{2(2\alpha - 1)}.$$

We have $1/\sqrt{2} \leqslant \hat{\alpha} \leqslant \alpha$. Moreover, when $\hat{\alpha} \leqslant u \leqslant \alpha$, the five cases in which the $\mathsf{C}(u, \hat{\alpha}, 0, 0)$-balancing algorithm consists are pairwise incompatible, and those rotations they trigger can always be performed; the algorithm itself transforms each $\mathsf{GCB}_{\alpha^\bullet}[\alpha]$-tree $\mathcal{T}$ whose root is not u-C-balanced into a $\mathsf{GCB}[\alpha]$-tree $\mathcal{T}'$ such that $\mathsf{bal}_\mathsf{C}(\mathcal{T}') \leqslant \mathsf{bal}_\mathsf{C}(\mathcal{T})$ and whose affected nodes are $\hat{\alpha}$-C-balanced.

Proof. Below, we will use the following (in)equalities, which are all easy to check with any computer algebra system (whereas some are quite tedious to check by hand) whenever $1/\sqrt{2} \leqslant \alpha < 3/4$. Except the first two inequalities, each of them is labelled and later reused to prove another inequality with the same label; equality $(2.1 + 2.5)$ is used to prove both subsequent inequalities (2.1) and (2.5).

$$1/\sqrt{2} \leqslant \hat{\alpha}; \qquad \hat{\alpha} \leqslant \alpha;$$
$$\alpha\alpha^\bullet < \hat{\alpha}; \tag{1.1}$$
$$(1-\hat{\alpha})(\alpha^\bullet - 1 + \hat{\alpha}) < \hat{\alpha}(1-\alpha^\bullet); \tag{1.2}$$
$$(1-\hat{\alpha})^2 < \hat{\alpha}^2(1-\alpha); \tag{1.3}$$
$$(1-\hat{\alpha})^2 = \hat{\alpha}(1-\alpha)(2\hat{\alpha}-1); \tag{2.1+2.5}$$
$$(1-\hat{\alpha})\alpha^2 < \hat{\alpha}(1-\alpha); \tag{2.2}$$
$$(1-\hat{\alpha})(1-\alpha) + \alpha\alpha^\bullet < \hat{\alpha}; \tag{2.3}$$
$$(1-\hat{\alpha})\alpha^2\alpha^\bullet < \hat{\alpha}(1-\alpha^\bullet); \tag{2.4}$$
$$1 + (\alpha^2 - 1)\hat{\alpha} < \hat{\alpha}. \tag{2.6}$$

This already proves the inequality $1/\sqrt{2} \leqslant \hat{\alpha} \leqslant \alpha$ of Lemma 7.

Then, let $|m|$ denote the weight of a node m in the tree $\mathcal{T}$, and let $|m|'$ denote its weight in $\mathcal{T}'$. When these weights are equal, we will prefer using the notation $|m|$ even when considering the weight of m in $\mathcal{T}'$. The root of $\mathcal{T}$ is denoted by n, its left and right children are denoted by n_1 and n_2, and so on. Hence, when $u \geqslant \hat{\alpha}$, we have $2u|n| \geqslant \sqrt{2}|n| \geqslant |n_1|+|n_2|$, which makes the inequalities $|n_1| > u|n|$ and $|n_2| > u|n|$ incompatible.

Proving that the desired rotations can indeed be performed amounts to showing that n and n_1 are actual tree nodes (instead of spurious empty nodes of weight $1/2$ or less) in case 1, and that n_{12} is also a tree node in case 3; cases 2 and 4 will be treated symmetrically. In cases 1 and 3, since $|n_1| > u|n| > |n|/2$, the node n is an actual tree node, and cannot be a leaf, which means, as desired, that n and n_1 are tree nodes; furthermore, $|n_1| > u(|n_1|+1) > (|n_1|+1)/\sqrt{2}$, i.e., $|n_1| > 1 + \sqrt{2} > 2$, and therefore $|n_1| \geqslant 3$ and $|n| \geqslant 4$. Then, in case 3, we also have $|n_{12}| = |n_1| - |n_{11}| > u|n| - (1-\hat{\alpha})|n| \geqslant (\sqrt{2}-1)|n| > 1$, which means that n_{12} is also a tree node.

It remains to prove that each affected node is $\hat{\alpha}$-C-balanced: if the root of $\mathcal{T}$ was not u-C-balanced, we will have $\mathsf{bal}_\mathsf{C}(\mathcal{T}') \leqslant \hat{\alpha} \leqslant u \leqslant \mathsf{bal}_\mathsf{C}(\mathcal{T})$.

When $\alpha^\bullet|n| \geqslant |n_1| > u|n| \geqslant \hat{\alpha}|n|$ and $|n_{11}| \geqslant (1-\hat{\alpha})|n|$, a simple rotation is performed, and

$$|n_{11}| \leqslant \alpha|n_1| \leqslant \alpha\alpha^\bullet|n| \leqslant \hat{\alpha}|n| = \hat{\alpha}|n_1|'; \tag{1.1}$$
$$\begin{aligned}(1-\hat{\alpha})|n_{12}| &= (1-\hat{\alpha})(|n_1| - |n_{11}|) \leqslant (1-\hat{\alpha})(\alpha^\bullet|n| - (1-\hat{\alpha})|n|) \\ &\leqslant \hat{\alpha}(1-\alpha^\bullet)|n| \leqslant \hat{\alpha}(|n| - |n_1|) = \hat{\alpha}|n_2|;\end{aligned} \tag{1.2}$$
$$\begin{aligned}(1-\hat{\alpha})|n_2| &= (1-\hat{\alpha})(|n| - |n_1|) \leqslant (1-\hat{\alpha})(1-\hat{\alpha})|n| \\ &\leqslant \hat{\alpha}(1-\alpha)\hat{\alpha}|n| \leqslant \hat{\alpha}(1-\alpha)|n_1| \leqslant \hat{\alpha}(|n_1| - |n_{11}|) = \hat{\alpha}|n_{12}|;\end{aligned} \tag{1.3}$$
$$|n|' = |n| - |n_{11}| \leqslant |n| - (1-\hat{\alpha})|n| = \hat{\alpha}|n| = \hat{\alpha}|n_1|', \tag{1.4}$$

which means precisely that n and n_1 are $\hat{\alpha}$-C-balanced in $\mathcal{T}'$.

Similarly, when $\alpha^\bullet|n| \geqslant |n_1| > u|n| \geqslant \hat{\alpha}|n|$ and $|n_{11}| < (1-\hat{\alpha})|n|$, a double rotation is performed, and

$$\begin{aligned}(1-\hat{\alpha})|n_{11}| &\leqslant (1-\hat{\alpha})^2|n| = \hat{\alpha}(1-\alpha)(\hat{\alpha}|n| - (1-\hat{\alpha})|n|) \\ &\leqslant \hat{\alpha}(1-\alpha)(|n_1| - |n_{11}|) = \hat{\alpha}(1-\alpha)|n_{12}| \\ &\leqslant \hat{\alpha}(|n_{12}| - |n_{122}|) = \hat{\alpha}|n_{121}|;\end{aligned} \tag{2.1}$$
$$\begin{aligned}(1-\hat{\alpha})|n_{121}| &\leqslant (1-\hat{\alpha})\alpha|n_{12}| \leqslant (1-\hat{\alpha})\alpha^2|n_1| \\ &\leqslant \hat{\alpha}(1-\alpha)|n_1| \leqslant \hat{\alpha}(|n_1| - |n_{12}|) = \hat{\alpha}|n_{11}|;\end{aligned} \tag{2.2}$$
$$\begin{aligned}|n_1|' &= |n_{11}| + |n_{121}| \leqslant |n_{11}| + \alpha|n_{12}| = (1-\alpha)|n_{11}| + \alpha|n_1| \\ &\leqslant (1-\alpha)(1-\hat{\alpha})|n| + \alpha\alpha^\bullet|n| \leqslant \hat{\alpha}|n| = \hat{\alpha}|n_{12}|';\end{aligned} \tag{2.3}$$
$$\begin{aligned}(1-\hat{\alpha})|n_{122}| &\leqslant (1-\hat{\alpha})\alpha|n_{12}| \leqslant (1-\hat{\alpha})\alpha^2|n_1| \leqslant (1-\hat{\alpha})\alpha^2\alpha^\bullet|n| \\ &\leqslant \hat{\alpha}(1-\alpha^\bullet)|n| = \hat{\alpha}|n_2|;\end{aligned} \tag{2.4}$$

$$\begin{aligned}(1-\hat{\alpha})|n_2| &= (1-\hat{\alpha})(|n|-|n_1|) \leqslant (1-\hat{\alpha})^2|n| = \hat{\alpha}(1-\alpha)(\hat{\alpha}|n|-(1-\hat{\alpha})|n|)\\ &\leqslant \hat{\alpha}(1-\alpha)(|n_1|-|n_{11}|) = \hat{\alpha}(1-\alpha)|n_{12}|\\ &\leqslant \hat{\alpha}(|n_{12}|-|n_{121}|) = \hat{\alpha}|n_{122}|; \qquad (2.5)\\ |n|' &= |n|-|n_1|+|n_{122}| \leqslant |n|-|n_1|+\alpha|n_{12}|\\ &\leqslant |n|+(\alpha^2-1)|n_1| \leqslant |n|+(\alpha^2-1)\hat{\alpha}|n| \leqslant \hat{\alpha}|n| = \hat{\alpha}|n_{12}|', \qquad (2.6)\end{aligned}$$

which means precisely that n, n_1 and n_{12} are $\hat{\alpha}$-C-balanced in $\mathcal{T}'$.

Finally, the cases where $|n_2| > u|n|$ are symmetrical, and therefore the conclusion of Lemma 7 is valid in these cases too. ◀

Similarly, the GC-balancing algorithm aims at making our tree root y-GC-balanced. If our tree is too unbalanced, we will either promote the responsible grandchild to being a child (if this grandchild was n_{11} or n_{22}) with a simple rotation, or split it in two (if this grandchild was n_{12} or n_{21}) with a double rotation. For adequate values of the parameters u, v and y, each node affected by the GC-balancing algorithm will be $(v,\hat{y})$-balanced for some real number $\hat{y} \leqslant y$ (that does not need to be given as parameter), thus avoiding any damage to the GC-balance of affected nodes, and improving that of our tree root by at least $y-\hat{y}$. Hence, here is a result that can be stated about the GC-balancing algorithm; its proof is similar to that of Lemma 7, and can be found in appendix.

▶ **Lemma 8.** *Let α and β be real numbers such that $1/\sqrt{2} \leqslant \alpha < 3/4$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$. Then, let*

$$\hat{\alpha} = \frac{1-2\alpha+6\alpha^2-\sqrt{(1-\alpha)(5-\alpha)}}{5(2\alpha-1)},\ \hat{\beta} = \frac{1+\alpha-\sqrt{(1-\alpha)^2+4\alpha(\alpha^2-\beta)}}{2(1-\alpha^2+\beta)}\alpha,$$

as well as $\delta = \min\{\hat{\alpha}, \hat{\beta}/\alpha\}$.

We have $1/4 < \hat{\beta} \leqslant \beta$. Moreover, when $\hat{\alpha} \leqslant u \leqslant \alpha$ and $\hat{\beta} \leqslant y \leqslant \beta$, if we apply the $\mathsf{GC}(y,0)$-balancing algorithm on a $\mathsf{GCB}_{\hat{\alpha},1}[\alpha,\beta]$-tree $\mathcal{T}$, cases 1 to 5 are pairwise incompatible, and those rotations they trigger can always be performed; if the root of $\mathcal{T}$ is not y-GC-balanced, the algorithm transforms $\mathcal{T}$ into a $\mathsf{GCB}_{u,y}[\alpha,\beta]$-tree $\mathcal{T}'$ such that $\mathsf{bal}_{\mathsf{C}}(\mathcal{T}') \leqslant \mathsf{bal}_{\mathsf{C}}(\mathcal{T})$ and whose affected nodes are $(\delta,\hat{\beta})$-balanced.

Finally, these building blocks can be combined to ensure that a $\mathsf{GCB}[\alpha,\beta]$-tree whose root is suddenly slightly out of line will be rebalanced.

▶ **Algorithm 9** (CGC-balancing). *Given real numbers u, v, y and $\hat{y}$, the $\mathsf{CGC}(u,v,y,\hat{y})$-balancing algorithm executes the following operations on the tree $\mathcal{T}$ it receives as input:*

1. *the $\mathsf{C}(u,v,0,0)$-balancing algorithm transforms $\mathcal{T}$ into a tree $\mathcal{T}^{(1)}$;*
2. *if $\mathcal{T} \neq \mathcal{T}^{(1)}$,*
   - *the $\mathsf{GC}(\hat{y},0)$-balancing algorithm is applied (in parallel) to the left and right sub-trees of $\mathcal{T}^{(1)}$, which transforms $\mathcal{T}^{(1)}$ into a tree $\mathcal{T}^{(2)}$;*
   - *then, the $\mathsf{GC}(\hat{y},0)$-balancing algorithm is applied to $\mathcal{T}^{(2)}$ itself;*
3. *otherwise, the $\mathsf{GC}(y,0)$-balancing algorithm is applied to $\mathcal{T}$.*

▶ **Proposition 10.** *Let α and β be real numbers such that $1/\sqrt{2} \leqslant \alpha < 3/4$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$; we recall that $\mathscr{B}(\alpha) = (\sqrt{1+4\alpha}-1)/2$. Then, let $\alpha^\bullet = 19/24$,*

$$\hat{\alpha} = \frac{1-2\alpha+6\alpha^2-\sqrt{(1-\alpha)(5-\alpha)}}{5(2\alpha-1)} \text{ and } \hat{\beta} = \frac{1+\alpha-\sqrt{(1-\alpha)^2+4\alpha(\alpha^2-\beta)}}{2(1-\alpha^2+\beta)}\alpha.$$

When $\hat{\alpha} \leqslant u \leqslant \alpha$ and $\hat{\beta} \leqslant y \leqslant \beta$, the $\mathsf{CGC}(u, \hat{\alpha}, y, \hat{\beta})$-balancing algorithm, when applied to a $\mathsf{GCB}_{\alpha\bullet,1}[\alpha, \beta]$-tree $\mathcal{T}$ whose root is not (u, y)-balanced, transforms $\mathcal{T}$ into a $\mathsf{GCB}_{\hat{\alpha},\hat{\beta}}[\alpha, \beta]$-tree $\mathcal{T}'$ such that $\mathsf{bal}_{\mathsf{C}}(\mathcal{T}') \leqslant \mathsf{bal}_{\mathsf{C}}(\mathcal{T})$ and whose affected nodes are all $(\hat{\alpha}, \hat{\beta})$-balanced.

Proof. If the root of $\mathcal{T}$ is (u, y)-balanced, the CGC-balancing algorithm does nothing. If this root is u-C-balanced but not y-GC-balanced, the $\mathsf{GC}(y, 0)$-balancing algorithm is called; Lemma 8 ensures that every affected node will be $(\hat{\alpha}, \hat{\beta})$-balanced and that $\mathsf{bal}_{\mathsf{C}}(\mathcal{T}') \leqslant \mathsf{bal}_{\mathsf{C}}(\mathcal{T})$.

Finally, if the root of $\mathcal{T}$ it is not u-C-balanced, we will call the $\mathsf{C}(u, \hat{\alpha}, 0, 0)$-algorithm, obtaining a $\mathsf{GCB}[\alpha, \beta]$-tree $\mathcal{T}^{(1)}$, whose root will be denoted by r. Nodes affected by this call are r and one or two of its children r_1 and r_2. The $\mathsf{GC}(\hat{\beta}, 0)$-balancing algorithm is then applied to the $\mathsf{GCB}_{\alpha,1}[\alpha, \beta]$-trees rooted at r_1 and r_2; as a result, $\mathcal{T}^{(2)}$ is a $\mathsf{GCB}_{\alpha,1}[\alpha, \beta]$-tree rooted at r, to which the $\mathsf{GC}(\hat{\beta}, 0)$-balancing algorithm is applied once more. No tree to which the C- and GC-balancing algorithms were applied saw its C-balance increase, and thus $\mathsf{bal}_{\mathsf{C}}(\mathcal{T}') \leqslant \mathsf{bal}_{\mathsf{C}}(\mathcal{T})$.

Moreover, each node m affected by the CGC-balancing algorithm is either affected by some call to the GC-balancing algorithm or the root of some tree that the GC-balancing algorithm did not modify; the latter case may only concern nodes r, r_1 and r_2. In both cases, m ends up being $(\hat{\alpha}, \hat{\beta})$-balanced, which completes the proof. ◀

Now is the time when we actually describe our update algorithm, which we will apply to a $\mathsf{GCB}[\alpha, \beta]$-tree in which a leaf s is to be inserted, or from which s is to be deleted. We have two regimes: the first one concerns trees with 11 nodes or less, which we can just restructure in order to make them as balanced as possible, and the second regime concerns trees with 12 nodes or more.

▶ **Algorithm 11** (Bottom-up update). *Let α, $\hat{\alpha}$ and β be real numbers given in Proposition 10. Our updating algorithm executes the following operations on the tree $\mathcal{T}$ it receives as input:*

- *if $\mathcal{T}$ contains 11 nodes or less, insert or delete s (if possible) and make the resulting tree a perfectly balanced tree, i.e., a tree in which the weights of two siblings differ by at most 1;*
- *if $\mathcal{T}$ contains 12 nodes or more,*
  1. *recursively rebalance the (left or right) sub-tree in which s lies, and then*
  2. *if the update turned out to be non-redundant, update the weight of the root of $\mathcal{T}$, and then apply the $\mathsf{CGC}(\alpha, \hat{\alpha}, \beta, \hat{\beta})$-balancing algorithm to $\mathcal{T}$.*

Due to the recursive flavour of this algorithm, the rebalancing operations are performed bottom-up. We will see in Section 4 how to perform these operations top-down.

▶ **Theorem 12.** *Let α and β be real numbers such that $1/\sqrt{2} \leqslant \alpha < 3/4$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$. The tree obtained by using Algorithm 11 to insert a leaf in a $\mathsf{GCB}[\alpha, \beta]$-tree $\mathcal{T}$ or delete a node from $\mathcal{T}$ is also a $\mathsf{GCB}[\alpha, \beta]$-tree.*

Proof. Our proof is a variant of the proof of [5].

Let $\mathcal{T}'$ and $\mathcal{T}''$ be the trees obtained just after step 1 and step 2 of Algorithm 11, respectively. We will prove by induction on $|\mathcal{T}|$ that $\mathcal{T}''$ is a $\mathsf{GCB}[\alpha, \beta]$-tree. The result being correct by construction when $|\mathcal{T}| \leqslant 12$, we assume that $|\mathcal{T}| \geqslant 13$.

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the left and right sub-trees of $\mathcal{T}$, and let $t = |\mathcal{T}|$, $t_1 = |\mathcal{T}_1|$ and $t_2 = |\mathcal{T}_2|$. Similarly, let $\mathcal{T}'_1$ and $\mathcal{T}'_2$ be the left and right sub-trees of $\mathcal{T}'$, and let $t' = |\mathcal{T}'|$ $t'_1 = |\mathcal{T}'_1|$ and $t'_2 = |\mathcal{T}'_2|$. Without loss of generality, we assume that the update was non-redundant, and that the tree $\mathcal{T}$ was altered by adding s to $\mathcal{T}_1$ or by deleting s from $\mathcal{T}_2$. This means that either $(t', t'_1, t'_2) = (t + 1, t_1 + 1, t_2)$ or $(t'_1, t'_2) = (t - 1, t_1, t_2 - 1)$. In both cases, the induction hypothesis ensures that $\mathcal{T}'_1$ and $\mathcal{T}'_2$ are $\mathsf{GCB}[\alpha, \beta]$-trees.

We prove now that the root of $\mathcal{T}'$ is $\alpha^\bullet$-C-balanced, thereby allowing us to use Proposition 10 and completing the induction. Indeed, let $t = t_1 + t_2$ and $t' = t'_1 + t'_2$. Since $4t_1 \leqslant 4\alpha t < 3t$, we know that $4t_1 \leqslant 3t - 1$.

Thus, in case of an insertion, $t'_1 = t_1 + 1 \leqslant (3t + 3)/4 = 3t'/4 \leqslant \alpha^\bullet t'$ and $t'_2 = t_2 \leqslant \alpha t \leqslant \alpha^\bullet t'$.

Similarly, in case of a deletion, $t'_1 = t_1 \leqslant (3t-1)/4 = (3t'+2)/4 = \alpha^\bullet t' - (t'-12)/24 \leqslant \alpha^\bullet t'$, whereas $t'_2 = t_2 - 1 \leqslant \alpha t - 1 \leqslant \alpha t' \leqslant \alpha^\bullet t'$. ◀

4 Top-down algorithm

In this section, we propose a top-down algorithm for inserting an element into (or deleting an element from) a weight-balanced tree. This algorithm is valid whenever $1/\sqrt{2} \leqslant \alpha < 3/4$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$, thereby completing the algorithm of Lai and Wood [14], which works only when $3/4 \leqslant \alpha \leqslant 9/11$ and $\beta = \alpha^2$. This new algorithm is inspired by theirs: we wish to perform a top-down restructuring pass while adjusting weight information. If the update is redundant, a second top-down pass will be needed to update this information, but no further restructuring will be needed.

More precisely, we aim at having a top-down algorithm that requires considering only a constant number of tree nodes at each step; this number may not depend on the parameters such as α and β. Moreover, we still wish, by using this algorithm, to perform only a constant number of rotations per update; this number *may* depend on α, β and other parameters. Consequently, the idea of the algorithm, which will be made more precise in Algorithm 16, is as follows:

1. If the tree is large enough, we may rebalance it to make sure it will remain balanced even if we recursively update one of its children. Like in Section 3, rotations performed in this phase should be so efficient that only an amortised constant number of rotations per update will be useful.
2. If the tree is small enough, we make it as balanced as possible, the notion of being top-down being void in this case.

A key object towards defining and proving the correctness of our algorithm is the notion of *robust* grandchildren-balanced trees.

▶ **Definition 13.** *Let n be a node of a binary tree $\mathcal{T}$, let n_1 and n_2 be its children, and let n_{11}, n_{12}, n_{21} and n_{22} be its grandchildren. Given real numbers α and β, we say that n is* robustly α-C-balanced *when*

$$\max\{|n_1|, |n_2|\} + 1 \leqslant \alpha(|n| + 1) \text{ and } \max\{|n_1|, |n_2|\} \leqslant \alpha(|n| - 1);$$

that n is robustly β-GC-balanced *when*

$$\max\{|n_{11}|, |n_{12}|, |n_{21}|, |n_{22}|\}+1 \leqslant \beta(|n|+1) \text{ and } \max\{|n_{11}|, |n_{12}|, |n_{21}|, |n_{22}|\} \leqslant \beta(|n|-1);$$

and that n is robustly (α, β)-balanced *when n is robustly α-C-balanced and robustly β-GC-balanced.*

Finally, we say that a tree $\mathcal{T}$ is a $\mathsf{RGCB}_{x,y}[\alpha, \beta]$-tree *when its root is robustly (x, y)-balanced and its other nodes are (α, β)-balanced. For the sake of concision, we simply say that $\mathcal{T}$ is a* $\mathsf{RGCB}[\alpha, \beta]$-tree *or a* $\mathsf{RGCB}[\alpha]$-tree *when $(x, y) = (\alpha, \beta)$ or $(x, y, \beta) = (\alpha, 1, 1)$, respectively.*

Inequalities $|n_i|+1 \leqslant \alpha(|n|+1)$ and $|n_i| \leqslant \alpha(|n|-1)$ ensure that, even if a leaf is inserted into or deleted from $\mathcal{T}$, the node n will remain α-C-balanced. Inequalities $|n_{ij}|+1 \leqslant \beta(|n|+1)$ and $|n_{ij}| \leqslant \beta(|n|-1)$ serve the same purpose, but for ensuring that n remains β-GC-balanced. Finally, each tree node is robustly $(1,1)$-balanced.

Setting $\mathsf{R}(t) = \min\{-t, t-1\}$ and observing that $\min\{t(|n|+1)-1, t(|n|-1)\} = t|n|+\mathsf{R}(t)$ for all real numbers t provides us with a more succinct characterisation of (α,β)-balanced nodes, which will require giving non-zero values to the parameters w, x and z: the node n is robustly (α,β)-balanced if and only if $\max\{|n_1|, |n_2|\} \leqslant \alpha|n| + R(\alpha)$ and $\max\{|n_{11}|, |n_{12}|, |n_{21}|, |n_{22}|\} \leqslant \beta|n| + \mathsf{R}(\beta)$.

We can now group several calls to the C- and GC-balancing algorithms into a so-called RCGC-balancing algorithm, whose aim is to make a tree root robustly balanced.

▶ **Algorithm 14** (RCGC-balancing). *Given real numbers u, v, y and $\hat{y}$, the $\mathsf{RCGC}(u,v,y,\hat{y})$-balancing algorithm executes the following operations, in this order, on the tree $\mathcal{T}$ it receives as input, thus transforming it into a tree $\mathcal{T}'$:*

1. *the $\mathsf{C}(u,v,\mathsf{R}(u),\mathsf{R}(v))$-balancing algorithm transforms $\mathcal{T}$ into a tree $\mathcal{T}^{(1)}$;*
2. *if $\mathcal{T} \neq \mathcal{T}^{(1)}$,*
   - *the $\mathsf{CGC}(v,v,\hat{y},\hat{y})$-balancing algorithm is applied (in parallel) to the left and right sub-trees of $\mathcal{T}^{(1)}$, which transforms $\mathcal{T}^{(1)}$ into a tree $\mathcal{T}^{(2)}$;*
   - *the $\mathsf{GC}(\hat{y},\mathsf{R}(\hat{y}))$-balancing algorithm transforms $\mathcal{T}^{(2)}$ into a tree $\mathcal{T}^{(3)}$;*
   - *the $\mathsf{CGC}(v,v,\hat{y},\hat{y})$-balancing algorithm is applied (in parallel) to the left and right sub-trees of $\mathcal{T}^{(3)}$, which yields the tree $\mathcal{T}'$;*
3. *otherwise, the $\mathsf{GC}(y,\mathsf{R}(y))$-balancing algorithm transforms $\mathcal{T}$ into a tree $\mathcal{T}^{(3)}$, and then,*
   - *if $\mathcal{T} \neq \mathcal{T}^{(3)}$, the $\mathsf{CGC}(v,v,\hat{y},\hat{y})$-balancing algorithm is applied (in parallel) to the left and right sub-trees of $\mathcal{T}^{(3)}$, which yields the tree $\mathcal{T}'$;*
   - *otherwise, $\mathcal{T}' = \mathcal{T}$.*

The idea is similar to that of the plain C- and GC-balancing algorithm, but we wish to transform a tree whose root is not robustly (u,y)-balanced into a tree $\mathcal{T}'$ whose root will be robustly $(v,\hat{y})$-balanced. Calling the $\mathsf{C}(u,v,\mathsf{R}(u),\mathsf{R}(v))$-, $\mathsf{GC}(\hat{y},\mathsf{R}(\hat{y}))$- and $\mathsf{GC}(y,\mathsf{R}(y))$-balancing algorithms achieves this goal, while possibly damaging the balance of the children of the root of $\mathcal{T}'$; we solve this issue by the CGC-balancing algorithm to the sub-trees rooted at these children.

▶ **Proposition 15.** *Let α, $\hat{\alpha}$, β and $\hat{\beta}$ be real numbers given in Proposition 10. The $\mathsf{RCGC}(\alpha,\hat{\alpha},\beta,\hat{\beta})$-balancing algorithm transforms each $\mathsf{GCB}_{\alpha,1}[\alpha,\beta]$-tree $\mathcal{T}$ with 30 nodes or more and whose root is not robustly (α,β)-balanced into a $\mathsf{RGCB}[\alpha,\beta]$-tree $\mathcal{T}'$ such that $\mathsf{bal}_{\mathsf{C}}(\mathcal{T}') \leqslant \mathsf{bal}_{\mathsf{C}}(\mathcal{T})$ and whose affected nodes are all $(\hat{\alpha},\hat{\beta})$-balanced.*

We finally describe our top-down updating algorithm, which we will apply to a $\mathsf{GCB}[\alpha,\beta]$-tree in which a leaf s is to be inserted, or from which s is to be deleted. Like the algorithm of Lai and Wood [14], this algorithm is not purely top-down, because (i) if we wanted to delete an internal node a, as illustrated in Figure 2, it requires maintaining a pointer to a, whose key will later be replaced by another node key, and (ii) if the update turned out to be redundant, a second top-down pass will be required to cancel every weight update we performed. Alternative representations of weight-balanced trees or $\mathsf{GCB}[\alpha,\beta]$-trees allow omitting this second pass: instead of storing the weight $|n|$ at each node n, we should store one of the weights $|n_1|$ or $|n_2|$ as well as a bit indicating which weight we stored.

▶ **Algorithm 16** (Top-down update). *Let α, $\hat{\alpha}$, β and $\hat{\beta}$ be real numbers given in Proposition 10. Our updating algorithm executes the following operations on the tree $\mathcal{T}$ it receives as input:*

- *if $\mathcal{T}$ contains 29 nodes or less, insert or delete s (if possible) and make the resulting tree a perfectly balanced tree, i.e., a tree in which the weights of two siblings differ by at most 1;*
- *if $\mathcal{T}$ contains 30 nodes or more,*
  1. *apply the $\mathsf{RCGC}(\alpha, \hat{\alpha}, \beta, \hat{\beta})$-balancing algorithm to $\mathcal{T}$, then*
  2. *update the weight of the root of $\mathcal{T}$, and finally*
  3. *recursively update the (left or right) sub-tree of $\mathcal{T}$ that contains s.*

▶ **Theorem 17.** *Let α and β be real numbers such that $1/\sqrt{2} \leqslant \alpha < 3/4$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$. The tree obtained by using Algorithm 16 to insert a leaf in a $\mathsf{GCB}[\alpha, \beta]$-tree $\mathcal{T}$ or delete a node from $\mathcal{T}$ is also a $\mathsf{GCB}[\alpha, \beta]$-tree.*

5 Complexity analysis

In this final section, we prove that Algorithms 11 and 16 proposed above perform an amortised constant number of rotations per attempted insertion or deletion.

The idea consists in evaluating the sums of child- and grandchild-balances of all tree nodes. Indeed, inserting or deleting a leaf s will slightly damage the child- and grandchild balances of the ancestors of s, which may increase the sum of these balances by no more than a constant. In the opposite direction, whenever the CGC- or RCGC-balancing algorithm changes a sub-tree $\mathcal{T}$, some node will see its child-balance decrease by approximately $\alpha - \hat{\alpha}$, or its grandchild-balance decrease by approximately $\beta - \hat{\beta}$; other node balances might be damaged in the process, but not to the point of exceeding $(\hat{\alpha}, \hat{\beta})$. Thus, not too many changes may be performed.

These ideas lead to the following result.

▶ **Theorem 18.** *Let α and β be real numbers such that $1/\sqrt{2} < \alpha < 3/4$ and $\mathscr{B}(\alpha) < \beta \leqslant \alpha^2$. Let η and ε be given by $\eta = \alpha - 1/\sqrt{2}$ and $\varepsilon = \min\{\beta - \mathscr{B}(\alpha), \alpha^2 - \beta\}$ if $\mathscr{B}(\alpha) < \beta < \alpha^2$, or $\varepsilon = 1$ if $\beta = \alpha^2$.*

Then, let $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_k$ be $\mathsf{GCB}[\alpha, \beta]$-trees defined as follows: $\mathcal{T}_0$ is the empty tree and, for all $\ell \geqslant 0$, the tree $\mathcal{T}_{\ell+1}$ is obtained by inserting a leaf in $\mathcal{T}_\ell$ or deleting a leaf from $\mathcal{T}_\ell$, using either Algorithm 11 or 16 to do so. Gradually transforming $\mathcal{T}_0$ into $\mathcal{T}_k$ via such steps requires only $\mathcal{O}(k/\eta + k/\varepsilon)$ rotations.

Proof outline. Each node n is given a real-valued counter $\mathsf{c}(n)$ that receives the value 0 when n is created (i.e., inserted in a tree) or affected by a rotation, and is increased by $2/|n|$ when a descendant of n is about to be created or deleted.

Finally, let C be the sum of all these counters $\mathsf{c}(n)$, and let $\delta = \min\{\alpha - \hat{\alpha}, \beta - \hat{\beta}, 1/62\}/2$. One can prove that the sum C increases by no more than 16 when a leaf is inserted or deleted, and decreases by at least δ when the CGC- or RCGC-balancing algorithms trigger a rotation and the tree contains at least 32 nodes; if the tree contains fewer than 32 nodes, rotations do not make C increase anyway.

Since the sum C is initially zero, and terminates with a non-negative value, no more than $\mathcal{O}(k/\delta)$ rotations are performed on trees with 32 nodes or more, and no more than $\mathcal{O}(k)$ rotations can be performed on trees with fewer than 32 nodes. ◀

In particular, let us focus on some approach to the critical point $(\alpha_\mathsf{c}, \beta_\mathsf{c}) = (1/\sqrt{2}, \mathscr{B}(1/\sqrt{2}))$. When setting $\alpha = \alpha_\mathsf{c} + x$ and $\beta = \beta_\mathsf{c} + x$, and provided that $0 < x < 1/10$, the inequalities $1/\sqrt{2} < \alpha < 3/4$ and $\mathscr{B}(\alpha) < \beta < \alpha^2$ are valid. Moreover,

- Theorem 2 states that $\mathsf{GCB}[\alpha, \beta]$-trees with $\mathbf{N}$ nodes are of height

$$h \leqslant -2\log_2(\mathbf{N}+1)/\log_2(\beta_\mathsf{c}) + 7\log_2(\mathbf{N}+1)x;$$

- Theorem 3 states that $\mathsf{GCB}[\alpha, \beta]$-trees with $\mathbf{N}$ nodes are of external path length

$$\lambda \leqslant (\mathbf{N}+1)\log_2(\mathbf{N}+1)/\Delta_\mathsf{c} + 2(\mathbf{N}+1)\log_2(\mathbf{N}+1)x,$$

  where $\Delta_\mathsf{c} = (\mathsf{H}_2(\alpha_\mathsf{c}) + \alpha_\mathsf{c}\mathsf{H}_2(\beta_\mathsf{c}/\alpha_\mathsf{c}))/(1+\alpha_\mathsf{c})$;
- Theorem 18 states that inserting or deleting k leaves in $\mathsf{GCB}[\alpha, \beta]$-trees requires $\mathcal{O}(k/x)$ rotations. More precisely, digging in the constants hidden in the $\mathcal{O}$ notation yields a (crude) upper bound of less than $30(1+28/x)k$ rotations.

References

1. Georgii Maksimovich Adel'son-Velskii and Evgenii Mikhailovich Landis. An algorithm for organization of information. *Doklady Akademii Nauk*, 146(2):263–266, 1962.
2. Mahdi Amani, Kevin Lai, and Robert Tarjan. Amortized rotation cost in AVL trees. *Information Processing Letters*, 116(5):327–330, 2016. doi:10.1016/j.ipl.2015.12.009.
3. Lukas Barth and Dorothea Wagner. Engineering top-down weight-balanced trees. In *22nd Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 161–174. SIAM, 2020. doi:10.1137/1.9781611976007.13.
4. Rudolf Bayer and Edward McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972. doi:10.1007/BF00288683.
5. Norbert Blum and Kurt Mehlhorn. On the average number of rebalancing operations in weight-balanced trees. *Theoretical Computer Science*, 11:303–320, 1980. doi:10.1016/0304-3975(80)90018-3.
6. Gerth Stølting Brodal and Allan Grønlund Jørgensen. Data structures for range median queries. In *International Symposium on Algorithms and Computation*, pages 822–831. Springer, 2009. doi:10.1007/978-3-642-10631-6_83.
7. Helen Cameron and Derick Wood. A note on the path length of red-black trees. *Information Processing Letters*, 42(5):287–292, 1992. doi:10.1016/0020-0190(92)90038-W.
8. Rolf Fagerberg, Rune Jensen, and Kim Larsen. Search trees with relaxed balance and near-optimal height. In *7th International Workshop on Algorithms and Data Structures (WADS)*, volume 2125 of *Lecture Notes in Computer Science*, pages 414–425. Springer, 2001. doi:10.1007/3-540-44634-6_38.
9. Leonidas Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 8–21. IEEE Computer Society, 1978. doi:10.1109/SFCS.1978.3.
10. Bernhard Haeupler, Siddhartha Sen, and Robert Tarjan. Rank-balanced trees. *ACM Transactions on Algorithms (TALG)*, 11(4):1–26, 2015. doi:10.1145/2689412.
11. Thomas Hibbard. Some combinatorial properties of certain trees with applications to searching and sorting. *Journal of the ACM*, 9(1):13–28, 1962. doi:10.1145/321105.321108.
12. Vincent Jugé. Grand-children weight-balanced binary search trees, 2024. doi:10.48550/arXiv.2410.08825.
13. Rolf Klein and Derick Wood. A tight upper bound for the path length of AVL trees. *Theoretical Computer Science*, 72(2&3):251–264, 1990. doi:10.1016/0304-3975(90)90037-I.
14. Tony Lai and Derick Wood. A top-down updating algorithm for weight-balanced trees. *International Journal of Foundations of Computer Science*, 4(4):309–324, 1993. doi:10.1142/S0129054193000201.
15. Daan Leijen. Set implementation in Haskell 2010. hackage.haskell.org/package/containers-0.7/docs/Data-Set.html, 2002.

16 Daan Leijen and Andriy Palamarchuk. Map implementation in Haskell 2010. `hackage.haskell.org/package/containers-0.7/docs/Data-Map.html`, 2008.

17 Jürg Nievergelt and Edward Reingold. Binary search trees of bounded balance. In 4^{th} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 137–142. ACM, 1972. `doi:10.1145/800152.804906`.

18 Jürg Nievergelt and C. K. Wong. Upper bounds for the total path length of binary trees. *Journal of the ACM*, 20(1):1–6, 1973. `doi:10.1145/321738.321739`.

19 Daniel Sleator and Robert Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985. `doi:10.1145/3828.3835`.

A Missing proofs of Sections 2 and 3

▶ **Theorem 3.** *When $(\alpha, \beta) \in \mathcal{D}$, each $\mathsf{GCB}[\alpha, \beta]$-tree with $\mathbf{N}$ nodes is of external path length*

$$\lambda \leqslant (\mathbf{N}+1)\log_2(\mathbf{N}+1)/\Delta,$$

where $\Delta = (\mathsf{H}_2(\alpha) + \alpha\mathsf{H}_2(\beta/\alpha))/(1+\alpha)$, and $\mathsf{H}_2(x) = -x\log_2(x) - (1-x)\log_2(1-x)$ is Shannon's binary entropy.

Proof. Let $\lambda(\mathcal{T})$ be the sum of the weights of the nodes of a tree $\mathcal{T}$: we prove by induction on $|\mathcal{T}|$ that $\lambda(\mathcal{T}) \leqslant |\mathcal{T}|\log_2(|\mathcal{T}|)/\Delta$.

When $|\mathcal{T}| = 1$, this inequality rewrites as $0 \leqslant 0$. When $|\mathcal{T}| = 2$, it rewrites as $\Delta \leqslant 1$, which follows from the fact that $\mathsf{H}_2(x) \leqslant 1$ whenever $x \in [0,1]$. Therefore, we assume that $|\mathcal{T}| \geqslant 3$.

Let n be the root of $\mathcal{T}$, and let $\mathcal{T}_1$ and $\mathcal{T}_2$ be its left and right sub-trees. Without loss of generality, we assume that $|\mathcal{T}_1| \geqslant |\mathcal{T}_2|$. Thus, $|\mathcal{T}_1| \geqslant |\mathcal{T}|/2 > 1$, and $\mathcal{T}_1$ is non-empty. Let n_1 be its root, and let $\mathcal{T}_{11}$ and $\mathcal{T}_{12}$ be the left and right sub-trees of n_1. Once again, without loss of generality, we assume that $|\mathcal{T}_{11}| \geqslant |T_{12}|$.

Let $t = |\mathcal{T}|$, $x = |\mathcal{T}_1|/|\mathcal{T}|$ and $y = |\mathcal{T}_{11}|/|\mathcal{T}_1|$. The induction hypothesis states now that

$$\begin{aligned}\Delta\lambda(\mathcal{T})/t &= \Delta(x+1) + \Delta\lambda(\mathcal{T}_{11})/t + \Delta\lambda(\mathcal{T}_{12})/t + \Delta\lambda(\mathcal{T}_2)/t\\ &\leqslant \Delta(x+1) + xy\log_2(txy) + x(1-y)\log_2(tx(1-y)) + (1-x)\log_2(t(1-x))\\ &\leqslant \Delta(x+1) + \log_2(t) - x\mathsf{H}_2(y) - \mathsf{H}_2(x),\end{aligned}$$

and it remains to prove that the quantity $\mathsf{F}_{\alpha,\beta}(x,y) = x\mathsf{H}_2(y) + \mathsf{H}_2(x) - \Delta(x+1)$ is non-negative.

First, let $\gamma = \beta/\alpha$. If $\gamma \leqslant x \leqslant \alpha$, since H_2 is decreasing on the interval $[1/2, 1]$ and $1/2 \leqslant y \leqslant \beta/x$, we know that $\mathsf{F}_{\alpha,\beta}(x,y) \geqslant \mathsf{F}_{\alpha,\beta}(x,\beta/x)$. Moreover, the function $\mathsf{G}\colon x \mapsto \mathsf{F}_{\alpha,\beta}(x,\beta/x)$ is concave on $[\gamma,\alpha]$, because its second derivative is

$$\mathsf{G}''(x) = -\frac{1-\beta}{(x-\beta)(1-x)\ln(2)} < 0$$

whenever $\beta \leqslant \gamma < x < \alpha \leqslant 1$. Observing that $\mathsf{G}(\alpha) = \mathsf{F}_{\alpha,\beta}(\alpha,\gamma) = 0$ and that

$$\begin{aligned}\mathsf{G}(\gamma) = \mathsf{F}_{\alpha,\beta}(\gamma,\alpha) &= (\beta\mathsf{H}_2(\alpha) + \mathsf{H}_2(\gamma)) - (\gamma+1)(\mathsf{H}_2(\alpha) + \alpha\mathsf{H}_2(\gamma))/(\alpha+1)\\ &= (1-\beta)(\mathsf{H}_2(\gamma) - \mathsf{H}_2(\alpha))/(\alpha+1) \geqslant 0,\end{aligned}$$

we conclude that $\mathsf{F}_{\alpha,\beta}(x,y) \geqslant \mathsf{G}(x) \geqslant \min\{\mathsf{G}(\alpha), \mathsf{G}(\gamma)\} \geqslant 0$ whenever $\gamma \leqslant x \leqslant \alpha$. Finally, if $1/2 \leqslant x \leqslant \gamma$, and since $1/2 \leqslant y \leqslant \alpha$, we have $\mathsf{F}_{\alpha,\beta}(x,y) \geqslant \mathsf{F}_{\alpha,x}(x,y) \geqslant \mathsf{F}_{\alpha,x}(x,\alpha) \geqslant 0$. ◀

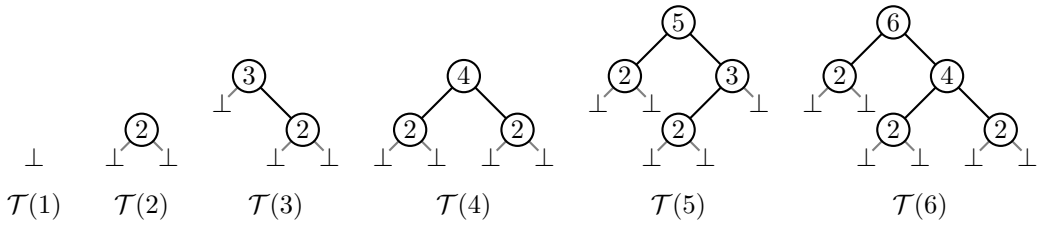


Figure 4 Trees $\mathcal{T}(1)$ to $\mathcal{T}(6)$. Nodes are labelled by their weight; empty trees are denoted by $\perp$.

▶ **Proposition 4.** *Let (α, β) be a pair belonging to $\mathcal{D}''$, and let $\mathbf{N} \geqslant 0$ be an integer. There exists a $\mathsf{GCB}[\alpha, \beta]$-tree with $\mathbf{N}$ nodes, of height $h \geqslant -2\log_2(\mathbf{N}+1)/\log_2(\beta) - 7$ and external path length $\lambda \geqslant (\mathbf{N}+1)\log_2(\mathbf{N}+1)/\Delta - 4\mathbf{N}$.*

Proof. Let $\gamma = \beta/\alpha$. For each integer $s \geqslant 1$, we define inductively a tree $\mathcal{T}(s)$ of weight s as follows:

- if $s \leqslant 2$, $\mathcal{T}(s)$ is the only tree of weight s;
- if $s \geqslant 3$, $\mathcal{T}(s)$ is the tree whose left child is $\mathcal{T}(s_1)$ and whose right child's children are $\mathcal{T}(s_{21})$ and $\mathcal{T}(s_{22})$, where $s_1 = s - \lfloor \alpha s \rfloor$, $s_{21} = \lfloor \gamma \lfloor \alpha s \rfloor \rfloor$ and $s_{22} = s - s_1 - s_{21}$.

We present in Figure 4 the trees $\mathcal{T}(s)$ obtained when $1 \leqslant s \leqslant 6$. Since $2/3 < \gamma \leqslant \alpha < 3/4$, these trees do not depend on the values of α and γ (or β).

We first prove by induction on s that $\mathcal{T}(s)$ is a $\mathsf{GCB}[\alpha, \beta]$-tree. This is visibly true when $s \leqslant 4$, hence we assume that $s \geqslant 5$. Let $s_2 = \lfloor \alpha s \rfloor$ be the weight of the right child of $\mathcal{T}(s)$: it suffices to prove that $s_1 \leqslant \gamma s$, $s_2 \leqslant \alpha s$, $s_{21} \leqslant \gamma s_2$ and $s_{22} \leqslant \gamma s_2$. The inequalities $s_2 \leqslant \alpha s$ and $s_{21} \leqslant \gamma s_2$ are immediate, and $s_2 \geqslant \lfloor 2s/3 \rfloor \geqslant 3$, thereby proving that:

- $s_1 \leqslant (1 - \alpha + 1/s)s \leqslant (1 - 2/3 + 1/5)s < 2s/3 < \gamma s$;
- $s_{22} \leqslant (1 - \gamma + 1/s_2)s_2 \leqslant (1 - 2/3 + 1/3)s_2 = 2s_2/3 < \gamma s_2$.

Now, let $h(s)$ be the height of $\mathcal{T}(s)$ and let $\lambda(s)$ be the sum of the weights of the nodes of $\mathcal{T}(s)$. We will prove inductively that $h(s) \geqslant h^+(s+\kappa)$ and $\lambda(s) \geqslant \lambda^+(s+\kappa) + v$ for all $s \geqslant 1$, where we set $h^+(x) = -2\log_\beta(x) - 7$, $\lambda^+(x) = x\log_2(x)/\Delta - 4x$, $\kappa = (\gamma+1)/(1-\beta)$ and $v = (1 + (\alpha+1)\kappa)/2$.

First, we observe, when $(\alpha, \beta) \in \mathcal{D}'$, that $4/5 \leqslant \mathsf{H}_2(3/4) \leqslant \Delta \leqslant \mathsf{H}_2(2/3) \leqslant \log_2(5e)/4$, $3 \leqslant \kappa \leqslant 4$ and $3 \leqslant v \leqslant 4$. We will use these inequalities several times below.

For example, $\kappa + 1 \leqslant 5 \leqslant (4/3)^6 \leqslant \beta^3$ and $\kappa + 2 \leqslant 6 \leqslant (4/3)^7 \leqslant \beta^{7/2}$, so that $h(s) = s - 2 \geqslant h^+(s+\kappa)$ when $s = 1$ or $s = 2$.

Then, if $s \geqslant 3$, observing that $s_{21} + \kappa \geqslant \beta s - \gamma - 1 = \beta(s+\kappa)$ and using the induction hypothesis proves that $h(s) \geqslant h(s_{21}) + 2 \geqslant h^+(s_{21}+\kappa) + 2 \geqslant 2 - 2\log_\beta(\beta(s+\kappa)) - 7 = h^+(s+\kappa)$.

Second, let $\lambda^-(s)$ be the smallest possible sum of the weights of the nodes of a binary tree with weight s. By construction, $\lambda(s) \geqslant \lambda^-(s)$, and λ^- shines as the non-decreasing convex function defined by $\lambda^-(1) = 0$ and $\lambda^-(s) = s + \lambda^-(\lfloor s/2 \rfloor) + \lambda^-(\lceil s/2 \rceil)$ for all $s \geqslant 2$. Consequently, observing that

$$\lambda^+(s+\kappa) + v \leqslant 5(s+\kappa)\log_2(s+\kappa)/4 - 4(s+\kappa) + v \leqslant 5(s+4)\log_2(s+4)/4 - 4(s+3) + 4$$

whenever $s \geqslant 1$ suffices to check (by computer) that $\lambda(s) \geqslant \lambda^-(s) \geqslant s\lfloor \log_2(s) \rfloor \geqslant \lambda^+(s+\kappa)$ for all integers $s \leqslant 26$.

Then, if $s \geqslant 27$, observe that $\alpha(1-\alpha)(s+\kappa) \geqslant 2/3 \times 1/4 \times 30 \geqslant 5$. It follows that:

- $s_1 + \kappa \geqslant (1-\alpha)s + \kappa \geqslant (1-\alpha)(s+\kappa) \geqslant 5$;
- $s_{21} + \kappa \geqslant \alpha\gamma s + \kappa - \gamma - 1 = \alpha\gamma(s+\kappa) \geqslant 5$;
- $s_{22} + \kappa \geqslant \alpha(1-\gamma)s + \kappa + \gamma - 1 \geqslant \alpha(1-\gamma)(s+\kappa) \geqslant 5$.

Moreover, the function λ^+ is increasing on the interval $(16^\Delta/e, +\infty)$, and $16^\Delta/e \leqslant 5$. In addition, the equality $\lambda^+(x) = (\alpha+1)x + \lambda^+((1-\alpha)x) + \lambda^+(\alpha\gamma x) + \lambda^+(\alpha(1-\gamma)x$ is valid for all $x \geqslant 1$. Thus, the induction hypothesis proves that

$$\begin{aligned}
\lambda(s) &\geqslant s + s_2 + \lambda(s_1) + \lambda(s_{21}) + \lambda(s_{22}) \\
&\geqslant s + s_2 + \lambda^+(s_1+\kappa) + \lambda^+(s_{21}+\kappa) + \lambda^+(s_{22}+\kappa) + 3v \\
&\geqslant (\alpha+1)s + \lambda^+((1-\alpha)(s+\kappa)) + \lambda^+(\alpha\gamma(s+\kappa)) + \lambda^+(\alpha(1-\gamma)(s+\kappa)) + (3v-1) \\
&\geqslant \lambda^+(s+\kappa) - (\alpha+1)\kappa + (3v-1) = \lambda^+(s+\kappa) + v.
\end{aligned}$$

Finally, we check by hand that $\lambda(s) \geqslant s\lfloor \log_2(s) \rfloor \geqslant 5s\log_2(s)/4 - 4s + 4 \geqslant s\log_2(s)/\Delta - 4s + 4$ when $1 \leqslant s \leqslant 8$, whereas, since λ^+ is increasing on $[5, +\infty)$,

$$\begin{aligned}
\lambda(s) &\geqslant \lambda^+(s+\kappa) + v \geqslant \lambda^+(s+3) + 3 = (s+3)\log_2(s+3)/\Delta - 4s - 9 \\
&\geqslant (s\log_2(s) + 3\log_2(s+3))/\Delta - 4s - 9 \\
&\geqslant s\log_2(s)/\Delta + 15\log_2(12)/4 - 4s - 9 \geqslant s\log_2(s)/\Delta - 4s + 4
\end{aligned}$$

when $s \geqslant 9$. ◀

▶ **Lemma 8.** *Let α and β be real numbers such that $1/\sqrt{2} \leqslant \alpha < 3/4$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$. Then, let*

$$\hat{\alpha} = \frac{1 - 2\alpha + 6\alpha^2 - \sqrt{(1-\alpha)(5-\alpha)}}{5(2\alpha-1)}, \; \hat{\beta} = \frac{1 + \alpha - \sqrt{(1-\alpha)^2 + 4\alpha(\alpha^2-\beta)}}{2(1-\alpha^2+\beta)}\alpha,$$

as well as $\delta = \min\{\hat{\alpha}, \hat{\beta}/\alpha\}$.

We have $1/4 < \hat{\beta} \leqslant \beta$. Moreover, when $\hat{\alpha} \leqslant u \leqslant \alpha$ and $\hat{\beta} \leqslant y \leqslant \beta$, if we apply the $\mathsf{GC}(y,0)$-balancing algorithm on a $\mathsf{GCB}_{\hat{\alpha},1}[\alpha,\beta]$-tree $\mathcal{T}$, cases 1 to 5 are pairwise incompatible, and those rotations they trigger can always be performed; if the root of $\mathcal{T}$ is not y-GC-balanced, the algorithm transforms $\mathcal{T}$ into a $\mathsf{GCB}_{u,y}[\alpha,\beta]$-tree $\mathcal{T}'$ such that $\mathsf{bal}_\mathsf{C}(\mathcal{T}') \leqslant \mathsf{bal}_\mathsf{C}(\mathcal{T})$ and whose affected nodes are $(\delta, \hat{\beta})$-balanced.

Proof. Below, we will use the following inequalities, which are all easy to check with any computer algebra system (whereas some look too frightening to check by hand) whenever $1/\sqrt{2} \leqslant \alpha < 3/4$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$. Except the first three inequalities, each of them is labelled and later reused to prove another inequality with the same label; the first two inequalities already prove that $1/4 < \hat{\beta} \leqslant \beta$, as stated in Lemma 8.

$$1/4 < \hat{\beta}; \qquad \hat{\beta} \leqslant \beta;$$

$$\alpha < 2\hat{\beta}; \qquad \alpha^2 < \delta; \tag{3.1}$$

$$(1-\delta)(\alpha - \hat{\beta}) < \delta(1-\alpha); \tag{3.2} \qquad (1-\delta)(1-\hat{\gamma}) \leqslant \delta\hat{\gamma}(1-\alpha); \tag{3.3}$$

$$1 - \hat{\beta} < \delta; \tag{3.4} \qquad (1-\delta)(\alpha - \hat{\beta}) < \delta(\hat{\beta} - \alpha\beta); \tag{4.1}$$

$$(1-\delta)\beta < \delta(1-\alpha); \tag{4.2} \qquad \alpha - \hat{\beta} + \alpha\beta < \delta; \tag{4.3}$$

$$(1-\delta)\alpha\beta < \delta(1-\alpha); \tag{4.4} \qquad 1 \leqslant \delta(1+\beta); \tag{4.5$^\text{a}$}$$

$$1 - \delta + (\delta + \beta\delta - 1)\alpha \leqslant \delta\hat{\beta}; \tag{4.5$^\text{b}$} \qquad 1 + (\beta - 1)\hat{\gamma} < \delta. \tag{4.6}$$

Then, let $|m|$ denote the weight of a node u in the tree $\mathcal{T}$, and let $|m|'$ denote its weight in $\mathcal{T}'$. When these weights are equal, we will prefer using the notation $|m|$ even when considering the weight of m in $\mathcal{T}'$. The root of $\mathcal{T}$ is denoted by n, its left and right children are denoted by n_1 and n_2, and so on. Finally, we set $\hat{\gamma} = \hat{\beta}/\alpha$, so that $\delta = \min\{\hat{\alpha}, \hat{\gamma}\}$.

Since $2\beta \geqslant \alpha$ and n is α-C-balanced, observing that $2\beta|n| \geqslant \alpha|n| \geqslant |n_1| = |n_{11}| + |n_{12}|$ proves that the inequalities $|n_{11}| > \beta|n|$ and $|n_{12}| > \beta|n|$ are incompatible. Similarly, the nodes n_1 and n_2 are α-C-balanced, and thus observing that $2\beta|n| \geqslant \alpha|n| = \alpha|n_1| + \alpha|n_2| \geqslant |n_{11}| + |n_{21}|$ proves that the inequalities $|n_{11}| > \beta|n|$ and $|n_{21}| > \beta|n|$ are incompatible. Finally, for symmetry reasons, all four inequalities $|n_{ij}| > \beta|n|$ are incompatible: this makes our description of the algorithm unambiguous, as announced.

Then, proving that the desired rotations can indeed be performed simply requires showing that, if some case 1 to 4 happens, the grandchild n_{ij} be a grandchild of n whose weight $|n_{ij}|$ is maximal is an actual tree node instead of an empty node of weight $1/2$ or less. If this were the case, we would have $|n_i| = 2|n_{ij}| \leqslant 1$, and n_i would be either an empty node or a leaf; in both cases, we would have $|n| \geqslant 2|n_i|$, i.e., $|n| \geqslant 4|n_{ij}|$, contradicting the inequality $|n_{ij}| > y|n| \geqslant \hat{\beta}|n| > |n|/4$.

It remains to prove that each affected node m is δ-C-balanced; its children being α-C-balanced, it will then be $\hat{\beta}$-GC-balanced. Furthermore, if a rotation was triggered, n was not β-GC-balanced but its children were α-C-balanced, which proves that $\mathsf{bal}_{\mathsf{C}}(\mathcal{T}) > \beta/\alpha \geqslant \hat{\delta} \geqslant \mathsf{bal}_{\mathsf{C}}(\mathcal{T}')$.

When $|n_{11}| > w|n| \geqslant \hat{\beta}|n|$, a simple rotation is performed, and

$$|n_{11}| \leqslant \alpha|n_1| \leqslant \alpha^2|n| \leqslant \delta|n| = \delta|n_1|'; \tag{3.1}$$

$$\begin{aligned}(1-\delta)|n_{12}| &= (1-\delta)(|n_1| - |n_{11}|) \leqslant (1-\delta)(\alpha|n| - \hat{\beta}|n|)\\ &\leqslant \delta(1-\alpha)|n| \leqslant \delta(|n| - |n_1|) = \delta|n_2|;\end{aligned} \tag{3.2}$$

$$\begin{aligned}(1-\delta)|n_2| &= (1-\delta)(|n| - |n_1|) \leqslant (1-\delta)(|n| - |n_{11}|/\alpha) \leqslant (1-\delta)(1-\hat{\gamma})|n|\\ &\leqslant \delta\hat{\gamma}(1-\alpha)|n| \leqslant \delta(1/\alpha - 1)|n_{11}| < \delta(|n_1| - |n_{11}|) = \delta|n_{12}|;\end{aligned} \tag{3.3}$$

$$|n| = |n| - |n_{11}| \leqslant (1-\hat{\beta})|n| \leqslant \delta|n| = \delta|n_1|', \tag{3.4}$$

which means precisely that n and n_1 are δ-C-balanced in $\mathcal{T}'$.

Similarly, when $|n_{12}| > w|n| \geqslant \hat{\beta}|n|$, a double rotation is performed, and

$$\begin{aligned}(1-\delta)|n_{11}| &= (1-\delta)(|n_1| - |n_{12}|) \leqslant (1-\delta)(\alpha|n| - \hat{\beta}|n|)\\ &\leqslant \delta(\hat{\beta}|n| - \alpha\beta|n|) \leqslant \delta(|n_{12}| - \beta|n_1|) \leqslant \delta(|n_{12}| - |n_{122}|) = \delta|n_{121}|;\end{aligned} \tag{4.1}$$

$$(1-\delta)|n_{121}| \leqslant (1-\delta)\beta|n_1| \leqslant \delta(|n_1| - \alpha|n_1|) \leqslant \delta(|n_1| - |n_{12}|) = \delta|n_{11}|; \tag{4.2}$$

$$\begin{aligned}|n_1|' &= |n_1| - |n_{12}| + |n_{121}| \leqslant |n_1| - |n_{12}| + \beta|n_1|\\ &\leqslant (\alpha - \hat{\beta} + \alpha\beta)|n| \leqslant \delta|n| = \delta|n_{12}|';\end{aligned} \tag{4.3}$$

$$(1-\delta)|n_{122}| \leqslant (1-\delta)\beta|n_1| \leqslant (1-\delta)\alpha\beta|n| \leqslant \delta(1-\alpha)|n| \leqslant \delta(|n| - |n_1|) = \delta|n_2|; \tag{4.4}$$

$$\begin{aligned}(1-\delta)|n_2| &\leqslant (1-\delta)|n_2| + \beta\delta|n_1| - \delta|n_{121}| = (1-\delta)|n| + (\delta + \beta\delta - 1)|n_1| - \delta|n_{121}|\\ &\leqslant (1-\delta)|n| + (\delta + \beta\delta - 1)\alpha|n| - \delta|n_{121}| &(4.5^{\mathrm{a}})\\ &\leqslant \delta\hat{\beta}|n| - \delta|n_{121}| \leqslant \delta|n_{12}| - \delta|n_{121}| = \delta|n_{122}|; &(4.5^{\mathrm{b}})\end{aligned}$$

$$\begin{aligned}|n|' &= |n| - |n_1| + |n_{122}| \leqslant |n| + (\beta - 1)|n_1|\\ &\leqslant |n| + (\beta - 1)|n_{12}|/\alpha \leqslant |n| + (\beta - 1)\hat{\gamma}|n| \leqslant \delta|n| = \delta|n_{12}|',\end{aligned} \tag{4.6}$$

which means precisely that n, n_1 and n_{12} are δ-C-balanced in $\mathcal{T}'$.

Finally, the cases where $|n_{21}| > w|n|$ or $|n_{22}| > w|n|$ are symmetrical, and therefore the conclusion of Lemma 8 is valid in these cases too. ◀

Scheduling on Identical Machines with Setup Time and Unknown Execution Time

Yasushi Kawase
The University of Tokyo, Japan

Kazuhisa Makino
Kyoto University, Japan

Vinh Long Phan
Toyota Motor Corporation, Japan

Hanna Sumita
Institute of Science Tokyo, Japan

Abstract

In this study, we investigate a scheduling problem on identical machines in which jobs require initial setup before execution. We assume that an algorithm can dynamically form a batch (i.e., a collection of jobs to be processed together) from the remaining jobs. The setup time is modeled as a known monotone function of the set of jobs within a batch, while the execution time of each job remains unknown until completion. This uncertainty poses significant challenges for minimizing the makespan. We address these challenges by considering two scenarios: each job batch must be assigned to a single machine, or a batch may be distributed across multiple machines. For both scenarios, we analyze settings with and without preemption. Across these four settings, we design online algorithms that achieve asymptotically optimal competitive ratios with respect to both the number of jobs and the number of machines.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online scheduling, Competitive analysis, Makespan minimization, Identical machines scheduling

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.41

Related Version *Full Version*: https://arxiv.org/abs/2507.11311

Funding This work was partially supported by the joint project of Kyoto University and Toyota Motor Corporation, titled "Advanced Mathematical Science for Mobility Society", JST ERATO Grant Number JPMJER2301, JST PRESTO Grant Number JPMJPR2122, and JSPS KAKENHI Grant Numbers JP17K12646, JP19K22841, JP20H00609, JP20H05967, JP20K19739, JP21K17708, JP21H03397, and JP25K00137.

1 Introduction

Efficient job allocation across multiple workers or machines is crucial for optimizing productivity in industrial environments. For example, consider distributing computational jobs across multiple virtual machines. The processing time for a batch of jobs consists of two components: the *setup time*, which includes configuring the environment or installing necessary software or libraries, and the *execution time*, which represents the duration of performing the actual tasks. While setup times are typically known in advance, the execution time of each job is often unpredictable until the job is completed. This uncertainty complicates the design of scheduling algorithms aimed at minimizing the *makespan*, which is the time when all jobs are completed.

In this paper, we introduce a scheduling problem involving n jobs on m identical machines with known setup times and unknown execution times, referred to as the *unknown execution time scheduling (UETS)* problem. We assume that an algorithm can dynamically form a

© Yasushi Kawase, Kazuhisa Makino, Vinh Long Phan, and Hanna Sumita;
licensed under Creative Commons License CC-BY 4.0

batch – a collection of jobs that are processed together – from the remaining jobs. The setup time for a batch is defined as a set function over subsets of jobs, representing the total time required to prepare the jobs in the batch for processing. We explore two primary scenarios: one called *sUETS*, in which each constructed batch must be assigned to a single machine, and the other called *mUETS*, in which a batch can be distributed across multiple machines. Additionally, we examine two settings based on whether preemption is allowed. In the non-preemptive setting, once a batch process is started, it must be run without interruption until completion. In the preemptive setting, the processing of a batch can be interrupted. After an interruption, only uncompleted jobs are regrouped into batches and the process is resumed from the setup phase (completed jobs do not need to be processed again). Note that, the non-preemptive setting generally requires longer processing times compared to the preemptive one, because preemption allows for more flexible handling of incomplete batches and may reduce redundant processing.

Our objective is to design online algorithms for these scheduling problems. We analyze the performance of an online algorithm by the competitive ratio, which is the worst-case ratio between the makespan achieved by the online algorithm and that of an optimal offline algorithm. We assume that offline algorithms have complete knowledge of each job's execution time in advance. We refer to the schedule produced by the optimal offline algorithm as the *optimal schedule* and its makespan as the *optimal makespan*. We refer to the makespan of the schedule produced by an algorithm simply as the algorithm's makespan. An online algorithm is said to be *ρ-competitive* if its makespan is at most ρ times the optimal makespan for any instance. We will design online algorithms with asymptotically optimal competitive ratios with respect to both the number of machines and the number of jobs.

1.1 Our results

We study the competitive ratios for the scheduling problem with setup time and unknown execution times. We first show that any algorithm designed for jobs with a release time of 0 can be adapted to handle arbitrary release times at the expense of only a constant factor increase in the competitive ratio. We formally state this in Theorem 2. Thus, we may assume that the release time of each job is 0, i.e., jobs can start processing immediately at time 0. A summary of our results is provided in Table 1.

For the non-preemptive sUETS problem, we present two algorithms. First, we construct an algorithm with a competitive ratio of m (Theorem 3). Second, we design an algorithm that is $\mathrm{O}\big(\sqrt{n/m}\big)$-competitive (Theorem 5). By combining these two algorithms, we obtain an $\mathrm{O}(n^{1/3})$-competitive algorithm (Corollary 6). Moreover, we prove lower bounds showing that every online algorithm for this problem must have a competitive ratio of at least $\Omega(m)$ (Theorem 11) and $\Omega(n^{1/3})$ (Theorem 10). For the preemptive sUETS problem, we design an algorithm with a competitive ratio of $\mathrm{O}(\log n/\log\log n)$ (Theorem 8), and we demonstrate that the competitive ratios of $\mathrm{O}(m)$ and $\mathrm{O}(\log n/\log\log n)$ are best possible (Theorem 11).

Turning to the non-preemptive mUETS problem, we first construct an $\mathrm{O}(\sqrt{m})$-competitive algorithm (Theorem 12). By integrating this with the $\mathrm{O}\big(\sqrt{n/m}\big)$-competitive algorithm for the non-preemptive sUETS problem (Theorem 5), we derive an $\mathrm{O}(n^{1/4})$-competitive algorithm for the non-preemptive mUETS problem (Corollary 13). We further prove that $\mathrm{O}(\sqrt{m})$ and $\mathrm{O}(n^{1/4})$ are optimal (Theorem 17). Finally, for the preemptive mUETS problem, we establish that the best possible competitive ratios are $\Theta(\log m/\log\log m)$ and $\Theta(\log n/\log\log n)$ (Theorems 8, 15, and 18).

Due to space limitations, some proofs are omitted in this version. The complete proofs are available in a full version of the paper.

Table 1 The competitive ratios of the UETS problems with n jobs and m machines.

	single	multiple
non-preemptive	$\Theta(m)$ (Thms. 3 and 10) $\Theta(n^{1/3})$ (Cor. 6 and Thm. 10)	$\Theta(\sqrt{m})$ (Thms. 12 and 17) $\Theta(n^{1/4})$ (Cor. 13 and Thm. 17)
preemptive	$\Theta(m)$ (Thms. 3 and 11) $\Theta\left(\frac{\log n}{\log \log n}\right)$ (Thms. 8 and 11)	$\Theta\left(\frac{\log m}{\log \log m}\right)$ (Thms. 15 and 18) $\Theta\left(\frac{\log n}{\log \log n}\right)$ (Thms. 8 and 18)

1.2 Related work

Classical scheduling problems typically assume that the setup time is 0 for every job batch. In this case, the well-known *list scheduling algorithm* achieves $(2 - 1/m)$-competitive, which is proven to be optimal [16–18, 32]. This algorithm assigns unprocessed jobs to any available machine in the order they appear on the job list, disregarding execution times.

When execution times are unknown until job completion, the problem falls under the category of *non-clairvoyant scheduling* [26]. Recent studies have explored non-clairvoyant scheduling in input prediction models [3, 21]. Shmoys et al. [32] introduced a technique to transform a ρ-competitive algorithm for a scheduling problem without release times into a 2ρ-competitive algorithm for the same problem with release times. For comprehensive overviews of online scheduling, see surveys [29, 31] and the book by Pinedo [28].

Scheduling problems that incorporate both setup times and execution times arise in various applications, such as cloud computing (where virtual machines must be initialized based on job types) and and production systems (where machines require reconfiguration, such as changing molds or colors) [1, 9, 13, 20, 25, 30]. For example, in plastic production systems, attaching a specific mold to a machine constitutes the setup time. If consecutive jobs use the same mold, no additional setup time is required. The total setup time for a batch can be precomputed as the minimum time needed for attaching, exchanging, and removing molds.

Gambosi and Nicosia [13] studied an online version of scheduling with setup times in the one-by-one model. Mäcker et al. [24] investigated non-clairvoyant scheduling with setup times and proposed an $O(\sqrt{n})$-competitive algorithm for minimizing maximum flow time on a single machine. Dogeas et al. [10] considered a scenario where the execution time of each job is only revealed after an obligatory test with a known duration. This test time can be viewed as a kind of setup time, although it differs in that the execution time becomes apparent.

Goko et al. [15] introduced a scheduling model with a metric state space that involves setup time and unknown execution time settings together. Their model includes a scheduling problem faced by repair companies where each worker needs to visit customers' houses, do the repair jobs, and then return to the office. The setup time corresponds to the shortest tour length for the customers' houses, and the execution time corresponds to the duration of the repair jobs. Their model also includes the online dial-a-ride problem [2,4–7,11,22,23], in which taxis are offered to pick up and drop passengers for transportation jobs. Our preemptive mUETS problem can be seen as an abstraction of their model, and hence, a similar approach can be used to solve it. However, the other problems (i.e., preemptive/non-preemptive sUETS and non-preemptive mUETS) are different and require other approaches.

The setting in which all jobs have an execution time of 0 and are released at time 0, with only the setup time considered as processing time, can be viewed as an offline load balancing problem. This problem is NP-hard even if the setup time is additive and the number of machines is two, as this special case corresponds to the PARTITION problem [14]. For the additive setup time, the list scheduling algorithm works as a $(2-1/m)$-approximation algorithm. Moreover, this problem admits a polynomial time approximation scheme (PTAS) [19]. Svitkina and Fleischer [33] presented an $\mathrm{O}(\sqrt{n/\log n})$-approximation algorithm for the submodular setup times, and demonstrated that this is best possible. Furthermore, Nagano and Kishimoto [27] provided a $2\cdot(\max_{j\in[m]}\frac{|S_j^*|}{1+(|S_j^*|-1)(1-\kappa_c(S_j^*))})$-approximation algorithm for the subadditive setup times, where $(S_1^*,\ldots,S_m^*)$ is an optimal partition and $\kappa_c(S)$ is the curvature of c at $S\subseteq J$.

2 Preliminaries

For a positive integer k, we write $[k]$ to denote the set $\{1,2,\ldots,k\}$. We are given a set J of n jobs and m identical machines. We assume that all jobs are given and released at time 0. As we will show in Section 2.2, this assumption only increases a constant factor in the competitive ratios. We denote the set of machines by $[m]$. We assume that $n\geq m\geq 2$, as the optimal scheduling is clear otherwise. The jobs are executed in batches, and each batch's processing time consists of two components: the *setup time* and the *execution time*.

The setup time is the total time one machine takes to set up jobs in the batch. For a batch $X\subseteq J$, the setup time for X is represented as $c(X)\in\mathbb{R}_+$. The execution time for each job $j\in J$, denoted by p_j, is unknown until its process is completed. We write $p(X)$ to denote the total execution time of a batch $X\subseteq J$, i.e., $p(X)=\sum_{j\in X}p_j$. Consequently, the overall processing time of a batch X on a single machine is given by $p(X)+c(X)$. We assume that the setup time $c(X)$ for every $X\subseteq J$ is available in advance, while the execution time of each job is unknown until its completion.

We also deal with a situation where one batch can be assigned to multiple machines. We assume that when a batch X is assigned to k machines, each of the machines incurs a setup time $c(X)$, and the jobs are executed on a first-come, first-served basis. For example, when a batch X is processed in a production system, we assume that each machine forms a queue of the attachments needed for jobs in X, and proceeds as follows: (i) dequeues and installs the first attachment in the queue, (ii) repeatedly executes an unprocessed job that is compatible with the current attachment as long as such a job exists, and (iii) removes the current attachment and returns to step (i) if the attachment queue is not empty. The minimum processing time for batch X across k machines falls within the range of $c(X)+p(X)/k$ and $c(X)+p(X)/k+\max_{j\in X}p_j$.

At each time, we can assign a batch of jobs that have not yet been completed and are not assigned anywhere else. In the sUETS and mUETS problems, a batch can be allocated to a single machine and multiple machines, respectively. Our objective is to minimize the makespan, which is the time at which all the jobs are completed.

We assume that the setup time function $c\colon 2^J\to\mathbb{R}_+$ is *monotone subadditive*. Monotonicity means that $c(X)\leq c(Y)$ for any $X\subseteq Y\subseteq J$. Subadditivity means that for any disjoint subsets X and Y from J, the sum of their setup times is greater than or equal to the setup time of their union, i.e., $c(X)+c(Y)\geq c(X\cup Y)$. Note that this assumption of subadditivity does not lose generality. To observe this, for any (not necessarily subadditive) function c, define the function $\bar{c}(X)=\min_{(X_1,\ldots,X_\ell)\colon\text{partition of }X}\sum_{i=1}^{\ell}c(X_i)$. This new function $\bar{c}$ is monotone and subadditive by construction – it represents the minimum total setup time

achievable if one were allowed to split the batch X into sub-batches and incur the setup cost separately for each. Importantly, replacing c with $\bar{c}$ in our analysis does not change the fundamental nature of the scheduling problem, and thus we assume without loss of generality that the setup time function is monotone and subadditive.

We explore two settings based on whether an allocated batch can be preempted. In the non-preemptive setting, once a batch is assigned, it must be processed to completion without interruption. Conversely, in the preemptive setting, the processing of a batch can be halted and canceled. Suppose that a batch X is assigned to k machines and is preempted after t units of time from the start of its processing. In this case, we assume that the machines will become available to process another batch after at most $c(X) + \max_{j \in X} p_j$ units of time. This assumption ensures that the processing time does not increase even if preemption occurs during the setup phase. Furthermore, at the moment of preemption, let $X' \subseteq X$ denote the set of jobs that have been completed. Then, the following inequality holds: $c(X) + p(X')/k + \max_{j \in X} p_j \geq t$. This condition ensures that the total time spent up to the point of preemption is consistent with the setup and execution times for the completed jobs.

Note that, in any setting, we can assume that an optimal schedule does not perform preemptions and assigns only one batch to each machine at the beginning because the setup time c is subadditive. Thus, an optimal schedule can be represented as a partition of jobs $(X_1^*, \ldots, X_m^*)$ and the optimal makespan is $\max_{i \in [m]}(c(X_i^*) + p(X_i^*))$ independently of the settings. Based on this observation, we derive a lower bound for the optimal makespan, as formalized in the following lemma.

▶ **Lemma 1.** *The optimal makespan is at least*

$$\max \left\{ \min_{(X_1,\ldots,X_m):\, \textit{partition of } J} \max_{i \in [m]} c(X_i),\ (c(J) + p(J))/m,\ \max_{j \in J}(c(\{j\}) + p_j) \right\}.$$

Proof. Let $(X_1^*, \ldots, X_m^*)$ be an optimal schedule. Then, the optimal makespan is at least

$$\max_{i \in [m]} (c(X_i^*) + p(X_i^*)) \geq \max_{i \in [m]} c(X_i^*) \geq \min_{(X_1,\ldots,X_m):\, \text{partition of } J} \max_{i \in [m]} c(X_i).$$

Additionally, we have

$$\max_{i \in [m]} (c(X_i^*) + p(X_i^*)) \geq \max_{j \in J} (c(\{j\}) + p_j) \geq (c(J) + p(J))/m$$

by monotone subadditivity of the setup time. Therefore, we have the lower bound. ◀

2.1 Examples of setup times

This subsection illustrates several examples of setup times raised in practical applications. In addition, we discuss the approximability of the partition (load balancing) problem

$$\min_{(X_1,\ldots,X_m):\, \text{partition of } J} \max_{i \in [m]} c(X_i) \tag{1}$$

for each class of setup times.

Constant setup times. One of the simplest examples of the setup time family is the constant setup time, where the setup time is a certain constant independent of the batch. Formally, the setup time is represented as $c(X) = 1$ for $X \in 2^J \setminus \{\emptyset\}$ and $c(\emptyset) = 0$. It is not difficult to see that any partition of the jobs $(X_1, \ldots, X_m)$ is optimal for the problem (1).

Type-specific setup times. In production systems with attachments, jobs are partitioned based on their types. Let T be the set of types and $(J_t)_{t\in T}$ is the partition of jobs with types T, i.e., $\bigcup_{t\in T} J_t = J$ and $J_t \cap J_{t'} = \emptyset$ for all distinct $t, t' \in T$. Let w_t be the setup time for type t. Then, the setup time for $X \subseteq J$ is defined as $c(X) = \sum_{t\in T:\, X\cap J_t\neq\emptyset} w_t$. Additionally, we say that a type-specific setup time c is unweighted if $w_t = 1$ for all $t \in T$, i.e., $c(X) = |\{t \in T : X \cap J_t \neq \emptyset\}|$ for $X \subseteq J$. We can obtain a PTAS for the problem (1) with type-specific setup times by treating each J_t as a single job and applying a PTAS for the load balancing problem [19]. Moreover, for the unweighted case, the problem (1) is solvable in polynomial time because an optimal solution can be obtained by simply balancing the number of types assigned to each machine.

Library-based setup times. In cloud computing environments, where each job requires the installation of multiple libraries, and each library takes a fixed amount of time to install. The setup time needed to process a batch of jobs is the total installation time of the required libraries. Let L be the set of libraries, and let $L_j \subseteq L$ be the set of required libraries for job $j \in J$. Let w_ℓ be the installation time for library $\ell \in L$. Then, the setup time for $X \subseteq J$ is defined as $c(X) = \sum_{\ell\in\bigcup_{j\in X} L_j} w_\ell$. Note that if every job $j \in J$ requires a unique library (i.e., $|L_j| = 1$), this is a type-specific setup time. This setup time is a monotone submodular function; more specifically, it is a weighted coverage function. Hence, we can apply the $O(\sqrt{n/\log n})$-approximation algorithm for the uniform submodular load balancing problem [33] to solve the problem (1).

TSP-based setup times. For applications like repair companies, the setup time is defined by the optimal value of a traveling salesman problem (TSP) instance. Let (V, d) be a metric space with an origin $o \in V$. The origin corresponds to the location of the company. Each job $j \in J$ is associated with a point $v_j \in V$. Then, the setup time for $X \subseteq J$ is defined as the optimal value of the TSP on $V' = \{v_j : j \in X\} \cup \{o\}$. Note that this class of setup times is a generalization of type-specific setup times. Indeed, for a star metric $(T \cup \{o\}, d)$ where $d(t, o) = w_t/2$ for all $t \in T$ and $d(t, t') = (w_t + w_{t'})/2$ for all distinct $t, t' \in T$, the setup time is $c(X) = \sum_{t\in T:\, X\cap J_t\neq\emptyset} w_t$ for $X \subseteq J$ when $v_j = t$ for each job $j \in J_t$. For TSP-based setup times, the problem (1) can be viewed as the m-TSP. It is known that the m-TSP admits a 2.5-approximation algorithm [8, 12].

2.2 Reduction of the Problem with Release Time

In this subsection, we show that our assumption that all jobs have a release time of 0 (i.e., each job is available for processing from time 0) does not lose generality. Suppose instead that each job has a release time, and its existence is hidden until its release time. Let ALG be a ρ-competitive algorithm for a UETS problem in a certain setting without release times. Then, we demonstrate that ALG can be transformed into a $(2\rho + 1)$-competitive algorithm for the UETS problem with release times of the corresponding setting. For this end, we use the *IGNORE* strategy, which appeared in the paper of Shmoys et al. [32] and is named by Ascheuer et al. [2].

The IGNORE strategy keeps the machines remain idle until a set S of jobs appears. Then, IGNORE immediately decides a schedule for the jobs in S following the algorithm ALG and assigns job batches to machines. We refer to this schedule as a subschedule for S. All jobs that arrive during the process of the subschedule are temporarily ignored until the subschedule for S is completed. After all the machines complete the jobs in S, IGNORE decides a subschedule for all new jobs that arrived during the previous process. If there are no such jobs, all the machines become idle. The IGNORE strategy repeats this procedure.

▶ **Theorem 2.** *The* IGNORE *strategy with a ρ-competitive algorithm* ALG *is $(2\rho+1)$-competitive for the UETS problem with release time.*

Proof. Fix an instance, and let OPT be the optimal makespan. For a subset of jobs J', we denote $\mathrm{ALG}(J')$ as the makespan of the algorithm, assuming that jobs in J' are given at time 0. Note that $\mathrm{ALG}(J') \leq \rho \cdot \mathrm{OPT}$ by the monotonicity of the optimal makespan with respect to the jobs.

If the instance contains no job, then IGNORE is clearly optimal. Thus, we assume that the instance contains at least one job. Let t^* be the last release time of the jobs. Then, $\mathrm{OPT} \geq t^*$ as the last released job can only be processed from t^*.

Suppose that the machines are idle at time t^*. Let R be the set of jobs processed in the last subschedule by the IGNORE strategy. Then, the makespan of IGNORE is

$$t^* + \mathrm{ALG}(R) \leq \mathrm{OPT} + \rho \cdot \mathrm{OPT} \leq (2\rho+1) \cdot \mathrm{OPT}.$$

On the other hand, suppose that some machines are not idle at time t^*. Then t^* is in the second last subschedule of the algorithm, and the last subschedule starts right after the second-to-last subschedule ends. Let R and S be the sets of jobs processed in the last and the second-to-last subschedules, respectively. Then, the makespan of IGNORE is

$$t^* + \mathrm{ALG}(S) + \mathrm{ALG}(R) \leq \mathrm{OPT} + \rho \cdot \mathrm{OPT} + \rho \cdot \mathrm{OPT} = (2\rho+1) \cdot \mathrm{OPT}.$$

Therefore, the competitive ratio of IGNORE is at most $2\rho+1$. ◀

It should be noted that this reduction remains valid even when the competitive ratio ρ of ALG depends on the number of machines m or the number of jobs n, provided it is monotonically nondecreasing with respect to n. From this theorem, the optimum competitive ratio is the same up to a constant factor regardless of the release time.

3 Single Machine Batch Allocation

In this section, we analyze the sUETS problem where each batch can be allocated only on a single machine. Recall that n is the number of jobs and m is the number of machines.

3.1 Algorithms

We first observe that the algorithm of processing the batch consisting of all jobs by a single machine without preemption is m-competitive, where m is the number of machines. Note that this algorithm is feasible for any settings of sUETS and mUETS since no cancellation is performed.

▶ **Theorem 3.** *The algorithm of processing the batch consisting of all jobs by a single machine is m-competitive for the non-preemptive sUETS problem.*

Proof. As the makespan of the algorithm is $c(J)+p(J)$ and the optimal makespan is at least $(C(J)+p(J))/m$ by Lemma 1, the competitive ratio is at most

$$\frac{c(J)+p(J)}{\max_{i\in[m]}(c(X_i^*)+p(X_i^*))} \leq \frac{c(J)+p(J)}{\sum_{i\in[m]}(c(X_i^*)+p(X_i^*))/m} \leq \frac{c(J)+p(J)}{(c(J)+p(J))/m} = m,$$

where the second inequality is implied by the subadditivity of the setup time c. ◀

Second, we analyze the competitive ratio of the list scheduling algorithm.

▶ **Theorem 4.** *The algorithm that greedily assigns a batch consisting of a single unprocessed job to any available single machine is* $\mathrm{O}(n/m)$*-competitive for the non-preemptive sUETS problem.*

Proof. The makespan of the algorithm is at most

$$\frac{1}{m}\sum_{j\in J}(c(\{j\})+p_j)+\max_{j\in J}(c(\{j\})+p_j)\le\frac{n\cdot\mathrm{OPT}}{m}+\mathrm{OPT}=\mathrm{O}(n/m)\cdot\mathrm{OPT},$$

where the inequality holds by Lemma 1. Thus, the competitive ratio is at most $\mathrm{O}(n/m)$. ◀

Next, we provide an $\mathrm{O}\big(\sqrt{n/m}\big)$-competitive algorithm, which is an improved version of the list scheduling algorithm. The idea is to reduce the total setup time by grouping jobs. The algorithm first divides the jobs into $m+\lceil\sqrt{mn}\rceil$ batches in such a way that this partition minimizes the maximum setup time, under the condition that each batch contains at most $\lceil\sqrt{n/m}\rceil$ jobs. Then, it processes the batches according to the list scheduling algorithm, which greedily allocates an unprocessed batch to any available machine. The formal description of this algorithm is summarized in Algorithm 1.

Algorithm 1 $\mathrm{O}(\sqrt{n/m})$-competitive algorithm for the non-preemptive sUETS problem.

```
Let k = m + ⌈√(mn)⌉;
Compute a k-partition (X_1, ..., X_k) of the jobs J such that |X_i| ≤ ⌈√(n/m)⌉ for all
  i ∈ [k], and among such partitions, minimize max_{i∈[k]} c(X_i);
for i ← 1, 2, ..., k do
    Wait until at least one machine is available;
    Assign batch X_i to an available machine;
```

▶ **Theorem 5.** *Algorithm 1 is* $\mathrm{O}\big(\sqrt{n/m}\big)$*-competitive for the non-preemptive sUETS problem.*

Proof. Let $(X_1^*,\dots,X_m^*)$ be the optimal schedule and let OPT denote the optimal makespan $\max_{i\in[m]}(c(X_i^*)+p(X_i^*))$. Define $k=m+\lceil\sqrt{mn}\rceil$, and let $(X_1,\dots,X_k)$ be a k-partition of the jobs J that minimizes $\max_{i\in[k]}c(X_i)$ subject to $|X_i|\le\lceil\sqrt{n/m}\rceil$ for all $i\in[k]$.

We first observe that $\max_{i\in[k]}c(X_i)\le\mathrm{OPT}$. This is because by dividing each X_i^* with $i\in[m]$ into $\lceil|X_i^*|/\sqrt{n/m}\rceil$ sub-batches of almost equal size, we can obtain a partition $(Y_1,\dots,Y_k)$ that refines $(X_1^*,\dots,X_m^*)$. Here, we have $|Y_{i'}|\le\max_{i\in[m]}\left\lceil|X_i^*|/\lceil|X_i^*|/\sqrt{n/m}\rceil\right\rceil\le\lceil\sqrt{n/m}\rceil$ for all $i'\in[k]$. Thus, it holds that $\max_{i\in[k]}c(X_i)\le\max_{i\in[k]}c(Y_i)\le\max_{i\in[m]}c(X_i^*)\le\mathrm{OPT}$.

Then, the makespan of the algorithm is at most

$$\begin{aligned}&\frac{1}{m}\sum_{i\in[k]}(c(X_i)+p(X_i))+\max_{i\in[k]}(c(X_i)+p(X_i))\\&\quad\le\frac{k\cdot\mathrm{OPT}+p(J)}{m}+\max_{i\in[k]}|X_i|\cdot\max_{j\in J}(c(\{j\})+p_j)\\&\quad\le\Big(k/m+1+\lceil\sqrt{n/m}\rceil\Big)\cdot\mathrm{OPT}=\mathrm{O}\big(\sqrt{n/m}\big)\cdot\mathrm{OPT},\end{aligned}$$

where the first inequality follows from the subadditivity of the setup time and the second inequality from Lemma 1. Hence, Algorithm 1 is $\mathrm{O}\big(\sqrt{n/m}\big)$-competitive. ◀

By combining Theorems 3 and 5, we obtain the following corollary.

▶ **Corollary 6.** *There exists an* $O(n^{1/3})$*-competitive algorithm for the non-preemptive sUETS problem.*

Proof. If $m \leq n^{1/3}$, then the algorithm that processes the batch J by a single machine is $m = O(n^{1/3})$-competitive by Theorem 3. On the other hand, if $m > n^{1/3}$, the competitive ratio of Algorithm 1 is $O\left(\sqrt{n/m}\right) = O(n^{1/3})$ by Theorem 5. Thus, an $O(n^{1/3})$-competitive algorithm exists for either case. ◀

▶ Remark 7. Obtaining the partition at line 2 in Algorithm 1 is computationally hard. Thus, we mention the competitive ratio when we can utilize an α-approximation algorithm for the problem (1). Let $(X'_1, \dots, X'_m)$ be an m-partition of the jobs J that is obtained by the approximation algorithm. Let $k' = m + \left\lceil \sqrt{mn/\alpha} \right\rceil$. Then, we can easily construct a k'-partition $(Y'_1, \dots, Y'_{k'})$ that is a refinement of $(X'_1, \dots, X'_m)$ and satisfies $|Y'_i| \leq \left\lceil \sqrt{\alpha \cdot n/m} \right\rceil$ for all $i \in [k']$. By using k' instead of k and employing this partition at line 2, Algorithm 1 is $O\left(\alpha + \sqrt{\alpha \cdot n/m}\right)$-competitive for the non-preemptive sUETS problem. By combining this with Theorem 4, we can obtain an $O(\min\{\sqrt{\alpha \cdot n/m}, n/m\})$-competitive algorithm. Moreover, by combining this with Theorem 3, we can also obtain an $O(\min\{(\alpha n)^{1/3}, \sqrt{n}\})$-competitive algorithm.

In the rest of this subsection, we provide an $O(\log n / \log\log n)$-competitive algorithm for the preemptive sUETS problem. For a given instance of the sUETS problem, let q be an integer such that $q^q \geq n > (q-1)^{q-1}$. Note that $q = \Theta(\log n / \log\log n)$. The execution of our algorithm consists of $q+1$ phases. Without loss of generality, we may assume that $m \geq 2q$, since otherwise the competitive ratio of the algorithm in Theorem 3 is $m = O(q) = O(\log n / \log\log n)$.

The intuition of our algorithm is as follows. Our algorithm repeatedly completes a $(1 - 1/q)$-fraction of the jobs in each phase. Consequently, all the jobs can be processed in $O(q)$ iterations. This increases the cost of the setup time by a factor of $O(q)$. Additionally, we ensure that at least m/q machines work at any point of the algorithm. This guarantees that the cost in terms of execution time will only increase by a factor of at most q. By these properties, the competitive ratio of the algorithm is $O(q) = O(\log n / \log\log n)$.

Let us explain our algorithm more precisely. Initially, it computes an m-partition $(X_1, \dots, X_m)$ of the jobs J that minimizes the maximum setup time $\max_{i \in [m]} c(X_i)$. In the first phase, X_i is allocated to machine i for each $i \in [m]$. Then, each machine processes the assigned batch until either the machine completes the batch or the number of uncompleted machines becomes less than or equal to $\lfloor m/q \rfloor$. At that time, the machines still processing will preempt their batches. In the kth phase ($k = 2, 3, \dots, q$), the algorithm computes an m-partition $(Y_1^{(k)}, \dots, Y_m^{(k)})$ of the remaining jobs such that, for any $i \in [m]$, $|Y_i^{(k)}| \leq n/q^{k-1}$ and $Y_i^{(k)} \subseteq X_j$ for some $j \in [m]$. We can always obtain such a partition by dividing each batch left in the previous phase into q sub-batches. Thereafter, each machine processes the assigned batch until either the batch is completed or the number of uncompleted machines becomes less than or equal to $\lfloor m/q \rfloor$. At the end of qth phase, the number of remaining jobs is at most $(n/q^{q-1}) \cdot \lfloor m/q \rfloor \leq m \cdot (n/q^q) \leq m$. In the $(q+1)$st phase, the algorithm assigns up to one remaining job to each machine. A formal description of the algorithm is provided in Algorithm 2.

▶ **Theorem 8.** *Algorithm 2 is* $O\left(\frac{\log n}{\log\log n}\right)$*-competitive for the preemptive sUETS problem.*

Proof. Let q be an integer such that $q^q \geq n > (q-1)^{q-1}$. If $m < 2q$, then the competitive ratio is $m = O(q) = O(\log n / \log\log n)$ by Theorem 3. Thus, we assume $m \geq 2q$.

Algorithm 2 $\mathrm{O}\left(\frac{\log n}{\log\log n}\right)$-competitive algorithm for the preemptive sUETS problem.

1 Let q be an integer such that $q^q \geq n > (q-1)^{q-1}$;
2 **if** $m < 2q$ **then** Assign batch J to one machine and exit;
3 Compute an m-partition $(X_1, \ldots, X_m)$ of the jobs J that minimizes $\max_{i\in[m]} c(X_i)$;
4 Let $R^{(0)} \leftarrow J$ be the set of uncompleted jobs;
5 **for** $k \leftarrow 1, 2, \ldots, q$ **do** // phase k
6 Compute an m-partition of $(Y_1^{(k)}, \ldots, Y_m^{(k)})$ of the remaining jobs $R^{(k-1)}$ such that, for each $i \in [m]$, $|Y_i^{(k)}| \leq n/q^{k-1}$ and $Y_i^{(k)} \subseteq X_j$ for some $j \in [m]$;
7 Assign batch $Y_i^{(k)}$ to machine i for each $i \in [m]$;
8 Each machine processes the assigned batch until the batch is completed or the number of uncompleted machines becomes less than or equal to $\lfloor m/q \rfloor$;
9 Let $R^{(k)}$ be the set of uncompleted jobs;
10 Assign up to one job $R^{(q)}$ per machine; // phase $q+1$

Let OPT be the optimal makespan and let $p_{\max} = \max_{j\in J} p_j$. In addition, let τ_k be the time length of phase $k \in [q+1]$. By the definition of the algorithm, the jobs in $R^{(k-1)} \setminus R^{(k)}$ are completed in phase $k \in [q]$. Thus, the total execution time of jobs in phase $k \in [q]$ is at most $p(R^{(k-1)} \setminus R^{(k)}) + p_{\max} \cdot \lfloor m/q \rfloor$ because at most $\lfloor m/q \rfloor$ jobs are partially executed. Taking into account the setup time of at most $\max_{i\in[m]} c(X_i)$ and the time for preemption of at most $\max_{i\in[m]} c(X_i) + p_{\max}$, the length of phase $k \in [q]$ is

$$\begin{aligned}\tau_k &\leq \max_{i\in[m]} c(X_i) + \frac{p(R^{(k-1)} \setminus R^{(k)}) + p_{\max} \cdot \lfloor m/q \rfloor}{\lfloor m/q \rfloor} + \left(\max_{i\in[m]} c(X_i) + p_{\max}\right) \\ &\leq \frac{q}{m-q} \cdot p(R^{(k-1)} \setminus R^{(k)}) + 4\mathrm{OPT} \leq \frac{2q}{m} \cdot p(R^{(k-1)} \setminus R^{(k)}) + 4\mathrm{OPT},\end{aligned}$$

where the second inequality is by Lemma 1 and the third is by $m \geq 2q$. In addition, by Lemma 1, the time length τ_{q+1} of phase $q+1$ is

$$\tau_{q+1} \leq \max_{j\in J}(c(\{j\}) + p_j) \leq \mathrm{OPT}.$$

Thus, the makespan of Algorithm 2 is

$$\begin{aligned}\sum_{k=1}^{q+1} \tau_k &\leq (4q+1)\mathrm{OPT} + \frac{2q}{m} \cdot \sum_{k=1}^{q} p(R^{(k-1)} \setminus R^{(k)}) \\ &\leq (4q+1)\mathrm{OPT} + \frac{2q}{m} \cdot p(J) \leq (4q+1)\mathrm{OPT} + 2q \cdot \mathrm{OPT} = \mathrm{O}(q) \cdot \mathrm{OPT}.\end{aligned}$$

Hence, the competitive ratio of Algorithm 2 is $\mathrm{O}(q) = \mathrm{O}(\log n / \log\log n)$. ◀

▶ Remark 9. Similar to Remark 7, suppose that we take an m-partition $(X_1, \ldots, X_m)$ of the jobs J that is an α-approximation of the problem (1) at line 6 in Algorithm 2. Then, the competitive ratio of Algorithm 2 becomes $\mathrm{O}\left(\alpha \cdot \frac{\log n}{\log\log n}\right)$-competitive for the preemptive sUETS problem. Moreover, by setting q to be the integer that satisfies $q^q \geq n^\alpha > (q-1)^{q-1}$, we can slightly improve the competitive ratio to $\mathrm{O}\left(\frac{\log n^\alpha}{\log\log n^\alpha}\right)$.

Algorithm 3 $O(\sqrt{m})$-competitive algorithm for the non-preemptive mUETS problem.

```
Let k = ⌊√m⌋;
Compute a k-partition (X_1, ..., X_k) of the jobs J that minimize max_{i∈[k]} c(X_i);
Assign batch X_i to ⌊m/k⌋ machines for each i ∈ [k];
```

3.2 Lower bounds

In this subsection, we show that the algorithms provided in Section 3.1 are asymptotically optimal with respect to both the number of machines and the number of jobs.

We first provide a lower bound for the non-preemptive setting.

▶ **Theorem 10.** *The competitive ratio of the non-preemptive sUETS problem is at least* $\Omega(m)$ *and* $\Omega(n^{1/3})$ *even for constant setup times.*

Proof. Let $n = m^3$ and $c(X) = 1$ for any non-empty batch $X \subseteq J$ (i.e., constant setup times). We will set the execution time 1 for at most m jobs (referred to as *heavy jobs*) and 0 for the other jobs (referred to as *light jobs*) depending on the behavior of a given online algorithm. It is not difficult to see that the optimal makespan is at most 2.

We fix an online algorithm. Let us consider its behavior when every job is light. Suppose that every batch assigned to machines consists of at most m jobs. Then, the algorithm assigns at least $n/m = m^2$ batches to the machines in total. Consequently, some machines must process at least m batches, which requires m setups. Thus, the makespan is at least m in this case, and we have done.

Conversely, suppose that the algorithm allocates a batch consisting of more than m jobs in the instance where every job is light. In another instance where m jobs in the first such batch are set to be heavy, the algorithm's behavior is the same until the batch is assigned. Thus, the algorithm assigns all the heavy jobs to one machine, and the makespan is at least m. Therefore, in both cases, the competitive ratio is at least $\Omega(m) = \Omega(n^{1/3})$. ◀

Next, we provide a lower bound for the preemptive setting.

▶ **Theorem 11.** *The competitive ratio of the preemptive sUETS problem is at least* $\Omega(m)$ *and* $\Omega(\log n / \log \log n)$ *even for constant setup times.*

4 Multiple Machines Batch Allocation

In this section, we explore the mUETS problem, where a batch can be allocated to multiple machines.

4.1 Algorithms

We first provide an $O(\sqrt{m})$-competitive algorithm for the non-preemptive mUETS problem. The algorithm computes a $\lfloor\sqrt{m}\rfloor$-partition of J that minimizes maximum setup time. Then, it allocates each batch to $\lfloor\sqrt{m}\rfloor$ machines. The algorithm is formally described in Algorithm 3.

▶ **Theorem 12.** *Algorithm 3 is* $O(\sqrt{m})$*-competitive for the non-preemptive mUETS problem.*

Algorithm 4 $\mathrm{O}\left(\frac{\log m}{\log\log m}\right)$-competitive algorithm for the preemptive mUETS problem.

```
Let q be an integer such that q^q ≥ m > (q−1)^{q−1};
Compute an m-partition X_1, ..., X_m of the jobs J that minimizes max_{i∈[m]} c(X_i);
Set k* ← ⌊log_q m⌋ + 1 and let I^(0) ← [m] be the set of unprocessed batch indices;
for k ← 1, 2, ..., k* do // phase k
    Assign batch X_i to q^{k−1} machines for each i ∈ I^(k−1);
    Continue processing the assignment until the number of uncompleted batches
      becomes ⌊m/q^k⌋. Once this condition is met, preempt all remaining
      uncompleted batches;
    Define I^(k) to be the set of uncompleted batch indices;
```

Proof. Let $(X_1^*, \dots, X_m^*)$ be an optimal schedule and let OPT be the optimal makespan. In addition, let $k = \lfloor\sqrt{m}\rfloor$ and $(X_1, \dots, X_k)$ be a k-partition of J that minimizes $\max_{i\in[k]} c(X_i)$. We analyze the algorithm that allocates each batch X_i to $\lfloor m/k\rfloor$ machines. The makespan of the algorithm is at most $\max_{i\in[k]}\left(c(X_i) + \frac{p(X_i)}{\lfloor m/k\rfloor} + \max_{j\in X_i} p_j\right)$. By the choice of $(X_1, \dots, X_k)$ and the subadditivity of the setup time c, we have

$$\begin{aligned}\max_{i\in[k]} c(X_i) &\le \max_{i\in[k]} c\left(\bigcup_{j=1}^{\lceil m/k\rceil} X^*_{\lceil m/k\rceil\cdot(i-1)+j}\right)\\ &\le \max_{i\in[k]} \sum_{j=1}^{\lceil m/k\rceil} c(X^*_{\lceil m/k\rceil\cdot(i-1)+j}) \le \left\lceil\frac{m}{k}\right\rceil\cdot \mathrm{OPT}, \qquad (2)\end{aligned}$$

where we denote $X_i^* = \emptyset$ for $i > k$. Therefore, by Lemma 1 and (2), the makespan is at most

$$\max_{i\in[k]}\left(c(X_i) + \frac{p(X_i)}{\lfloor m/k\rfloor} + \max_{j\in X_i} p_j\right) \le \left(\left\lceil\frac{m}{k}\right\rceil + \frac{m}{\lfloor m/k\rfloor} + 1\right)\cdot \mathrm{OPT} = \mathrm{O}(\sqrt{m})\cdot\mathrm{OPT},$$

which means that the competitive ratio of the algorithm is $\mathrm{O}(\sqrt{m})$. ◀

By combining Theorems 5 and 12, we can obtain the following corollary.

▶ **Corollary 13.** *There is an $\mathrm{O}(n^{1/4})$-competitive algorithm for the non-preemptive mUETS problem.*

Proof. If $m \le \sqrt{n}$, then the algorithm in Theorem 12 is $\mathrm{O}(\sqrt{m}) = \mathrm{O}(n^{1/4})$-competitive. On the other hand, if $m > \sqrt{n}$, the competitive ratio of Theorem 5 is $\mathrm{O}(\sqrt{n/m}) = \mathrm{O}(n^{1/4})$. Thus, in either case, there is an $\mathrm{O}(n^{1/4})$-competitive algorithm. ◀

▶ Remark 14. To account for the computational issue, suppose that we only have an m-partition $(X'_1, \dots, X'_m)$ of the jobs J that is an α-approximation for the problem (1). Let $k' = \lfloor\sqrt{\alpha m}\rfloor$. Then, the schedule that assigns batch $\bigcup_{j=1}^{\lceil m/k'\rceil} X'_{\lceil m/k'\rceil\cdot(i-1)+j}$ to $\lfloor m/k'\rfloor$ machines for each $i \in [k']$ is $\mathrm{O}(\sqrt{\alpha m})$-competitive for the non-preemptive mUETS problem. By combining this with Remark 7, we can obtain an $\mathrm{O}(\min\{\sqrt{\alpha}\cdot n^{1/4}, \sqrt{n}\})$-competitive algorithm for the non-preemptive mUETS problem.

For the preemptive mUETS problem, Algorithm 2 is also $\mathrm{O}(\log n/\log\log n)$-competitive, and this is asymptotically best possible as we will see in Theorem 18. Thus, even when we are allowed to assign a job batch to more than one machine, we cannot improve the competitive

ratio with respect to the number n of jobs. In contrast, we show that the competitive ratio with respect to the number m of machines can be exponentially improved by allowing batches to be assigned to multiple machines.

Let q be an integer such that $q^q > m \geq (q-1)^{q-1}$ and let k^* be $\lfloor \log_q m \rfloor + 1$. Here, $k^* \leq q$ and $q^{k^*-1} \leq m < q^{k^*}$. We have $q \geq 2$ from the assumption that $m \geq 2$.

Our algorithm for the preemptive mUETS is similar to Algorithm 2 for the preemptive sUETS. However, instead of splitting the uncompleted batch into several smaller batches of similar size, the algorithm increases the number of machines assigned to that batch. This ensures that the competitive ratio of the algorithm does not depend on the number of jobs n.

Our algorithm computes an m-partition $(X_1, \dots, X_m)$ of the jobs J that minimizes the maximum setup time $\max_{i \in [m]} c(X_i)$. In the first phase, each batch X_i is allocated to one machine. Then, each batch is processed until it is completed, or the number of uncompleted batches becomes less than or equal to $\lfloor m/q \rfloor$. In the kth phase ($k = 2, 3, \dots, k^*$), each uncompleted batch X_i is assigned to q^{k-1} machines, and it is processed until it is completed, or the number of uncompleted batches becomes $\lfloor m/q^k \rfloor$. If the number of uncompleted batches becomes smaller than $\lfloor m/q^k \rfloor$ because multiple batches are completed simultaneously, then break the tie arbitrarily and defer the rest batches to the next phase. Note that this allocation is feasible because $\lfloor m/q^{k-1} \rfloor \cdot q^{k-1} \leq m$. At the end of the k^*th phase, all the batches are completed since $\lfloor m/q^{k^*} \rfloor = 0$ by $k^* = \lfloor \log_q m \rfloor + 1 > \log_q m$. Our algorithm is formally described in Algorithm 4. It should be noted that this algorithm is based on a similar idea to that of [15, Algorithm 1].

▶ **Theorem 15.** *Algorithm 4 is* $\mathrm{O}\left(\frac{\log m}{\log \log m}\right)$*-competitive for the preemptive mUETS problem.*

Proof. Let OPT be the optimal makespan and let $p_{\max} = \max_{j \in J} p_j$. For each $k \in [k^*]$, let $S^{(k)} = I^{(k-1)} \setminus I^{(k)}$ be the set of indices of batches that have been completed in the kth phase. By the definition of the algorithm, we have

$$|S^{(k)}| = \left\lfloor \frac{m}{q^{k-1}} \right\rfloor - \left\lfloor \frac{m}{q^k} \right\rfloor \geq \left\lfloor \frac{m}{q^{k-1}} \right\rfloor - \frac{1}{q} \cdot \left\lfloor \frac{m}{q^{k-1}} \right\rfloor = \left(1 - \frac{1}{q}\right) \left\lfloor \frac{m}{q^{k-1}} \right\rfloor \geq \frac{1}{2} \cdot \left\lfloor \frac{m}{q^{k-1}} \right\rfloor.$$

Let τ_k be the time length of phase $k \in [k^*]$. We first bound it for $k \in [k^* - 1]$. As every batch in $S^{(k+1)}$ is not finished in phase k and the preemption takes time at most $\max_{i \in S^{(k)}} (c(X_i) + \max_{j \in X_i} p_j) \leq 2\mathrm{OPT}$, we have

$$\begin{aligned}
\tau_k &\leq \min_{i \in S^{(k+1)}} \left(c(X_i) + \frac{p(X_i)}{q^{k-1}} + p_{\max} \right) + 2\mathrm{OPT} \\
&\leq \frac{1}{|S^{(k+1)}|} \cdot \sum_{i \in S_{k+1}} \left(c(X_i) + \frac{p(X_i)}{q^{k-1}} + p_{\max} \right) + 2\mathrm{OPT} \\
&\leq \sum_{i \in S^{(k+1)}} \frac{1}{|S^{(k+1)}|} \cdot \frac{p(X_i)}{q^{k-1}} + 4\mathrm{OPT} \leq \frac{m/q^k}{\frac{1}{2} \cdot \lfloor m/q^k \rfloor} \cdot \sum_{i \in S^{(k+1)}} q \cdot \frac{p(X_i)}{m} + 4\mathrm{OPT}.
\end{aligned}$$

Note that $x/\lfloor x \rfloor \leq 2$ if $x \geq 1$ and $m/q^k \geq 1$ for $k \in [k^* - 1]$. Thus, we have

$$\tau_k \leq \sum_{i \in S^{(k+1)}} 4q \cdot \frac{p(X_i)}{m} + 4\mathrm{OPT} \tag{3}$$

for every $k \in [k^* - 1]$.

Next, we bound the time length τ_{k^*} of phase k^*. As the batch X_i for each $i \in S^{(k^*)}$ is processed by q^{k^*-1} machines, we have

$$\begin{aligned}\tau_{k^*} &\leq \max_{i \in S^{(k^*)}} \left(c(X_i) + \frac{p(X_i)}{q^{k^*-1}} + p_{\max} \right) \\ &\leq \frac{p(J)}{q^{k^*-1}} + 2\mathrm{OPT} < q \cdot \frac{p(J)}{m} + 2\mathrm{OPT} \leq (q+2)\mathrm{OPT}. \end{aligned} \tag{4}$$

Hence, by (3) and (4), the makespan of the algorithm is

$$\begin{aligned}\sum_{k=1}^{k^*} \tau_k &\leq \sum_{k=1}^{k^*-1} \left(\sum_{i \in S^{(k+1)}} 4q \cdot \frac{p(X_i)}{m} + 4\mathrm{OPT} \right) + (q+2)\mathrm{OPT} \\ &\leq 4q \cdot \frac{p(J)}{m} + 4(q-1)\mathrm{OPT} + (q+2)\mathrm{OPT} \leq (9q-2)\mathrm{OPT} = \mathrm{O}(q) \cdot \mathrm{OPT}. \end{aligned}$$

Therefore, the competitive ratio of Algorithm 4 is $\mathrm{O}(q) = \mathrm{O}(\log m / \log\log m)$. ◀

▶ Remark 16. Suppose that we take an m-partition $(X_1, \ldots, X_m)$ of the jobs J that is an α-approximation for the problem (1) at line 2 in Algorithm 2. Then, the competitive ratio of Algorithm 4 becomes $\mathrm{O}\big(\alpha \cdot \frac{\log m}{\log\log m}\big)$-competitive for the preemptive mUETS problem. Moreover, by setting q to be the integer that satisfies $q^q \geq m^\alpha > (q-1)^{q-1}$, we can slightly improve the competitive ratio to $\mathrm{O}\big(\frac{\log m^\alpha}{\log\log m^\alpha}\big)$.

4.2 Lower Bounds

In this subsection, we show the asymptotic optimality of our algorithms. We first provide lower bounds for the non-preemptive case.

▶ **Theorem 17.** *The competitive ratio of the non-preemptive mUETS is at least* $\Omega(\sqrt{m})$ *and* $\Omega(n^{1/4})$ *even for unweighted type-specific setup times.*

Next, we provide lower bounds for the preemptive case.

▶ **Theorem 18.** *The competitive ratio of the preemptive mUETS problem is at least* $\Omega(\log m / \log\log m)$ *and* $\Omega(\log n / \log\log n)$ *even for unweighted type-specific setup times.*

5 Conclusion and Discussion

In this paper, we introduced the UETS problem and studied it in terms of competitive analysis. We obtained tight bounds of the competitive ratio in each setting. In the following, we discuss the application of our results to variant settings.

When a batch $X \subseteq J$ is assigned to k machines, we have assumed that every machine incurs a setup time $c(X)$. However, for instance, in the production system example, unnecessary reassignment of attachments can be skipped. Additionally, if a machine processes multiple batches, the setup time incurred from the second or subsequent batch may be reduced. Our algorithmic results remain valid even for these cases as long as an optimal schedule assigns only one batch to each machine separately. This is because, for algorithms, a decrease in setup time only contributes to decreasing the makespan.

In the preemptive setting, we assumed that a job completed at the time of preemption does not need to be executed again. We can also consider a setting in which, if a batch process is preempted, the entire batch must be restarted from scratch. This means that even

completed jobs in the batch must be executed again. Algorithm 4 also works in this setting, and it is $O(\log m/\log\log m)$-competitive because its analysis does not use the information of which jobs are completed. However, Algorithm 2 may not achieve the competitive ratio of $O(\log n/\log\log n)$ as the proof does not work for this setting. Nevertheless, if we can obtain information on the execution time of completed jobs, we can reestablish an $O(\log n/\log\log n)$-competitive algorithm by processing the jobs that are completed once but need to be reprocessed between phases.

References

1 Ali Allahverdi, Chi To Ng, TC Edwin Cheng, and Mikhail Y Kovalyov. A survey of scheduling problems with setup times or costs. *European journal of operational research*, 187(3):985–1032, 2008. doi:10.1016/j.ejor.2006.06.060.

2 Norbert Ascheuer, Sven O. Krumke, and Jörg Rambau. Online Dial-a-Ride Problems: Minimizing the Completion Time. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, volume 1770, pages 639–650, 2000. doi:10.1007/3-540-46541-3_53.

3 Evripidis Bampis, Alexander V. Kononov, Giorgio Lucarelli, and Fanny Pascual. Non-clairvoyant makespan minimization scheduling with predictions. In Satoru Iwata and Naonori Kakimura, editors, *Proceedings of the International Symposium on Algorithms and Computation*, volume 283 of *LIPIcs*, pages 9:1–9:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ISAAC.2023.9.

4 Alexander Birx. *Competitive analysis of the online dial-a-ride problem*. PhD thesis, Technische Universität Darmstadt, 2020.

5 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.

6 Alexander Birx, Yann Disser, and Kevin Schewior. Improved bounds for open online dial-a-ride on the line. In *Proceedings of Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 145, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.21.

7 Vincenzo Bonifaci, Maarten Lipmann, and Leen Stougie. *Online multi-server dial-a-ride problems*. SPOR-Report : reports in statistics, probability and operations research. Technische Universiteit Eindhoven, 2006.

8 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Operations Research Forum*, 3(1):20, 2022. doi:10.1007/s43069-021-00101-z.

9 Srikrishnan Divakaran and Michael Saks. An online scheduling problem with job set-ups. *DIM ACS Technical Report: 2000*, 34, 2000.

10 Konstantinos Dogeas, Thomas Erlebach, and Ya-Chun Liang. Scheduling with Obligatory Tests. In *Proceedings of the Annual European Symposium on Algorithms*, pages 48:1–48:14, 2024. doi:10.4230/LIPIcs.ESA.2024.48.

11 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.

12 Greg N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation Algorithms for Some Routing Problems. *SIAM Journal on Computing*, 7(2):178–193, 1978. doi:10.1137/0207017.

13 Giorgio Gambosi and Gaia Nicosia. On-line scheduling with setup costs. *Information Processing Letters*, 73(1–2):61–68, 2000. doi:10.1016/S0020-0190(99)00152-0.

14 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, volume 174. W. H. Freeman, 1979.

15 Hiromichi Goko, Akitoshi Kawamura, Yasushi Kawase, Kazuhisa Makino, and Hanna Sumita. Online scheduling on identical machines with a metric state space. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, pages 32:1–32:21, 2022. doi:10.4230/LIPIcs.STACS.2022.32.

16 Ronald L. Graham. Bounds for Certain Multiprocessing Anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966. doi:10.1002/j.1538-7305.1966.tb01709.x.

17 Dan Gusfield. Bounds for naive multiple machine scheduling with release times and deadlines. *Journal of Algorithms*, 5(1):1–6, 1984. doi:10.1016/0196-6774(84)90035-X.

18 Leslie A. Hall and David B. Shmoys. Approximation schemes for constrained scheduling problems. In *Proceedings of Annual Symposium on Foundations of Computer Science*, pages 134–139, 1989. doi:10.1109/SFCS.1989.63468.

19 Dorit S Hochbaum and David B Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987. doi:10.1145/7531.7535.

20 Hongtao Hu, K. K.H. Ng, and Yichen Qin. Robust Parallel Machine Scheduling Problem with Uncertainties and Sequence-Dependent Setup Time. *Scientific Programming*, 2016. doi:10.1155/2016/5127253.

21 Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. *ACM Transactions on Parallel Computing*, 10(4):1–26, 2023. doi:10.1145/3593969.

22 Sven O. Krumke. *Online optimization: Competitive analysis and beyond*. PhD thesis, Technische Universität Berlin, 2001.

23 Maarten Lipmann, Xiwen Lu, Willem E. de Paepe, Rene A. Sitters, and Leen Stougie. On-Line Dial-a-Ride Problems Under a Restricted Information Model. *Algorithmica*, 40(4):319–329, 2004. doi:10.1007/s00453-004-1116-z.

24 Alexander Mäcker, Manuel Malatyali, Friedhelm Meyer auf der Heide, and Sören Riechers. Non-clairvoyant scheduling to minimize max flow time on a machine with setup times. In *Proceedings of the International Workshop on Approximation and Online Algorithms*, pages 207–222. Springer, 2017. doi:10.1007/978-3-319-89441-6_16.

25 David M Miller, Hui-Chuan Chen, Jessica Matson, and Qiang Liu. A hybrid genetic algorithm for the single machine scheduling problem. *Journal of Heuristics*, 5(4):437–454, 1999. doi:10.1023/A:1009684406579.

26 Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theoretical computer science*, 130(1):17–47, 1994. doi:10.1016/0304-3975(94)90151-1.

27 Kiyohito Nagano and Akihiro Kishimoto. Subadditive load balancing, 2019. doi:10.48550/arXiv.1908.09135.

28 Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 2008. doi:10.1007/978-3-031-05921-6.

29 Kirk Pruhs, Jirí Sgall, and Eric Torng. Online scheduling. In Joseph Y.-T. Leung, editor, *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004. doi:10.1201/9780203489802.CH15.

30 Vinod K Sahney. Single-server, two-machine sequencing with switching time. *Operations Research*, 20(1):24–36, 1972. doi:10.1287/opre.20.1.24.

31 JiŘí Sgall. *On-line scheduling*, pages 196–231. Springer Berlin Heidelberg, 1998.

32 David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24(6):1313–1331, 1995. doi:10.1137/S0097539793248317.

33 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.

The Parameterized Landscape of Labeled Graph Contractions

Manuel Lafond
Department of Computer Science, University of Sherbrooke, Canada

Bertrand Marchand
Department of Computer Science, University of Sherbrooke, Canada

Abstract

In this work, we study the problem of computing a *maximum common contraction* of two vertex-labeled graphs, i.e. how to make them identical by contracting as little edges as possible in the two graphs. We study the problem from a parameterized complexity point of view, using parameters such as the maximum degree, the degeneracy, the clique-width or treewidth of the input graphs as well as the number of allowed contractions. We put this complexity in perspective with that of the *labeled contractibility problem*, i.e determining whether a labeled graph is a contraction of another. Surprisingly, our results indicate very little difference between these problems in terms of parameterized complexity status. We only prove their status to differ when parameterizing by both the degeneracy and the number of allowed contractions, showing W[1]-hardness of the maximum common contraction problem in this case, whereas the contractibility problem is FPT.

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

Keywords and phrases Parameterized complexity - contractions - labels - widths

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.42

Related Version *Full Version*: `https://arxiv.org/abs/2502.16096`

1 Introduction

Graphs are used as an abstract model in a wide variety of applicative fields. For instance, in bioinformatics, they can represent structured RNAs [40, 45, 36, 8], evolutionary histories [26], or interaction networks [31]. In several applications, a common computational task is the quantitative *comparison* of graphs, in order to underline a common structure. For example, the comparison of *phylogenetic networks* (graphs representing evolutionary histories) has a rich line of work in evolutionary bioinformatics [13, 37, 26, 11, 12, 22, 33]. Likewise, the *network alignment problem* has been formulated to compare biological interaction networks [17, 19, 28]. As for the comparison of molecules, it has motivated work on the *maximum common subgraph* problem [1, 2, 18]. The *graph edit distance* [23] was also studied for this purpose, especially in a context of image processing and machine learning [43, 44].

The philosophy of the graph edit distance is to compare two input graphs by counting the number of "edit operations" required to transform one graph into the other. This edition process can amount to "reducing" the first graph to a "maximum common reduced graph" before applying "augmenting" operations to get the second graph. For instance, the computation of the graph edit distance, given its use of the vertex deletions and insertions, amounts to the computation of a maximum common subgraph under a special cost function [10]. Following this philosophy, the *edge contraction* operation has been used to define one of the most widely-used distance on evolutionary trees, called the Robinson-Foulds distance [41], whose computation finds a *maximum common contraction* of the two trees. Recently, this approach was generalized to the comparison of phylogenetic networks [37], which are directed graphs representing evolution. However, as far as the authors know, the

© Manuel Lafond and Bertrand Marchand;
licensed under Creative Commons License CC-BY 4.0

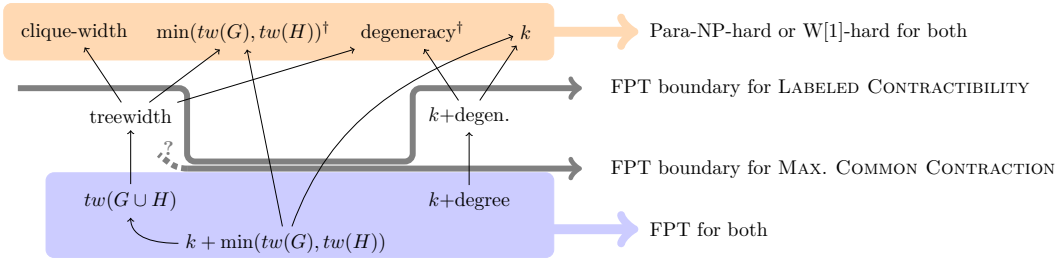


Figure 1 Illustration of our results on the compared parameterized complexity of Labeled Contractibility and Maximum Labeled Common Contraction. Results marked with a † are derived from the litterature [37, 9]. The clique-width, degeneracy and treewidth are to be understood as the maximum value of these parameters on the two input graphs. An arrow from (1) to (2) indicates that bounding (1) implies bounding (2).

problem of computing a maximum common contraction between two undirected graphs has received little to no attention from an algorithmic perspective. There are several deterring factors that may explain this. First, if the graphs are unlabeled, then determining whether no contractions are required is equivalent to checking whether they are isomorphic, a notoriously difficult problem to tackle. To make matters worse, even if the input graphs are two (unlabeled) trees, determining whether one is a contraction of another is NP-hard [38]. On the other hand, this hardness does not apply to *labeled* trees, where each vertex has a unique identifier, since polynomial-time algorithms exist (even just *leaf-labeled* trees as in [41]).

In this article, we study the Maximum Common Labeled Contraction, i.e. the problem of computing a maximum-size common contraction of two fully-labeled graphs, from the perspective of *parameterized complexity*. The input graphs are uniquely labeled (two distinct vertices of a graph have distinct labels), but each graph may have labels not present in the other. We use structural parameterizations, such as the treewidth or clique-width of the input graphs, but also the maximum degree, degeneracy or number of contractions. We also study the *contractibility* problem (given two labeled graphs, is one a contraction of the other ?) and compare the parameterized complexity aspects of both problems. Our results are summarized on Figure 1, and outline little difference in complexity between both problems. We fully establish the location of the barrier between FPT and W[1]-hardness/para-NP-hardness for our chosen parameters – with the notable exception of the maximum common contraction problem when both graphs have bounded treewidth, which remains open. We see that the two problems behave similarly, apart from the parameter $k + \delta$, with δ the degeneracy.

This paper is organized as follows. After an overview of related works, preliminary notions and results are given in Section 2. Then, Section 3 proves our results regarding the contractibility problem, and Section 4 those on the maximum common contraction problem. Due to space constraints, some proofs have been deferred to the Appendix, or to the full online version (`https://arxiv.org/abs/2502.16096`).

Related works. A rich related line of work is the study of the H-Contractibility problem on undirected, unlabeled graphs. It consists in deciding whether an input graph G can be transformed into a graph isomorphic to H using only edge contractions. In a seminal article [9], H-contractibility was proven NP-hard with $H = P_4$, the path on 4 vertices. Follow-up works [34, 35] gave characterizations of the graphs H such that H-Contractibility is NP-hard. On the positive side, H-Contractibility was proven polynomial-time solvable if

G is chordal [7, 5], or if H is planar (but still fixed) [29]. A polynomial algorithm for H of bounded degree and G of bounded treewidth was also given in [38], in addition to the proof that deciding if a tree is a contraction of another is NP-hard (already mentioned above). Variations of the problem, for instance deciding whether k contractions are enough to make a graph fall into a given *class*, have also been studied, including under the parameterized complexity point of view. These include the contractibility into *grid graphs* [42], *bipartite graphs* [24], *paths/trees* [25, 30] or graphs of *bounded degree* [6].

As mentioned in the introduction, maximum common contractions were studied on *phylogenetic networks* [37], which are directed acyclic graphs with a single root and labeled leaves. In [37], the NP-hardness of the problem is proven, and a polynomial-time tractable sub-case is identified, but the parameterized complexity of the problem was not explored. Nonetheless, hardness proofs from [37] as well as [9] can be easily adapted to fully-labeled undirected graphs. It gives the following starting points: contractibility into a P_4 graph is NP-hard on labeled graphs [9], the maximum common contraction problem on labeled graphs is NP-hard if both input graphs have constant degree (and thus also constant degeneracy) [37, Theorem 8], and the labeled contractibility problem is NP-hard on graphs of bounded degeneracy (also in [37, Theorem 8], as although the reduction is to the maximum common contraction problem, it falls back to a contractibility instance of bounded degeneracy).

2 Preliminary notions

For an integer n, we may use the notation $[n] = \{1, 2, \ldots, n\}$. We denote the vertex set and edge set of a graph G by $V(G)$ and $E(G)$, respectively. The subgraph of G induced by a subset of vertices $X \subseteq V(G)$ is denoted $G[X]$. We write $G - X$ for the graph $G[V(G) \setminus X]$, and if $X = \{x\}$ has a single element we may write $G - x$. For a vertex u, $N_G(u)$ is the set of neighbors of u in G, and $N_G[u] = N_G(u) \cup \{u\}$. We say that two disjoint subsets $X, Y \subseteq V(G)$ are *adjacent* if there exists an edge between an element of X and an element of Y. The maximum degree of G is denoted $\Delta(G)$. The *degeneracy* of G is $\delta(G)$, which is the smallest integer such that every subgraph of G has a vertex of degree at most $\delta(G)$.

Given two graphs G and H, we write $G = H$ if and only if $V(G) = V(H)$ and $E(G) = E(H)$. This differs from the more common notion of isomorphism: the vertex sets of the two graphs must consist of precisely the same elements, and edges must connect the same pairs of vertices. We can thus view the vertices of G and H as uniquely labeled, and equality requires that vertices with the same labels share the same edges in both graphs. However, we prefer to avoid labeling functions, and instead compare the vertices and edges directly. Note that sets of labels may both intersect ($V(G) \cap V(H) \neq \emptyset$, in fact no common contraction is possible otherwise) and contain elements not present in the other graph ($V(G) \Delta V(H) \neq \emptyset$).

Labeled contractions of graphs. Given a graph G and an edge uv of G, the *labeled contraction* (u, v) is an operation that transforms G as follows: (1) add an edge between u and every vertex of $N_G(v) \setminus N_G[u]$; (2) remove v and all its incident edges. The graph obtained from G after the labeled contraction (u, v) is denoted $G/(u, v)$. Note that because the sets of vertices of our graphs matter, contracting (u, v) is different than contracting (v, u), that is, $G/(u, v) \neq G/(v, u)$ (as these two graphs remove a different vertex). Do note that $G/(u, v)$ and $G/(v, u)$ are isomorphic in the traditional sense.

A *labeled contraction sequence* S on G is a list of vertex pairs $S = ((u_1, v_1), \ldots, (u_k, v_k))$ such that, for each $i \in \{0, 1, \ldots, k-1\}$, if G_i is the graph obtained after the application of the first i labeled contractions (with $G_0 = G$), then $u_{i+1}v_{i+1}$ is an edge of G_i and G_{i+1} is

obtained by applying the labeled contraction (u_{i+1}, v_{i+1}) on G_i. We denote the resulting graph G_k as G/S (if S cannot be applied on G, then G/S is undefined). The number of pairs in S is denoted $|S|$, which here is k.

We say that a graph H is a *labeled contraction of* G if there exists a labeled contraction sequence S such that $G/S = H$. Again, we emphasize that we require equality here. If a graph M is a labeled contraction of both G and H, it is called a *common labeled contraction*. A *maximum common labeled contraction* of G and H is a common labeled contraction with a maximum number of vertices. Equivalently, it is the result of applying a minimum number of labeled contractions on G and H. Formally, we are interested in the following problem.

Maximum Common Labeled Contraction
Input: two graphs G and H, integer k.
Question: Are there labeled contraction sequences S_1 and S_2 such that $G/S_1 = H/S_2$ and $|S_1| + |S_2| \leq k$?

This can be viewed as the computation of a graph distance, since $|S_1| + |S_2|$ give a minimum number of contractions and *expansions* required to transform G into H, where expansions are the reverse of contractions. Note also that the problem is equivalent to asking, given G and H whether there is a common contraction M such that $2|V(M)| \geq |V(G)| + |V(H)| - k$.

An "easier" variant is when the common contraction must be the smaller input graph.

Labeled Contractibility
Input: two graphs G and H with $V(H) \subseteq V(G)$.
Question: is H a labeled contraction of G ?

▶ Remark 1. In the remainder, all contractions are labeled, so we may simply write "contraction" instead of "labeled contraction".

Maximum Common Labeled Contraction generalizes Labeled Contractibility. Indeed, H is a contraction of G if and only if G and H have a common contraction of size at least $|V(H)|$, or using at most $|V(G) \setminus V(H)|$ contractions. Therefore, our hardness results that apply to Labeled Contractibility transfer to Maximum Common Labeled Contraction. Note that in either problem, we do not require the input graphs to be connected. In Labeled Contractibility, the connected components of G can be matched uniquely to those of H using common vertices. In Maximum Common Labeled Contraction, one could proceed as follows: for each connected component C_1 from G and C_2 from H, compute a maximum common contraction between $G[C_1]$ and $G[C_2]$, and assign a weight on $\{C_1, C_2\}$ equal to the size of the common contraction ($-\infty$ if none exists). Then, find a maximum-weight perfect matching in the resulting edge-weighted bipartite graph. Hence, disconnected graphs only affect the complexity by a potential polynomial factor.

Witness structures. When applying a contraction sequence $G/S = H$, each vertex that gets removed from G can be seen as "subsumed" by exactly one of the remaining vertices in H. It is common to replace the result of a contraction sequence with the vertex partition of $V(G)$ that groups subsets of vertices that end up in the same vertex. Such a partition *witnesses* the existence of a contraction sequence from G to H, formalized as follows.

Let G, H be two graphs with $V(H) \subseteq V(G)$. Let $\mathcal{W} = \{W_1, \ldots, W_{|V(H)|}\}$ be a partition of $V(G)$ with $|V(H)|$ non-empty sets. For $u \in V(G)$, we denote by $\mathcal{W}(u) \in \mathcal{W}$ the set of $\mathcal{W}$ that contains u. We say that $\mathcal{W}$ is a *witness structure of* G *into* H if all the following conditions are satisfied (see Figure 2):

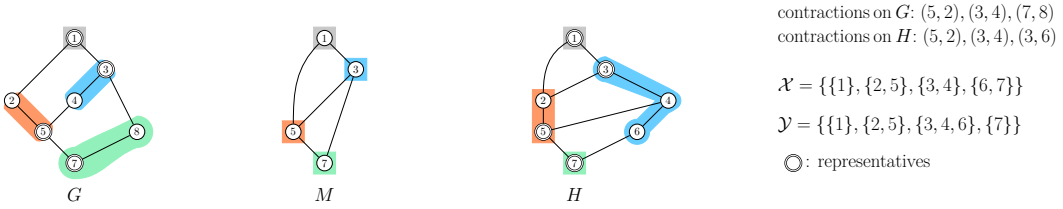


Figure 2 Example of two graphs G, H and a common contraction M. On the graphs, the shaded areas represent the witness sets. The witness structures of respectively G and H into M are called $\mathcal{X}$ and $\mathcal{Y}$, and are given on the right. Note that $V(G) \neq V(H)$, as allowed by our problem statement.

- For every $W_i \in \mathcal{W}$, the induced subgraph $G[W_i]$ is connected.
- Every $W_i \in \mathcal{W}$ contains exactly one vertex of H. This vertex is called the *representative* of W_i (in H).
- For every distinct $u, v \in V(H)$, $uv \in E(H)$ if and only if $\mathcal{W}(u)$ and $\mathcal{W}(v)$ are adjacent.

We note that there is a natural bijection between $\mathcal{W}$ and $V(H)$ formed by the representatives. The following equivalence is well-known in the case of unlabeled graphs [9, 34], and is easily seen to hold in labeled graphs. An example of two graphs, a common contraction, and the corresponding witness structures is given on Figure 2.

▶ **Observation 2.** *Let G, H be graphs. Then H is a contraction of G if and only if there exists a witness structure of G into H.*

Note that each contraction reduces the number of vertices by exactly one, and so if $\mathcal{W}$ is a witness structure of G into H, then the number of contractions needed is $\sum_{W_i \in \mathcal{W}}(|W_i| - 1)$. To finish, we prove that contractions within a witness set can essentially be done in any order, as long as the representative stays. The idea is that if we apply any contraction within a witness set first, we can update the affected witness set and get a witness structure for the modified graph.

▶ **Observation 3.** *Let $\mathcal{W}$ be a witness structure of G and H, and let $uv \in E(G)$ such that u, v are in the same witness set of $\mathcal{W}$. If $v \notin V(H)$, then H is a contraction of $G/(u, v)$.*

3 Labeled Contractibility

We begin by describing where the hardness barrier resides for the contractibility problem. We focus on difficult parameters first, then study those that lead to FPT algorithms.

3.1 W[1]-hardness in parameter k

We first consider the parameter k, which is the number of contractions needed to transform G into H, or equivalently $k = |V(G)| - |V(H)|$. We reduce from the well-known Multicolored Clique problem, a W[1]-hard problem [20] which we recall.

Multicolored Clique
Input: a graph $G_C = (V, E)$ along with a partition $\{V_1, \ldots, V_k\}$ of V into k sets, with each V_i called a color class.
Output: does G_C contains a *multicolored clique*, i.e., a clique that contains exactly one vertex per color class?

Let $G_C = (V, E)$ be an instance of MultiColored Clique, with $V = V_1 \cup \ldots \cup V_k$ partitioned into k color classes. We assume that for every distinct $i, j \in [k]$, each vertex $v \in V_i$ has at least one non-neighbor in V_j. This is without loss of generality, as we can add an isolated vertex to each V_i without changing the answer to the instance.

Let us construct two graphs G and H from G_C. We start with the simpler H, which is obtained by copying G_C, and making each V_i a clique. More specifically

$$V(H) = V(G_C) \qquad E(H) = E(G_C) \cup \bigcup_{i \in [k]} \{uv : u \in V_i, v \in V_i, u \neq v\}.$$

To obtain G, start with a copy of H, then add k new vertices $t_1, \ldots, t_k$, which are not present in H. Then, make $\{t_1, \ldots, t_k\}$ a clique, and for each $i \in [k]$, add every possible edge between t_i and the vertices V_i in G. This concludes the construction of G and H.

▶ **Theorem 4.** *The Labeled Contractibility problem is W[1]-hard in parameter k, which is the number of contractions needed to transform one input graph into the other.*

Proof sketch. Considering the above construction, if G_C has a multicolored clique $u_1, \ldots, u_k$, then we can apply the set of contractions (u_i, t_i), for $i \in [k]$. This can only add edges between distinct u_i, u_j vertices, but those edges were already present in G (and in H) since the u_i's form a clique. In other words, this just gets rid of the t_i vertices without adding any new extra edge, and thus the resulting graph is identical to H.

Conversely, if G can be contracted into H, then we can argue that each t_i must be contracted with some $u_i \in V_i$ (contracting it with a $u_j \in V_j, j \neq i$ would add all edges between u_j and V_i, which are not in H). For $i \neq j$, if we contract (u_i, t_i) then (u_j, t_j), the edge $u_i u_j$ will be created if not already present. This must be prevented to reach H, so $u_i u_j$ must already be in G, so the u_i's contracted with the t_i's must form a clique. ◀

We observe that this problem is easily seen to be in XP when parameterized by k. Indeed, given graphs G and H, we can just try every sequence of k contractions in G that suppress a vertex of $V(G) \setminus V(H)$. There are $O(n^2)$ choices and we make at most k contractions, resulting in complexity of the form $O(n^{2k})$. Note also that since our reduction is from Multicolored Clique and it preserves the parameter k exactly, it also implies [16, Corollary 14.23] no $f(k) \cdot n^{o(k)}$ algorithm for Labeled Contractibility under the Exponential-Time Hypothesis (ETH). Therefore, the aforementioned XP algorithm is essentially optimal under ETH.

3.2 para-NP-hardness for clique-width

We recall the definition of clique-width. Each vertex of a graph can be assigned a color, and the clique-width is the minimum number of colors required to build a graph using a sequence of the following operations:

- Creation of a new vertex with color i.
- Disjoint union of two (vertex-colored) graphs.
- Adding all possible edges between vertices colored i and vertices colored j, $i \neq j$.
- Recoloring all vertices with color i to color j, where $i \neq j$.

The clique-width of a graph G is denoted $\mathrm{cw}(G)$.

We describe a reduction from Unary Bin Packing, and more specifically the variant in which we ask that every bin is filled exactly up to its capacity, and the input integers are encoded in unary (an integer a takes a bits in the input). We call it Unary Perfect Bin Packing.

Unary Perfect Bin Packing
Input: Integers $a_0, \ldots, a_{n-1}$ (item sizes) encoded in unary, bin capacity C, and number of bins k, such that $\sum_{x=0}^{n-1} a_x = Ck$.
Output: Is there an assignment $\phi : [0, n-1] \to [0, k-1]$ of each item to a bin such that $\forall i \in [0, k-1]$, $\sum_{j\,:\,\phi(j)=i} a_j = C$?

Note that item size and bin subscripts are indexed at 0 to simplify some calculations later on. We know from [27] that Unary Bin Packing is NP-hard and W[1]-hard in the number of bins k. It turns out that Unary Perfect Bin Packing is also W[1]-hard in parameter k, and also NP-hard [32]. We use this problem to show that Labeled Contractibility is NP-hard even when both input graphs have clique-width at most 4.

Reduction description. Consider an input to Unary Perfect Bin Packing, with item sizes $a_0, \ldots, a_{n-1}$, bin capacity C, and number of bins k. An illustration of the reduction is given in Figure 3. From an instance $a_0, \ldots, a_{n-1}, C, k$ of Unary Perfect Bin Packing, we construct graphs G, H as follows. Both G and H contain a set of vertices $B = \{b_0, \ldots, b_{k-1}\}$, each corresponding to a bin, connected as a clique in both graphs. Again in both G and H, there is also a set of Ck vertices $D = \{d_0, \ldots, d_{Ck-1}\}$, connected as a clique in both graphs. In H, each vertex $d_j \in D$ has exactly one neighbor in B, which is b_i such that $\lfloor j/C \rfloor = i$ (thus each b_i has exactly C neighbors in D). This finishes the construction of H. In G, there also vertices $T = \{t_0, \ldots, t_{n-1}\}$ not present in H, which represent the items. The set T forms an independent set. The vertex sets T and B form a bi-clique, so every possible edge between the two sets is present. For each $t_x \in T$, we also include in G a set of vertices $A_x = \{\alpha_x^0, \ldots, \alpha_x^{a_x-1}\}$ containing a_x elements, each having t_x as a neighbor. We denote the set $A_0 \cup \ldots \cup A_{n-1}$ as A (see Figure 3), which forms an independent set and is absent from H. The set A forms a bi-clique with the vertices D in G.

For H to be a contraction of G, there must be a way for each b_i to acquire the neighbors specified by H. In particular, each b_i is connected in H to exactly C vertices of the set D (recall that C is the bin capacity). Specifically, in H there is an edge (b_i, d_j) for any $i \in [0, k-1]$ and $j \in [0, Ck-1]$ such that $\lfloor j/C \rfloor = i$. To do this, each b_i must be contracted with some t_x vertices, making b_i acquire a certain number of neighbors in A. Each such neighbor must then disappear, and to do that there must be a contraction with an element of D. Since $|A| = |D|$, each element of A is contracted with a single element of D, so b_i must have acquired exactly C neighbors into A to start with. We proceed to the proof of this idea.

▶ **Theorem 5.** *The Labeled Contractibility problem is NP-hard, even if the input graphs have clique-width at most* 4.

Proof sketch. The graph G actually has clique-width 3. We can first create the $G[T \cup A]$ subgraph, a forest of star trees, using two colors. We can create the B clique and then connect it to all of T using a third color, recoloring B to the color of T, then adding D and connecting it to A reusing that third color. As for H, we can incorporate each subgraph $H[\{b_i\} \cup (N_H(b_i) \cap D)]$ one at a time. We assume we already have the $b_1, \ldots, b_{i-1}$ vertices using one color, and their neighbors in D using another, and we add b_i using a new color and its neighbors in D using another new color, which allows connecting them to the previous B and D vertices (and then we recolor b_i and its neighbors in D).

To see that the generated instance is equivalent, suppose that ϕ assigns items to bins perfectly. If bin i has items $a_{i_1}, \ldots, a_{i_q}$, in G we contract $(b_i, t_{i_1}), \ldots, (b_i, t_{i_q})$. Since each t_{i_j} has a_{i_j} neighbors in A, this gives b_i exactly C neighbors from A, say $v_0, \ldots, v_{C-1}$. Then to ensure that b_i has the correct neighbors from D, we contract $(v_0, d_{iC}), (v_1, d_{iC+1}), \ldots, (v_{C-1}, d_{(i+1)C-1})$. This results in the graph H.

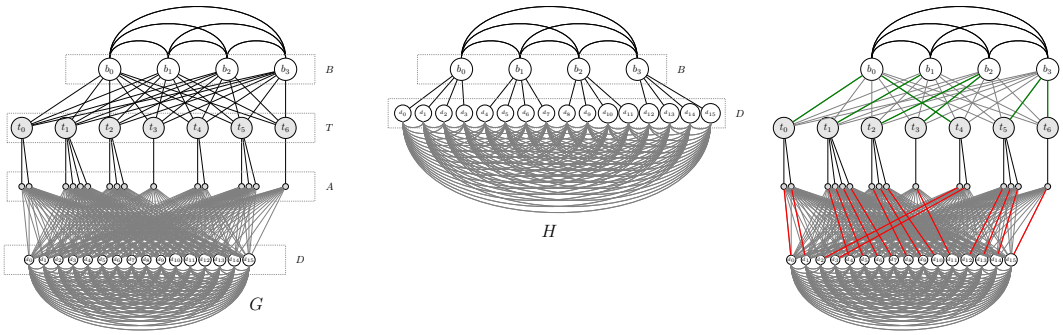


Figure 3 Illustration of the reduction from Unary Perfect Bin Packing to fully-labeled contractibility, on an instance. This instance of Unary Perfect Bin Packing has 4 bins (represented by the vertices labelled $b_0, \ldots, b_3$), 7 items (represented by the vertices $t_0, \ldots t_6$), the items have size respectively $2, 4, 3, 1, 2, 3$ and 1 as represented by their number of neighbors (without counting b_i), and the size of each bin is 4 (as represented by the number of neighbors each b_i vertex has in H). On the right, the contraction of the highlighted edges yields H. As we discuss in the proof, the green edges then encode a solution to Unary Perfect Bin Packing.

Conversely, if we can contract G into H, then we can argue in terms of witness sets: $b_i \in B$ cannot be in the same witness set as $v \in A$, since b_i would become adjacent to all of D; a $t_i \in T$ cannot be with a $d_j \in D$, as d_j would become adjacent to all of B. This implies that each $t_i \in T$ is in the witness set of an element of B, and each $v \in A$ with an element of D. Moreover, each $d_j \in D$ *must* have some $v \in A$ in its witness set, as otherwise d_j becomes impossible to connect with any $b_i \in B$. Hence the witness sets containing vertices of A and D form a perfect matching. This then makes it easy to show that if some b_i has $t_{i_1}, \ldots, t_{i_q}$ in its witness set, then these members of T must have had exactly $|N_H(b_i)| = C$ neighbors in D, establishing the correspondence with perfect bin assignments. ◀

3.3 An FPT algorithm in k and degeneracy

Let us start with a definition of the degeneracy. Given a total order σ of the vertices of a graph G, we denote by $N^G_{\geq,\sigma}(u)$ the neighbors of u that are after it according to σ. We define the degeneracy $\delta(G)$ of a graph as: $\delta(G) = \min_\sigma \max_{u \in V(G)} |N^G_{\geq,\sigma}(u)|$.

We call $N^G_{\geq,\sigma}(u)$ the *remaining neighborhood* of u according to σ. This denomination is to be understood in an interpretation of degeneracy as an elimination process.

▶ **Lemma 6.** *If H is obtained from G by $\leq k$ contractions, then $\delta(H) \leq \delta(G) + k$*

Proof sketch. We prove that a single contraction of an edge u, v in a graph only increases the degeneracy by at most one. Given an elimination ordering σ optimal for $\delta(G)$, we modify it to place the merged vertex (whose degree might have doubled compared to u, v) as late into σ as possible. By placing it after all of its neighbors, we make its remaining neighborhood empty, while augmenting it by at most 1 vertex for other vertices. ◀

We describe a branching algorithm for Labeled Contractibility that tries a bounded number of contractions at each recursion. On a recursive call that receives graphs G, H and integer k, the enumeration proceeds as described below. This algorithm is also given in pseudo-code in the appendix.

- If $G = H$ we return true. If $k < 0$, or $|V(G) \setminus V(H)| > k$, we return false.
- Otherwise, we take a vertex u of G of minimum degree. Assuming that G is the result of at most k contractions, by Lemma 6, the degree of u is at most $\delta(G) + k$.
- If u is in G but not in H, we know that u must be removed by a contraction at some point. So we branch into at most $\delta(G) + k$ ways of contracting u with one of its neighbors (which is safe by Observation 3 on the order of contractions), and decrease k by 1.
- Otherwise, u is in both G and H and must be kept. If $N_G(u) = N_H(u)$, then we may simply ignore u from now on, as further contractions on G that remove vertices outside of H will not change the neighborhood of u. We thus remove u from both graphs (note that this step may fail on the maximum common contraction problem).
- If $N_G(u) \subsetneq N_H(u)$, i.e. u has a neighbor v in H, but $v \notin N_G(u)$, then we know that some contraction must affect u or v, as otherwise the edge uv will never be created. We branch into all ways of contracting u with one of its neighbors not in H, or v with one of its neighbors not in H. (if there is no such neighbor for either of them, we return false). This branches into at most $2k$ cases, because $|V(G) \setminus V(H)| \leq k$.
- If u has a neighbor $v \in V(H)$ in G, but $v \notin N_H(u)$ (i.e. both u, v must be kept, but they share an unwanted edge) then we return false.
- Finally, if none of the above holds, then $N_H(u) \subseteq N_G(u)$, and u has a neighbor v in G with $v \notin V(H)$. This vertex v must be contracted into one of its neighbors. It cannot be contracted into a vertex of $V(H) \setminus N_H(u)$ (as $N_H(u)$ is already complete), it may therefore only be contracted into a vertex in $V(G) \setminus V(H)$(which contains $\leq k - 1$ elements when not counting v) or $N_G(u) \cup \{u\}$ (which contains $\leq \delta(G) + k + 1$ elements) We branch over these $\delta(G) + 2k$ possibilities.

▶ **Theorem 7.** *Labeled Contractibility can be solved in time* $O\left((\delta(G) + 2k)^k \cdot (n + m)\right)$, *where* k *is the number of contractions,* $n = |V(G)|$, *and* $m = |E(G)|$.

Proof sketch. The algorithm is a bounded search tree with two kinds of recursive calls: non-branching calls (when $N_H(u) = N_G(u)$) and branching calls involving a contraction. In the latter, we make at most $\delta(G) + 2k$ recursive calls that each decrease k by 1. ◀

4 Maximum Common Labeled Contractions

We now turn to maximum common contractions. We start with our hardness result on graphs of small degeneracy, contrasting with the positive result for the contractibility problem.

4.1 W[1]-hardness in 4-degenerate graphs

We show that Maximum Common Labeled Contraction is W[1]-hard in parameter k even on graphs of degeneracy 4. Therefore, there is little hope for an FPT algorithm in parameter $k + \delta(G)$. To give credit where is due, the reduction was initially inspired by [39] for the hardness of domination problems on graphs of bounded degeneracy, although the adaptations required for our proof makes it largely different. We will need the following lemma, which is a generalization of Observation 3.

▶ **Lemma 8.** *Let* H *be a contraction of a graph* G, *and let* $\mathcal{W}$ *be the witness structure of* G *into* H. *Let* $R \subseteq E(G)$ *be a matching of* G *such that, for every edge* $xy \in R$, *the vertices* x *and* y *are in the same witness set of* $\mathcal{W}$.

Let G' *be the graph obtained after contracting every edge of* R, *making sure not to remove a representative vertex of* $\mathcal{W}$. *Then* H *is a contraction of* G'.

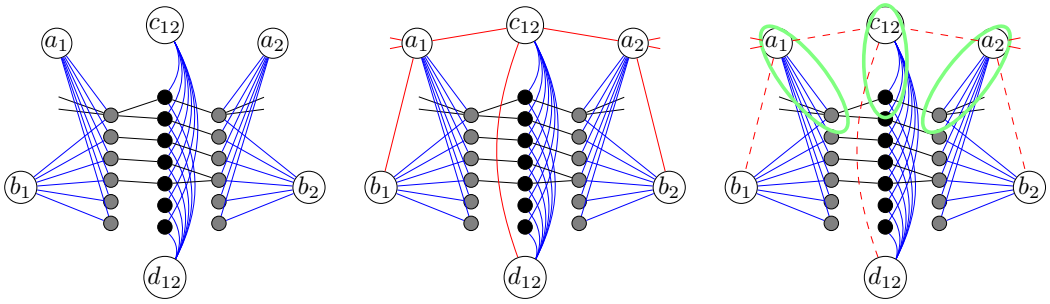


Figure 4 Left: an illustration of the vertices of and edges of G relating V_1 and V_2. Gray vertices represent V_1' and V_2', and black vertices represent Z_{12}' (the bottom two vertices in each group represent extra vertices). Some edges are colored blue only for clarity. Middle: the same subgraph but in H. The red edges are those that are unique to H. Note that a_1 has other neighbors $c_{13}, \ldots, c_{1k}$, similar with a_2, but all the neighbors of b_1, b_2, c_{12}, d_{12} are shown. Right: witness sets corresponding to contractions in a solution for the forward direction (assume witness sets $\{a_1, u_1\}, \{a_2, u_2\}, \{c_{12}, z_e\}$ such that $e = u_i u_j$). Whether the red edges are present or not, this will result in the same graph.

We can now describe our reduction. Let $G_C = (V, E)$ be an instance of Multicolored Clique with $V = V_1 \cup V_2 \cup \ldots \cup V_k$ partitioned into k color classes. Construct an instance G, H of Maximum Common Labeled Contraction, starting with the description of G. The reader may refer to Figure 4. First, for each $i \in [k]$, define V_i' as a set of vertices obtained by starting with V_i, then adding $4k^3$ new vertices. We refer to elements of $V_i' \setminus V_i$ as *extra vertices*. Add to G the set of vertices $V_1' \cup \ldots \cup V_k'$. Then for each distinct $i, j \in [k]$, and for each edge edge $e = uv$ of G_C with $u \in V_i$ and $v \in V_j$, create a new vertex z_e and add the path $u - z_e - v$ (or said differently, add the edge uv to G and subdivide it, which creates z_e). We may write $z_e = z_{uv} = z_{vu}$. Denote by Z_{ij} the set of z_e vertices created at this step (we consider that $Z_{ij} = Z_{ji}$). Next, obtain the set of vertices $Z_{ij}' = Z_{ji}'$ by adding $4k^3$ new vertices to Z_{ij} (these new vertices are also called extra vertices).

Then for each $i \in [k]$, add two new vertices a_i and b_i, and make a_i and b_i adjacent to every vertex of V_i' in G. Note that a_i and b_i are not adjacent, and that the extra vertices of V_i' are only adjacent to a_i and b_i. Finally, for each distinct $i, j \in [k]$, add two vertices c_{ij} and d_{ij}, non-adjacent. We define $c_{ji} = c_{ij}$ and $d_{ji} = d_{ij}$. Make c_{ij} and d_{ij} adjacent to every vertex in Z_{ij}'. In particular, extra vertices of Z_{ij}' are only adjacent to c_{ij}, d_{ij}. This completes the construction of G. To construct H, start with a copy of G, then add the following edges: $a_i b_i$ for each $i \in [k]$; $a_i c_{ij}$, $a_j c_{ij}$ and $c_{ij} d_{ij}$ for each distinct $i, j \in [k]$; Finally, define $K := 2(k + \binom{k}{2})$. This concludes the construction.

▶ **Theorem 9.** *The Maximum Common Labeled Contraction is W[1]-hard in parameter k, the total number of contractions needed to achieve a common contraction, even if both input graphs have degeneracy* 4.

Proof sketch. The graphs G, H have degeneracy at most 4 since we can have an elimination order that removes every Z_{ij}' first (vertices of degree 4), then the V_i' vertices (remaining degree 2), followed by the c_{ij}, d_{ij} vertices and then the a_i and b_i vertices.

Next, if G_C has a multicolored clique $u_1, \ldots, u_k$, in either graph G or H, we contract each (a_i, u_i) pair, and each $(c_{ij}, z_{u_i u_j})$ pair, as in Figure 4. Since, in G and H, the edges $u_i z_{u_i u_j}$ and $u_j z_{u_i u_j}$ exist, the effect is "collapsing" these red edges in H. Since the rest of the graphs are the same, this results in identical graphs. The other direction assumes that K contraction suffice on G and H to reach a common contraction, and produces a multicolored clique of G_C. We can argue that in G, one of a_i or b_i is incident to a contraction because of the red

edge a_ib_i in H, and one of c_{ij} or d_{ij} is incident to a contraction because of $c_{ij}d_{ij} \in E(H)$. These require $K/2$ contractions in G and there must be $K/2$ contractions in H. These are all of the contractions, which are therefore a matching. Lemma 8 is used in a more involved argument to show that it is actually the a_i and c_{ij} vertices that are incident to a contraction (using b_i or d_{ij} vertices leads us to add too many new neighbors to extra vertices). Once this is established, we can argue that the contraction partners u_i of a_i and u_j of a_j must form an edge u_iu_j, because the situation from Figure 4 on the right is enforced. ◀

4.2 Parameterization by $k + \Delta$

Let G be a graph of maximum degree Δ. It is not hard to see that applying at most k contractions to G results in a graph of maximum degree $O(k\Delta)$, which we make precise in the next lemma. It then becomes relatively easy to get an FPT algorithm in $k + \Delta$.

▶ **Lemma 10.** *Let G be a graph of maximum degree $\Delta \geq 2$. Then applying $t \geq 0$ contractions to G results in a graph of maximum degree at most $\Delta + t(\Delta - 2)$.*

Moreover, this bound is tight, that is, for any t, there exists a graph G of maximum degree Δ and a sequence of t contractions that results in a graph of maximum degree $\Delta + t(\Delta - 2)$.

Proof sketch. For the upper bound, consider a witness set W from a witness structure of G into a graph H obtained after t contractions. Note that $G[W]$ has at least $|W| - 1$ edges since it is connected. After contracting W to a single vertex, its degree is at most $|W|\Delta - 2(|W| - 1)$, since the inside edges are double-counted. Using $|W| \leq t + 1$ gives the bound. The lower bound is achieved from a graph that starts as a path with $t + 1$ vertices. We add leaves adjacent to every path vertex until they have degree Δ. Contracting the whole path gives the lower bound. ◀

▶ **Proposition 11.** *The Maximum Common Labeled Contraction problem can be solved in time $O((8(k+1)\Delta)^{k+1} \cdot n)$, where k is the number of contractions, $\Delta = \max(\Delta(G), \Delta(H))$, $n = |V(G)| + |V(H)|$ and $m = |E(G)| + |E(H)|$.*

Proof sketch. Consider a recursive branching algorithm as follows. If there exists a vertex u present in one graph but no the other, we know that u must be incident to a contraction and must disappear. By Observation 3, we may do this contraction first. By Lemma 10, u has degree at most $\Delta + k(\Delta - 2)$, which we bound by $(k + 1)\Delta$ for simplicity. We branch on all the ways of contracting u with a neighbor. If no such u exists, then $V(G) = V(H)$. If $E(G) = E(H)$, we are done, so assume there is $uv \in E(G) \setminus E(H)$. We observe that u or v must be incident to some contraction, although it could be in either graph, and u/v could be the kept vertex or not. There are $8(k + 1)\Delta$ ways to branch on such contractions. Since each branch reduces the parameter k by 1, we get a recursive search tree with $(8(k + 1)\Delta)^k$ calls. We add a factor proportional to $k\Delta n$ to check all the edges at each recursion. ◀

4.3 Tractability when parameterizing by the treewidth of $G \cup H$

We now prove that Maximum Common Contraction is FPT in $tw(G \cup H)$, i.e. the treewidth of the graph with vertices $V(G) \cup V(H)$, and edges $E(G) \cup E(H)$. We remark that $tw(G \cup H)$ could be unbounded even if both G, H have bounded treewidth [3], but nonetheless we derive consequences from our result at the end of the section. We use Courcelle's theorem [15] by formulating the problem in the monadic second-order logic on graphs. More precisely, we use an extended version of Courcelle's theorem [4] allowing for *labeled edges* and *labeled vertices* if labels are taken from a finite set. Indeed, in $G \cup H$, we still

need to distinguish vertices in $V(G)$ and $V(H)$, and edges in $E(G)$ and $E(H)$. Alternatively, following the terminology of [21], we are applying Courcelle's theorem on a *relational structure* (Theorem 11.37 in [21]) on $V(G) \cup V(H)$, containing two unary relations labeling $V(G)$ and $V(H)$, and two binary relations for $E(G)$ and $E(H)$. One can verify that the treewith of this relational structure (in the sense of [21, Definition 11.23]) is $tw(G \cup H)$.

▶ **Theorem 12.** *Maximum Common Conraction on fully-labeled graphs is fixed-parameter tractable in $tw(G \cup H)$.*

Proof. In this proof, we denote $V(G)$ by V_1, $V(H)$ by V_2, $E(G)$ by E_1 and $E(H)$ by E_2 for simplicity. We use the following formulation of the problem: we are looking for sub-sets $S_1 \subseteq E_1$ and $S_2 \subseteq E_2$ of edges, and a subset $R \subseteq V_1 \cap V_2$ of vertices ("representatives") such that S_1 and S_2 induce, in G and H respectively, connected components (the witness sets) and such that each connected component contains exactly one vertex in R.

This can be expressed in MSOL in the following way:

$$\begin{aligned}&\exists S_1 \subseteq E_1, \exists S_2 \subseteq E_2, \exists R \subseteq V_1 \cap V_2 \text{ s.t}\\ &\qquad \mathbf{contraction}_1(S_1, R) \text{ and } \mathbf{contraction}_2(S_2, R) \text{ and}\\ &\qquad \forall r, r' \in R\ \mathbf{edge}_1(S_1, r, r') \text{ iff } \mathbf{edge}_2(S_2, r, r')\end{aligned}$$

Where in this formula, for $i = 1, 2$ $\mathbf{contraction}_i(S_i, R)$ checks that that there is exactly one element of R per connected component of V_i induced by S_i, and $\mathbf{edge}_i(S_i, r, r')$ is true if and only there is an edge in E_i between the connected components induced by S_i on V_i containing r and r', respectively. We express $\mathbf{contraction}_i(S_i, R)$ as follows:

$$\begin{aligned}\mathbf{contraction}_i(S_i, R) = \forall r, r' \in R, r \neq r', \neg\mathbf{path}(r, r', S_i) \text{ and } \forall u \in V_i \setminus R\ \exists r \in R \text{ s.t.}\\ \mathbf{path}(u, r, S_i) \text{ and } \forall r' \neq r\ \neg\mathbf{path}(u, r', S_i)\end{aligned}$$

Where in this formula, $\mathbf{path}(x, y, U)$ checks whether $x = y$ or there exists a path between x and y using only edges in U (which can be checked in MSOL [14]) if $x \neq y$. Indeed, two vertices are in the same connected component induced by a sub-set of edges if and only there is a path between them using only edges from the sub-set.

As for $\mathbf{edge}_i(S_i, r, r')$, it can be expressed as:

$$\mathbf{edge}_i(S_i, r, r') = \exists x, y \in V_i \text{ s.t. } \mathbf{path}(x, r, S_i) \text{ and } \mathbf{path}(y, r, S_i) \text{ and } \{x, y\} \in E_i$$

Overall, this expression only uses quantification over edge subsets (S_1, S_2) and vertex sub-sets (R). By the extended version of Courcelle's theorem allowing to optimize for the size of monadic variables ([4] or [21, Exercise 11.44]), one can find S_1, S_2, R verifying this formula while maximizing $|R|$ (or equivalently minimizing $|S_1| + |S_2|$) in a complexity FPT in the treewidth of $(V_1 \cup V_2, E_1 \cup E_2) = G \cup H$. ◀

We now derive consequences of Theorem 12 by establishing parameterizations that lead $G \cup H$ to have bounded treewidth. In the following, we have graphs G and H and $G \cup H$ is the graph with vertex set $V(G) \cup V(H)$ and edge set $E(G) \cup E(H)$. For space reasons, the definition of a tree decomposition can be found in the appendix.

▶ **Lemma 13.** *If H is a contraction of G, then $tw(G \cup H) \leq 2 \cdot tw(G)$.*

Proof sketch. Start with a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G. Then take a witness structure $\mathcal{W}$ of G into T. For each bag X_t of $\mathcal{T}$, and for each $x \in X_t$, add to X_t the representative of $\mathcal{W}(x)$ in H, if not already there. This at most doubles the treewidth.

The bags that contain some $u \in V(G) \cup V(H)$ are connected in the modified decomposition, as otherwise a bag that separates u would also separate $\mathcal{W}(u)$, which should be connected. Also, for each $uv \in E(H)$ some bag has both u, v because $\mathcal{W}(u), \mathcal{W}(v)$ are adjacent. ◀

▶ **Lemma 14.** *If G and H are such that $\exists S_1, S_2$ labeled contraction sequences of size $|S_1| + |S_2| \leq k$ and $G/S_1 = H/S_2$, then $tw(G \cup H) \leq \min(tw(G), tw(H)) + k$*

Proof sketch. Let M be a maximum common contraction of G and H. Since contractions do not increase the treewidth, $tw(M) \leq \min(tw(G), tw(H))$. Then, take a tree decomposition $\mathcal{T}$ of M, then add $(V(G) \setminus V(H)) \cup (V(H) \setminus V(G))$ to every bag. This adds at most k vertices to each bag, and one can argue that this is a valid tree decomposition of $G \cup H$. ◀

▶ **Corollary 15.** *The following results holds:*

- *Labeled Contractibility is FPT in parameter* $\max(tw(G), tw(H))$.
- *Maximum Common Labeled Contraction is FPT in parameter* $k + \min(tw(G), tw(H))$.

Conclusion. We have explored the parameterized complexity of computing a maximum common contraction of two fully-labeld graphs, in which each label is used at most once in each graph. Natural follow-ups could look at the case where labels can be used more than once. Note that it is the hardest possible case, as it contains both unlabelled and uniquely labeled graphs. A first step could be to bound the number of times each label may be present. As for the unlabeled case, it consists in computing edge-contraction distances between graph isomorphism classes, and has not been explored yet. We finish with some open problems: (1) given Theorem 5, is Labeled Contractibility still NP-hard if both input graphs have clique-width three, or even if they are both cographs (clique-width two)? (2) is there a $2^{O(\delta k)}n^c$ time algorithm for Labeled Contractibility, and/or a $2^{O(\Delta k)}n^c$ time algorithm for Maximum Common Labeled Contraction? Or better, could there be subexponential algorithms parameterized by δk or Δk? (3) is Maximum Common Labeled Contraction FPT in $\max(tw(G), tw(H))$?

References

1. Faisal N Abu-Khzam, Édouard Bonnet, and Florian Sikora. On the complexity of various parameterizations of common induced subgraph isomorphism. *Theor. Comp. Sci.*, 2017.
2. Tatsuya Akutsu, Avraham A Melkman, and Takeyuki Tamura. Improved hardness of maximum common subgraph problems on labeled graphs of bounded treewidth and bounded degree. *International Journal of Foundations of Computer Science*, 31(02):253–273, 2020. doi:10.1142/S0129054120500069.
3. Bogdan Alecu, Vadim V Lozin, Daniel A Quiroz, Roman Rabinovich, Igor Razgon, and Viktor Zamaraev. The treewidth and pathwidth of graph unions. *SIAM J on Discrete Mathematics*, 2024.
4. Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991. doi:10.1016/0196-6774(91)90006-K.
5. Rémy Belmonte, Petr A Golovach, Pinar Heggernes, Pim van't Hof, Marcin Kamiński, and Daniël Paulusma. Detecting fixed patterns in chordal graphs in polynomial time. *Algorithmica*, 69(3):501–521, 2014. doi:10.1007/S00453-013-9748-5.
6. Rémy Belmonte, Petr A Golovach, Pim van't Hof, and Daniël Paulusma. Parameterized complexity of three edge contraction problems with degree constraints. *Acta Informatica*, 51(7):473–497, 2014. doi:10.1007/S00236-014-0204-Z.

7 Rémy Belmonte, Pinar Heggernes, and Pim van't Hof. Edge contractions in subclasses of chordal graphs. *Discrete Applied Mathematics*, 160(7-8):999–1010, 2012. doi:10.1016/J.DAM.2011.12.012.

8 Théo Boury, Yann Ponty, and Vladimir Reinharz. Automatic exploration of the natural variability of rna non-canonical geometric patterns with a parameterized sampling technique. In *WABI 2023 – 23rd Workshop on Algorithms in Bioinformatics*, 2023. doi:10.4230/LIPIcs.WABI.2023.20.

9 Andries Evert Brouwer and Henk Jan Veldman. Contractibility and np-completeness. *Journal of Graph Theory*, 11(1):71–79, 1987. doi:10.1002/JGT.3190110111.

10 Horst Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern recognition letters*, 18(8):689–694, 1997. doi:10.1016/S0167-8655(97)00060-3.

11 Gabriel Cardona, Mercè Llabrés, Francesc Rosselló, and Gabriel Valiente. Metrics for phylogenetic networks i: Generalizations of the robinson-foulds metric. *TCBB*, 6(1):46–61, 2008. doi:10.1109/TCBB.2008.70.

12 Gabriel Cardona, Mercè Llabrés, Francesc Rosselló, and Gabriel Valiente. Metrics for phylogenetic networks ii: Nodal and triplets metrics. *TCBB*, 6(3):454–469, 2008. doi:10.1109/TCBB.2008.127.

13 Gabriel Cardona, Joan Carles Pons, Gerard Ribas, and Tomás Martínez Coronado. Comparison of orchard networks using their extended μ-representation. *TCBB*, 21(3):501–507, 2024. doi:10.1109/TCBB.2024.3361390.

14 Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In *Formal models and semantics*, pages 193–242. Elsevier, 1990. doi:10.1016/B978-0-444-88074-1.50010-X.

15 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

16 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015. doi:10.1007/978-3-319-21275-3.

17 Yves-Pol Deniélou, Frédéric Boyer, Alain Viari, and Marie-France Sagot. Multiple alignment of biological networks: A flexible approach. In *CPM*, pages 263–273. Springer, 2009. doi:10.1007/978-3-642-02441-2_23.

18 Hans-Christian Ehrlich and Matthias Rarey. Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(1):68–79, 2011.

19 Fazle E Faisal, Lei Meng, Joseph Crawford, and Tijana Milenković. The post-genomic era of biological network alignment. *EURASIP Journal on Bioinformatics and Systems Biology*, 2015.

20 Michael R Fellows, Danny Hermelin, Frances Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comp. Sci.*, 410(1):53–61, 2009. doi:10.1016/J.TCS.2008.09.065.

21 J Flum and M Grohe. Parameterized complexity theory. 2006. texts in theoretical computer science. an eatcs series, 2006.

22 Philippe Gambette, Leo Van Iersel, Mark Jones, Manuel Lafond, Fabio Pardi, and Celine Scornavacca. Rearrangement moves on rooted phylogenetic networks. *PLoS Comp. Bio.*, 2017.

23 Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13:113–129, 2010. doi:10.1007/S10044-008-0141-Y.

24 Pinar Heggernes, Pim Van'T Hof, Daniel Lokshtanov, and Christophe Paul. Obtaining a bipartite graph by contracting few edges. *SIAM J on Discrete Mathematics*, 2013.

25 Pinar Heggernes, Pim Van't Hof, Benjamin Lévêque, Daniel Lokshtanov, and Christophe Paul. Contracting graphs to paths and trees. *Algorithmica*, 68:109–132, 2014. doi:10.1007/S00453-012-9670-2.

26 Daniel H Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press, 2010.

27 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79(1):39–49, 2013. doi:10.1016/J.JCSS.2012.04.004.

28 Maxim Kalaev, Vineet Bafna, and Roded Sharan. Fast and accurate alignment of multiple protein networks. In *RECOMB*, pages 246–256. Springer, 2008. doi:10.1007/978-3-540-78839-3_21.

29 Marcin Kamiński, Daniël Paulusma, and Dimitrios M Thilikos. Contractions of planar graphs in polynomial time. In *ESA*, pages 122–133. Springer, 2010. doi:10.1007/978-3-642-15775-2_11.

30 Ramaswamy Krithika, Pranabendu Misra, Ashutosh Rai, and Prafullkumar Tale. Lossy kernels for graph contraction problems. In *36th IARCS Annual Conference on Foundations of Software Technology and Theor. Comp. Sci.* Schloss Dagstuhl – Leibniz Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.23.

31 Vincent Lacroix, Ludovic Cottret, Patricia Thébault, and Marie-France Sagot. An introduction to metabolic networks and their structural analysis. *TCBB*, 5(4):594–617, 2008. doi:10.1109/TCBB.2008.79.

32 Manuel Lafond, Alitzel López Sánchez, and Weidong Luo. Cluster Editing on Cographs and Related Classes. In Olaf Beyersdorff, Michał Pilipczuk, Elaine Pimentel, and Nguyen Kim Thang, editors, *42nd International Symposium on Theoretical Aspects of Computer Science (STACS 2025)*, volume 327 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 64:1–64:21, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2025.64.

33 Kaari Landry, Aivee Teodocio, Manuel Lafond, and Olivier Tremblay-Savard. Defining phylogenetic network distances using cherry operations. *TCBB*, 20(3):1654–1666, 2022. doi:10.1109/TCBB.2022.3162991.

34 Asaf Levin, Daniël Paulusma, and Gerhard J Woeginger. The computational complexity of graph contractions i: Polynomially solvable and np-complete cases. *Networks: An International Journal*, 51(3):178–189, 2008. doi:10.1002/NET.20214.

35 Asaf Levin, Daniël Paulusma, and Gerhard J Woeginger. The computational complexity of graph contractions ii: two tough polynomially solvable cases. *Networks: An International Journal*, 52(1):32–56, 2008. doi:10.1002/NET.20249.

36 Bertrand Marchand, Yann Ponty, and Laurent Bulteau. Tree diet: reducing the treewidth to unlock fpt algorithms in rna bioinformatics. *Algorithms for Molecular Biology*, 17(1):8, 2022. doi:10.1186/S13015-022-00213-Z.

37 Bertrand Marchand, Nadia Tahiri, Olivier Tremblay-Savard, and Manuel Lafond. Finding Maximum Common Contractions Between Phylogenetic Networks. In *WABI*, pages 16:1–16:24, 2024. doi:10.4230/LIPIcs.WABI.2024.16.

38 Jiří Matoušek and Robin Thomas. On the complexity of finding iso-and other morphisms for partial k-trees. *Discrete Mathematics*, 108(1-3):343–364, 1992. doi:10.1016/0012-365X(92)90687-B.

39 Mohsen Alambardar Meybodi, Fedor V Fomin, Amer E Mouawad, and Fahad Panolan. On the parameterized complexity of [1, j]-domination problems. *Theor. Comp. Sci.*, 804:207–218, 2020. doi:10.1016/J.TCS.2019.11.032.

40 Philippe Rinaudo, Yann Ponty, Dominique Barth, and Alain Denise. Tree decomposition and parameterized algorithms for rna structure-sequence alignment including tertiary interactions and pseudoknots. In *WABI*, pages 149–164. Springer, 2012.

41 David F Robinson and Leslie R Foulds. Comparison of phylogenetic trees. *Mathematical biosciences*, 53(1-2):131–147, 1981.

42 Saket Saurabh, Uéverton dos Santos Souza, and Prafullkumar Tale. On the parameterized complexity of grid contraction. *Journal of Computer and System Sciences*, 129:22–38, 2022. doi:10.1016/J.JCSS.2022.04.004.

43 Francesc Serratosa. Redefining the graph edit distance. *SN Computer Science*, 2(6):438, 2021. doi:10.1007/S42979-021-00792-5.

44 Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, 2009. doi:10.14778/1687627.1687631.

45 Jizhen Zhao, Russell L Malmberg, and Liming Cai. Rapid ab initio prediction of rna pseudoknots via graph tree decomposition. *J of Mathematical Biology*, 56:145–159, 2008.

A Proofs for Section 2 (Preliminary notions)

▶ **Observation 2.** *Let G, H be graphs. Then H is a contraction of G if and only if there exists a witness structure of G into H.*

Proof. The forward direction can be obtained by starting with G and a partition of $V(G)$ in which every vertex is by itself, then after each contraction (u, v), we update the set associated with u by adding all the elements associated with v. When we are done, only $V(G)$ remains and the resulting vertex-set association forms a witness structure. For the reverse direction, we note that the connected subgraphs $G[W_i]$ can all be contracted to a single vertex independently, and under the conditions of witness structures, doing so must result in H. ◀

▶ **Observation 3.** *Let $\mathcal{W}$ be a witness structure of G and H, and let $uv \in E(G)$ such that u, v are in the same witness set of $\mathcal{W}$. If $v \notin V(H)$, then H is a contraction of $G/(u, v)$.*

Proof. Denote $G' = G/(u, v)$. It suffices to consider the partition $\mathcal{W}'$ of $V(G')$ obtained from $\mathcal{W}$ by just removing v from $\mathcal{W}(u)$. It is easy to see that $|\mathcal{W}'| = |V(H)|$, that $\mathcal{W}(u) \setminus \{v\}$ still contains the same representative of $V(H)$ since $v \notin V(H)$, and that $G'[\mathcal{W}(u) \setminus \{v\}]$ is still connected. Moreover, the edges with one endpoint in $\mathcal{W}(u) \setminus \{v\}$ and the other outside are the same as $\mathcal{W}(u)$, and so $\mathcal{W}'$ is a witness structure of G' into H. By Observation 2, H is a contraction of G. ◀

B Proofs for Section 3 (Labeled Contractibility)

▶ **Theorem 4.** *The* Labeled Contractibility *problem is W[1]-hard in parameter k, which is the number of contractions needed to transform one input graph into the other.*

Proof. Let G_C be an instance of Multicolored Clique and let G, H be constructed as above. We show that G_C has a multicolored clique if and only if H is a contraction of G.

Suppose that G_C has a multicolored clique $u_1, \ldots, u_k$, where $u_i \in V_i$ for each $i \in [k]$. We present a witness structure $\mathcal{W}$ of G into H. In $\mathcal{W}$, add the sets $\{u_i, t_i\}$ for each $i \in [k]$, and the singleton sets $\{\{v\} : v \in V(G_C) \setminus \{u_1, \ldots, u_k\}\}$. This corresponds to the sequence of contractions $(u_1, t_1), \ldots, (u_k, t_k)$. It is clear that $|\mathcal{W}| = |V(H)|$, that each element of $\mathcal{W}$ has a unique element of $V(H)$, and that each $G[W_i]$ is connected. We must argue that $vw \in E(H)$ if and only if there is some edge between $\mathcal{W}(v)$ and $\mathcal{W}(w)$ in G.

Let $v, w \in V(H)$ be two distinct vertices of H, which are also vertices of G and G_C. If $vw \in E(H)$, then because G started as a copy of H, $vw \in E(G)$. This edge still exists between $\mathcal{W}(v)$ and $\mathcal{W}(w)$ in G, because $\mathcal{W}(v) \neq \mathcal{W}(w)$. So instead assume that $vw \notin E(H)$. Let $v \in V_i, w \in V_j$, and notice that $i \neq j$ since we made V_i and V_j cliques in H. Suppose for contradiction that there is some edge between $\mathcal{W}(u)$ and $\mathcal{W}(v)$ in G. By construction, this edge cannot be vw, and it cannot be $t_i w$ nor $t_j v$, which do not exist. So $\mathcal{W}(v)$ must contain

t_i and $\mathcal{W}(w)$ must contain t_j. By the construction of $\mathcal{W}$, this means that $v = u_i$ and $w = u_j$. Since u_i, u_j are in a clique, $vw \in E(G)$ and $vw \in E(H)$, a contradiction. Thus $\mathcal{W}(u)$ and $\mathcal{W}(v)$ are not adjacent in G, and it follows that $\mathcal{W}$ is a witness structure of G into H. By Observation 2, H is a contraction of G.

Conversely, suppose that one may apply at most k contractions to G to obtain H. Let $\mathcal{W}$ be a witness structure of G into H. For $i \in [k]$, consider the vertex t_i of G, and denote by u the representative of $\mathcal{W}(t_i)$, where $u \in V(H)$. We claim that $u \in V_i$, so assume instead that $u \in V_j$, with $i \neq j$. Recall that t_i is adjacent to every vertex in V_i in G, so if $u \in V_j$, then $\mathcal{W}(u) = \mathcal{W}(t_i)$ is adjacent to every witness set that contains a vertex of V_i. But $u \in V_j$, a contradiction since we assume that u has at least one non-neighbor in V_i in G_C, and thus in H as well. It follows that the unique vertex u of G in the same witness set as t_i belongs to V_i.

Since this holds for every $i \in [k]$, this also implies that no two t_i vertices belong to the same witness set. We may then define the set of k distinct vertices $u_1, \ldots, u_k$ where, for $i \in [k]$, u_i is the vertex of V_i in the same witness set as t_i. We argue that this set forms a clique of G_C. Let $i, j \in [k]$ with $i \neq j$. Because of the edge $t_i t_j$, $\mathcal{W}(t_i)$ and $\mathcal{W}(t_j)$ are adjacent in G. Thus there is an edge between $\mathcal{W}(u_i)$ and $\mathcal{W}(u_j)$, and by the conditions of witness sets the edge $u_i u_j$ must exist in H. Thus $u_1, \ldots, u_k$ forms a multicolored clique of G_C. ◀

▶ **Theorem 5.** *The* Labeled Contractibility *problem is NP-hard, even if the input graphs have clique-width at most* 4*.*

Proof. We first prove the correctness of the reduction, and then argue about the clique-width of the constructed graphs.

Let $a_0, \ldots, a_{n-1}, C, k$ be an instance of Unary Perfect Bin Packing, and G, H constructed as described above. Suppose first that Unary Perfect Bin Packing is a yes-instance, i.e., that there exists an assignmnent ϕ of the items to the bins such that the sum of the sizes of the items in a bin is exactly C for each of the k bins. In G, we first contract each edge (b_i, t_x) such that $\phi(x) = i$ (green edges on the example in Figure 3), keeping the vertex b_i. Since each item is assigned to a bin, each t_x is contracted into some vertex in B, therefore the set of vertices T disappears. Since for each bin i, $\sum_{x\,:\,\phi(x)=i} a_x = C$, and each t_x has exactly $|A_x| = a_x$ neighbors in A, each b_i now has exactly C neighbors in A. Let i be an integer between 0 and $k-1$, and let us denote $v_i^0, \ldots, v_i^{C-1}$ the C neighbors of b_i in A in this partially contracted graph. We next contract each v_i^y (for $0 \leq y \leq C-1$) with the vertex d_{Ci+y} of D (we can, as A and D form a bi-clique in G), keeping the d_{Ci+y} vertices. Doing this for all i, we delete the set A, which leaves only B and D. As required, B D each form a clique, and there is an edge $b_i d_j$ if and only if $\lfloor j/C \rfloor = i$. We have therefore described a contraction sequence from G to H.

In the other direction, suppose that H is a contraction of G. Let $\mathcal{W}$ be a witness structure of G into H. To ease notation a bit, we will denote by $W_i^b = \mathcal{W}(b_i)$ the witness set of each b_i and by $W_j^d = \mathcal{W}(d_j)$ the witness set of each d_j. We argue that each element of T must be part of some W_i^b. Indeed, as B and D are the only vertex sets left in H, the sets W_i^b and W_j^d must form a partition of $V(G)$, and in particular of the elements of A and T are in one of those witness sets. However, if a vertex t_x from T is in some W_j^d, then the after the contractions the vertex d_j becomes connected to all of B, which is not the case in H. Each vertex t_x is therefore in one of the sets W_i^b.

We can then define ϕ as $\phi(x) = i$ if and only if $t_x \in W_i^b$ (such an i is unique as the W_i^b do not intersect).

Let us now argue that ϕ assigns items to bins perfectly. Let u be some vertex of A. If u is in a witness set W_i^b, then after contracting the vertex b_i becomes connected to all of D, which is not the case in H. Therefore, u is in some witness set W_j^d. We also argue the opposite, that for any $d_j \in D$, there is some $u \in A$ such that $u \in W_j^d$. Indeed, if some W_j^d contains no element of A, then note that in G the neighborhood of d_j is composed of $D \setminus \{d_j\} \cup A$, and other elements of D cannot be in W_j^d. Therefore, we would have $W_j^d = \{d_j\}$. In G, $\{d_j\}$ is only adjacent to witness sets that contain some vertex of A or D. As we just argued, each vertex of A is in the witness set of a D vertex. Thus the witness set W_j^d has no neighbor belonging to a W_i^b set, which is a contradiction as d_j shares an edge with some vertex of B in H.

Therefore, we have both that all vertices in A are in some W_j^d, and that each W_j^d contains some vertex in A. Since $|A| = \sum_{x=0}^{n-1} a_x = Ck = |D|$, this implies that each W_j^d is of the form $\{d_j, u\}$, for some $u \in A$ uniquely associated to j. In particular, this implies that each vertex t_x, which has a_x neighbors in A, is therefore adjacent to exactly a_x distinct witness sets W_j^d. We write $S_x = \{d_j \in D : N_G(t_x) \cap W_j^d \neq \emptyset\}$, i.e., the set of vertices of D whose witness set is adjacent to that of t_x. We have $|S_x| = a_x$, and for $x \neq y$, we also have $S_x \cap S_y = \emptyset$. Now, $\forall i \in [0, k-1]$, $\cup_{x:\phi(x)=i} S_x$ denotes the representatives of the witness sets of D neighbor to W_i^b, and therefore the neighborhood of b_i outside of B in H. As each b_i has exactly C neighbors outside of B in H, we must have $\sum_{x\,:\,\phi(x)=i} |S_x| = \sum_{x\,:\,\phi(x)=i} a_x = C$. Therefore, $a_0, \ldots, a_{n-1}, C, k$ is a yes-instance.

Clique-width analysis. We finish by arguing that both G and H have clique-width at most 4. Let us start with H, which we construct inductively on $k = 0, 1, \ldots, k-1$. For $k = 0$, we add vertices $d_0, \ldots, d_{C-1}$ with color 1, b_0 with color 2, and add all edges between colors 1 and 2. This finishes the construction for $k = 0$, with vertices of D added so far having color 1 and those of B having color 2. Then, given the graph H constructed with $k-1$ vertices in B, with all vertices of D having color 1 and those of B having color 2, we introduce the vertices $d_{(k-1)C}, \ldots, d_{kC-1}$ under color 3, vertex b_{k-1} under color 4. We add all edges between color 3 and 4 (which creates exactly the required edges between b_{k-1} and $d_{(k-1)C}, \ldots, d_{kC-1}$), then connect color 1 and 3 as a bi-clique (making D a clique) and likewise for colors 2 and 4 (making B a clique). We finish this step by recoloring 3 with 1 and 4 with 2, in case we may pursue the construction.

As for G, it actually has clique-width 3. We can first construct the subgraph induced by A and T using two colors. Indeed, $G[A \cup T]$ is a forest of star trees, so we can just create each star tree independently with two colors, and then take the disjoint union of all those star trees. This can be done in a way that $G[A \cup T]$ is built with vertices of A having color 1 and those of T having color 2. We then construct the B clique independently using two colors and recolor every vertex of B to 3. We then add B to the construction and add all edges between B and T (colors 1 and 3). Then recolor 3 to 1, so that B and T have the same colors. In the same manner, we construct the D clique and give it color 3, and add edges between colors 2 and 3 (A and D). This results in G. ◀

▶ **Lemma 6.** *If H is obtained from G by $\leq k$ contractions, then $\delta(H) \leq \delta(G) + k$*

Proof. We prove the result by induction. The base case $k = 0$ is trivial. Let us therefore consider now G' obtained from G by a single contraction, of u into v. Let also σ be an elimination order such that $\max_{x \in V(G)} |N^G_{\geq,\sigma}(x)| = \delta(G)$. We distinguish below two cases. In both, we find an order σ' of the vertices of G' such that $\max_{x \in V(G')} |N^{G'}_{\geq,\sigma'}(x)| \leq \delta(G) + 1$, which allows to conclude.

- If $N(u) \cup N(v)$ (neighborhood of v in G') contains no element ranked higher than v in σ, we simply get σ' by removing u from σ. We have $N^G_{\sigma,\geq}(v) = N^{G'}_{\geq,\sigma'}(v) = \emptyset$. As for $x \in V(G')$ and $x \neq v$, $|N^{G'}_{\geq,\sigma'}(x)| \leq |N^G_{\geq,\sigma}(x)|$ if x is before u in σ (the size decreases by 1 if x is a neighbor of both u and v and stays the same otherwise), $|N^{G'}_{\geq,\sigma'}(x)| = |N^G_{\geq,\sigma}(x)|$ if x is after v in σ and $|N^{G'}_{\geq,\sigma'}(x)| \leq |N^G_{\geq,\sigma}(x)| + 1$ if x is between u and v in σ (with an increase in the case where x is a neighbor of u but not v). Overall, we do have $\max_{x \in V(G')} |N^{G'}_{\geq,\sigma'}(x)| \leq \delta(G) + 1$.
- Otherwise, let t be the highest-ranked element of $N(u) \cup N(v)$ by σ (it is after v, as otherwise we would be in the previous case). In this case, we get σ' by removing u an moving v right after t in σ. By doing so, we have $|N_{\geq,\sigma'}(v)| = 0$, while for $x \in V(G')$ such that $x \neq v$, $|N^{G'}_{\geq,\sigma'}(x)| \leq |N^G_{\geq,\sigma}(x)| + 1$ if x is a neighbor of u after u in σ, or if x is a neighbor of v after v in σ. If x is not a neighbor of u and v, or if x is before u in σ, $|N^{G'}_{\geq,\sigma'}(x)| \leq |N^G_{\geq,\sigma}(x)|$. Again, we obtain $\max_{x \in V(G')} |N^{G'}_{\geq,\sigma'}(x)| \leq \delta(G) + 1$.

We have therefore found an ordering σ' of G' such that $\max_{x \in V(G')} |N^{G'}_{\geq,\sigma'}(x)| \leq \delta(G) + 1$. By definition of the degeneracy, $\delta(G') \leq \delta(G) + 1$. Therefore, by induction over the number k of contractions, $\delta(H) \leq \delta(G) + k$ ◀

C Proofs for Section 4 (Maximum Common Labeled Contractions)

▶ **Lemma 8.** *Let H be a contraction of a graph G, and let $\mathcal{W}$ be the witness structure of G into H. Let $R \subseteq E(G)$ be a matching of G such that, for every edge $xy \in R$, the vertices x and y are in the same witness set of $\mathcal{W}$.*

Let G' be the graph obtained after contracting every edge of R, making sure not to remove a representative vertex of $\mathcal{W}$. Then H is a contraction of G'.

Proof. As in the proof of Observation 3, it suffices to observe that if we start from $\mathcal{W}$ and contract (x, y), where $xy \in R$, then after removing y from $\mathcal{W}(y)$, the resulting collection $\mathcal{W}'$ is a witness structure of $G/(x, y)$ into H (this of course requires y to not be a representative). Moreover, if $R \setminus \{xy\}$ is non-empty, then any edge that remains still has its ends in one witness set, because R is a matching. Thus we may continue applying such contractions inductively. ◀

▶ **Theorem 9.** *The Maximum Common Labeled Contraction is W[1]-hard in parameter k, the total number of contractions needed to achieve a common contraction, even if both input graphs have degeneracy 4.*

Proof of Theorem 9. Consider an instance $G_C = (V, E)$ of Multicolored Clique and graphs G, H constructed as above, along with parameter $K = 2(k + \binom{k}{2})$, the total number of contractions.

We first argue that the degeneracy of G and H is at most 4. Since G is a subgraph of H, it suffices to consider H only. We provide an elimination sequence in which every vertex has at most four neighbors before being deleted. First consider any extra vertex in a V'_i or Z'_{ij} set. These have degree two, so we may delete them first. Next consider any z_e vertex of H from some Z_{ij} set, where $e = uv$. It has four neighbors $\{u, v, c_{ij}, d_{ij}\}$. Delete all the Z_{ij} vertices, for every i, j. After that, consider any u vertex in some V_i set. It has two remaining neighbors a_i and b_i. Delete all of those next. Then delete the d_{ij} vertices (one remaining neighbor c_{ij}) and the c_{ij} vertices (two remaining neighbors a_i, a_j). The resulting graph has maximum degree 1 as the a_ib_i's form a matching, and it follows that H and G have degeneracy 4.

We next show that G_C admits a multicolored clique if and only if G, H have a common contraction that can be achieved with a total of at most K contractions.

Suppose that G_C admits a multicolored clique $u_1, u_2, \ldots, u_k$, where each u_i is in V_i. We perform the same contractions in G and H. In either graph:

- for each $i \in [k]$, contract (a_i, u_i);
- for each distinct $i, j \in [k]$, contract $(c_{ij}, z_{u_i u_j})$.

Note that the kept vertex is a_i and c_{ij}, respectively. This clearly requires $K = 2(k + \binom{k}{2})$ contractions in total. We must argue that applying them in G or H results in the same graph. Let M_G and M_H be the graphs obtained by applying the above contractions on G and H, respectively. We use the witnesses for our arguments.

Let $\mathcal{W}$ be the witness structure of G into M_G. Note that $\mathcal{W}$ contains $\{a_i, u_i\}$ for $i \in [k]$, $\{c_{ij}, z_{u_i u_j}\}$ for distinct $i, j \in [k]$, and every other vertex is in a witness set of size 1. Also note that $\mathcal{W}$ is also a witness structure of H into M_H, since we apply the same contractions on both graphs. We argue that two witness sets of $\mathcal{W}$ are adjacent in G if and only if the same two witness sets are adjacent in H, from which we can deduce that $M_G = M_H$.

- First consider $i \in [k]$ and some vertex $v \in V_i' \setminus \{u_i\}$. Then $\{v\}$ is an element of $\mathcal{W}$. From the construction, we see that H adds no neighbor to v. Thus $N_G(v) = N_H(v)$, and it follows that $\{v\}$ is adjacent to some $W \in \mathcal{W}$ in G if and only if $\{v\}$ is adjacent to W in H.
- Consider distinct $i, j \in [k]$ and a $z \in Z_{ij}' \setminus \{z_{u_i u_j}\}$ vertex. Then $\{z\} \in \mathcal{W}$, and again from $N_G(z) = N_H(z)$ we make the same conclusion as in the previous case.
- From now on we do not need to consider the adjacencies of $\{v\}$ and $\{z\}$ witness sets. Only witness sets of the form $\{b_i\}, \{d_{ij}\}, \{a_i, u_i\}, \{c_{ij}, z_{u_i u_j}\}$ remain. Consider vertex b_i, where $i \in [k]$. Then $\{b_i\} \in \mathcal{W}$. In either graph, $\{b_i\}$ is adjacent to $\{a_i, u_i\}$ and no other remaining witness set.
- Consider vertex d_{ij}, with distinct $i, j \in [k]$. In either graph, $\{d_{ij}\}$ is only adjacent to $\{c_{ij}, z_{u_i u_j}\}$ among the witness sets that remain to consider.
- The remaining witness sets have the form $\{a_i, u_i\}$ or $\{c_{ij}, z_{u_i u_j}\}$. Consider $i \in [k]$ and witness set $\{a_i, u_i\}$. Among the witness sets that remain to consider, in H, it is adjacent to all witness sets in $\{\{c_{ij} z_{u_i u_j}\} : j \in [k] \setminus \{i\}\}$, but not to any other $\{a_j u_j\}$ set nor to any other $\{c_{jh} z_e\}$ witness set with $j, h \neq i$. The same is true in G, because u_i has every $z_{u_i u_j}, j \in [k] \setminus \{i\}$ in its neighborhood.
- Finally, for distinct $i, j \in [k]$, we must consider $\{c_{ij}, z_{u_i u_j}\}$. In either graph, all their adjacencies were handled previously, as these are not adjacent to any other $\{c_{xy}, z_{u_x u_y}\}$ set.

We have thus shown that if we partition $V(G)$ into $\mathcal{W}$ and apply the corresponding contractions, and do the same on H, we obtain the same graph.

In the converse direction, let M be a common contraction of G and H achievable in a total of K contractions. We need two witness structures, one for G into M and one for H into M. So let $\mathcal{X}$ be a witness structure of G into M, and $\mathcal{Y}$ a witness structure of H into M. Note that for a vertex w, we write $\mathcal{X}(w)$ for the witness set of $\mathcal{X}$ that contains w, and $\mathcal{Y}(w)$ for the witness set of $\mathcal{Y}$ that contains w.

We must consider multiple ways that G and H could be turned into M, so we split the proof into claims.

▷ Claim 16. For each $i \in [k]$, at least one of $\mathcal{X}(a_i)$ or $\mathcal{X}(b_i)$ has two elements or more. Likewise, for each distinct $i, j \in [k]$, at least one of $\mathcal{X}(c_{ij})$ or $\mathcal{X}(d_{ij})$ has two elements or more.

Proof. Suppose for contradiction that $\mathcal{X}(a_i) = \{a_i\}$ and $\mathcal{X}(b_i) = \{b_i\}$. Then a_i, b_i are representatives in $\mathcal{X}$ and thus vertices of M. Since $a_ib_i \notin E(G)$, they do not share an edge in M. Now considering H, we have $\mathcal{Y}(a_i) \neq \mathcal{Y}(b_i)$ since a_i, b_i are both in M. However, $\mathcal{Y}(a_i), \mathcal{Y}(b_i)$ are adjacent in H, and thus according to $\mathcal{Y}$ we should have the edge a_ib_i in M, a contradiction.

The argument for $\mathcal{X}(c_{ij}), \mathcal{X}(d_{ij})$ is identical. Assume that $\mathcal{X}(c_{ij}) = \{c_{ij}\}$ and $\mathcal{X}(d_{ij}) = \{d_{ij}\}$. Then c_{ij}, d_{ij} are in M and do not share an edge since $c_{ij}d_{ij} \notin E(G)$. However, $c_{ij}d_{ij} \in E(H)$, leading to the same contradiction as above. ◁

Claim 16 lets us construct a matching $R \subseteq E(G)$ as follows. For each $i \in [k]$, by Claim 16 there is either an edge a_iu or b_iu whose ends are in the same set of $\mathcal{X}$, where u is a neighbor of a_i or b_i in G and thus $u \in V'_i$. Choose any such edge and add it to R (make one choice per $i \in [k]$, which clearly gives a matching so far). Likewise for distinct $i, j \in [k]$, there is either an edge $c_{ij}z$ or $d_{ij}z$ whose ends are in the same set of $\mathcal{X}$, where $z \in Z'_{ij}$. Add any such edge to R, for each i, j, and notice that this does produce a matching R such that the ends of each edge are in the same witness set.

By Lemma 8, R corresponds to a set of contractions we can perform first in a sequence from G to M. We have $|R| = k + \binom{k}{2} = K/2$, and applying the contractions removes $K/2$ vertices from G. Since $V(G) = V(H)$, at least $K/2$ vertices must also disappear from H to M, and it follows that H also requires at least $K/2$ contractions. Because M can be reached with a total of K contractions, we deduce from Claim 16 that after applying the contractions corresponding to R, there is no room for further contractions. It follows that *exactly* one of a_i or b_i has a partner in its witness set of $\mathcal{X}$, and the same holds for c_{ij} versus d_{ij} (we note that a_i, b_i cannot be in the same witness set of $\mathcal{X}$ since they are not neighbors, same for c_{ij}, d_{ij}). Furthermore, the edges contracted from G to M must consist of precisely R, although we do not know which end of each edge is kept or removed. However we can argue that it is the a_i's and the c_{ij}'s that are part of the matching, not the b_i's or d_{ij}'s.

▷ Claim 17. For each $i \in [k]$, $\mathcal{X}(a_i) = \{a_i, u\}$ for some $u \in V'_i$. Likewise for each distinct $i, j \in [k]$, $\mathcal{X}(c_{ij}) = \{c_{ij}, z\}$ for some $z \in Z'_{ij}$.

Proof. Let $i \in [k]$ and note that by Claim 16 and the above discussion that either $|\mathcal{X}(a_i)| = 2$ or $|\mathcal{X}(b_i)| = 2$, but not both. Suppose for contradiction that $\mathcal{X}(b_i) = \{b_i, u\}$ for some $u \in V'_i$. Then $\mathcal{X}(a_i) = \{a_i\}$. Since at most one vertex of V'_i can be part of a contraction, in M every vertex of $V'_i \setminus \{u\}$ is present. So by inspecting $N_G(a_i)$, we infer that in M, the vertex a_i is adjacent to $V'_i \setminus \{u\}$, and to u or b_i, whichever is the representative of $\mathcal{X}(b_i)$, and a_i has no other neighbor in M.

Consider vertex c_{ij} of H and $\mathcal{Y}(c_{ij})$. We cannot have $\mathcal{Y}(a_i) = \mathcal{Y}(c_{ij})$, as this would make $\mathcal{Y}(a_i)$ adjacent to all extra vertices of Z'_{ij} that are in H (and such vertices exist in M since there are $4k^3 > K$ of them). The established adjacencies of a_i in M do not allow this. Thus $\mathcal{Y}(c_{ij}) \neq \mathcal{Y}(a_i)$ and the two witness sets are adjacent in H because $a_ic_{ij} \in E(H)$. The representative x of $\mathcal{Y}(c_{ij})$ must be a neighbor of a_i in M, and so x is in $V'_i \cup \{b_i\}$. However, this makes x adjacent to all extra vertices of Z'_{ij} that remain in M, which is clearly not possible by applying the contractions on G of the matching R discussed above. Therefore, $|\mathcal{X}(a_i)| = 2$.

Let us move on to the second part, that each $\mathcal{X}(c_{ij})$ has two elements. The proof goes along the same lines. Suppose instead that $\mathcal{X}(d_{ij}) = \{d_{ij}, z\}$ for some $z \in Z'_{ij}$ and $\mathcal{X}(c_{ij}) = \{c_{ij}\}$. From $N_G(c_{ij})$ we infer that in M, c_{ij} has only elements of $Z'_{ij} \cup \{d_{ij}\}$ in its neighborhood. Now consider $\mathcal{Y}(a_i)$ and note that $\mathcal{Y}(a_i) \neq \mathcal{Y}(c_{ij})$ as this would make c_{ij} adjacent to extra vertices of V'_i. The two witness sets are adjacent, and so the representative x of $\mathcal{Y}(a_i)$ is in $Z'_{ij} \cup \{d_{ij}\}$. This makes x adjacent to extra vertices of V'_i, which is not possible from the contractions of R on G. Thus $|\mathcal{X}(c_{ij})| = 2$. ◁

By Claim 17, each a_i is contracted with exactly one $u_i \in V_i'$. We claim that if we take all u_i's in the same witness set as an a_i, we get a multicolored clique of G_C.

▷ Claim 18. Consider distinct $i, j \in [k]$ and u_i, u_j be such that $\mathcal{X}(a_i) = \{a_i, u_i\}$ and $\mathcal{X}(a_j) = \{a_j, u_j\}$. Then $u_i \in V_i$, $u_j \in V_j$, and $u_iu_j \in E(G_C)$.

Proof. We know that $u_i \in V_i'$ and $u_j \in V_j'$. Let us assume that either u_i or u_j is an extra vertex, or that $u_iu_j \notin E(G_C)$. Either way, in G and H the vertex $z_{u_iu_j}$ does not exist. Now let $z \in Z_{ij}'$ such that $\mathcal{X}(c_{ij}) = \{c_{ij}, z\}$, which exists by Claim 17. In G and H, at least one of u_i or u_j is not adjacent to z, as otherwise z would be $z_{u_iu_j}$. Suppose without loss of generality that u_i is not adjacent to z, and let $x \in \{a_i, u_i\}$ be the representative of $\mathcal{X}(a_i)$, and $y \in \{c_{ij}, z\}$ the representative of $\mathcal{X}(c_{ij})$. Then in G, the witness sets $\mathcal{X}(x) = \{a_i, u_i\}$ and $\mathcal{X}(y) = \{c_{ij}, z\}$ are not adjacent.

Observe that in M, the neighborhood of x includes all extra vertices $V_i' \setminus \{x\}$. Likewise, y has among its neighbors in M all extra vertices of $Z_{ij}' \setminus \{y\}$. One implication is that in H, $\mathcal{Y}(x)$ must be adjacent to all witness sets of $\mathcal{Y}$ that contain an extra vertex of V_i' (except possibly $\mathcal{Y}(x)$ itself). Since there are $4k^3 > 2K$ extra vertices, this is only possible if $\mathcal{Y}(x)$ has a vertex that is a neighbor of all extra vertices, which is either b_i or a_i. Since b_i is in M, $b_i \notin \mathcal{Y}(x)$, and thus a_i is in $\mathcal{Y}(x)$. Using the same reasoning on y and c_{ij}, we get that c_{ij} is in $\mathcal{Y}(y)$. Hence $\mathcal{Y}(x)$ and $\mathcal{Y}(y)$ are adjacent in H, a contradiction since $\mathcal{X}(x)$ and $\mathcal{X}(y)$ are not adjacent in G. ◁

By this last claim, we can take the set $\{u_i : u_i \in \mathcal{X}(a_i), i \in [k]\}$, and all these vertices are in G_C and every pair shares an edge. Thus G_C has a multicolored clique. ◀

▶ **Proposition 11.** *The* Maximum Common Labeled Contraction *problem can be solved in time* $O((8(k+1)\Delta)^{k+1} \cdot n)$*, where* k *is the number of contractions,* $\Delta = \max(\Delta(G), \Delta(H))$*,* $n = |V(G)| + |V(H)|$ *and* $m = |E(G)| + |E(H)|$*.*

Proof. Consider an instance of Maximum Common Labeled Contraction consisting of labeled graphs G, H of maximum degree Δ or less, and parameter k for the allowed number of contractions. We describe a recursive branching algorithm in which each branching path makes at most k contractions (or otherwise returns that no solution is possible in this path). By Lemma 10, any instance encountered in a recursion has maximum degree at most $\Delta + k(\Delta - 2)$, which we upper bound by $(k+1)\Delta$ for simplicity.

So we assume that we are in a recursive call which receives G and H and integer k. Suppose that there is some $v \in V(G)$ such that $v \notin V(H)$. Then to obtain any common contraction, v must be in the same witness set as one of its neighbors. We branch on every possible contraction (w, v), for every neighbor w of v. We decrease the parameter k by 1 in each branch. This branches into at most $(k+1)\Delta$ cases, and by Observation 3 one of these branches will lead to a maximum common contraction with k operations if one exists. The same branching can be applied if H has some v not in G.

So assume that $V(G) = V(H)$. If $E(G) = E(H)$ then we are done, so assume without loss of generality that G has an edge uv that is not in H. Observe that to obtain a common contraction, u or v must be incident to at least one contraction, in at least one of G or H, as otherwise the edge uv stays present in G and it remains a non-edge in H. That is, one of u or v has a neighbor in the same witness set, in one of the graphs. By Observation 3, we can branch into all the contractions that u or v could be part of, in either graph, i.e., we branch into every contraction (x, y) where $\{u, v\} \cap \{x, y\} \neq \emptyset$, and where $xy \in E(G) \cup E(H)$. The number of such possible contractions incident to u is at most $2(|N_G(u)| + |N_H(u)|) \leq 2((k+1)\Delta + (k+1)\Delta) = 4(k+1)\Delta$ (we multiply by 2 because contractions are ordered pairs). The same bound holds for v, so in total we branch into at most $8(k+1)\Delta$ possibilities, still decreasing k by 1 in each case.

It is easy to see that the algorithm finds a common contraction achievable with at most k contractions if and only if one exists. For the complexity, the algorithm creates a recursion tree in which each vertex has at most $8(k+1)\Delta$ children, and whose depth is at most k. In each recursive call, there are at most $2k\Delta \cdot n$ edges, so we can check in time $O(k\Delta n)$ whether G and H are the same graph, and if not find a vertex or a vertex pair to branch on. This results in a running time of $O((8(k+1)\Delta)^{k+1} \cdot n)$. ◀

Let G be a graph. A *tree decomposition* of G is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ where T is a tree and, for each $t \in V(T)$, $X_t \subseteq V(G)$. The sets X_t are called *bags*. Moreover, $\mathcal{T}$ satisfies the following:

1. for each $x \in V(G)$, some bag X_t of $\mathcal{T}$ contains x.
2. for each $x \in V(G)$, the vertices of T corresponding to bags that contain x form a connected subtree of T.
3. for each edge $uv \in E(G)$, some bag contains both u and v.

The *width* of $\mathcal{T}$ is $\max_{t \in V(T)}(|X_t| - 1)$, and the *treewidth* of G is the minimum width of a tree decomposition of G.

▶ **Lemma 13.** *If H is a contraction of G, then $tw(G \cup H) \leq 2 \cdot tw(G)$.*

Proof. Let $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be a tree decomposition of G, and $\mathcal{W}$ a witness structure of G into H. We build a new decomposition $\mathcal{T}' = (T, \{Z_t\}_{t \in T})$ and argue it is a tree decomposition of $G \cup H$. The underlying tree T is the same for $\mathcal{T}$ and $\mathcal{T}'$. As for the bags, we set:

$$Z_t = X_t \cup \{u \in V(H) : \exists x \in X_t \text{ s.t } x \in \mathcal{W}(u)\}$$

In other words, we take each bag X_t, and for each $x \in X_t$ we add the representative in H of the witness set of x, which results in Z_t. Let us check that $\mathcal{T}'$ is a valid tree decomposition of $G \cup H$.

- (connectivity of vertex representation) First note that a vertex $x \in V(G) \setminus V(H)$ belongs to Z_t if and only if x belongs to X_t, since only vertices of H are added to bags. Since the bags containing x are connected in $\mathcal{T}$, they are still connected in $\mathcal{T}'$. Now Let $u \in V(H)$ and $t \in V(T)$. First, note that since $u \in V(G)$, there is at least one bag of $\mathcal{T}$ and thus $\mathcal{T}'$ containing it. Then, let t_1, t_2, t_3 be three vertices of T such that t_3 is on the path from t_1 to t_2. Suppose that $u \in Z_{t_1}$, $u \in Z_{t_2}$ but $u \notin Z_{t_3}$. By definition of $\mathcal{T}'$, $\exists a \in X_{t_1}$ such that $a \in \mathcal{W}(u)$ and $b \in X_{t_2}$ such that $b \in \mathcal{W}(u)$. Note that $a = u$ is possible, in which case $u \in X_{t_1}$ (likewise, $b = u$ is possible). We also have $\mathcal{W}(u) \cap X_{t_3} = \emptyset$, as otherwise u would be added to Z_{t_3}, and in particular, $a, b \notin X_{t_3}$.
  If $a = b$, we have a contradiction with $\mathcal{T}$ being a valid tree decomposition of G, since $a \in V(G)$ and the bags of $\mathcal{T}$ containing it would be disconnected by t_3. If $a \neq b$, since $\mathcal{T}$ is a tree decomposition of G, X_{t_3} separates a from b, i.e. intersects all paths from a to b in G. However, by the connectivity of $\mathcal{W}(u)$, there is a path from a to b lying entirely in $\mathcal{W}(u)$, which yields a contradiction.
- (edge representation) For $uv \in E(G)$, some bag of $\mathcal{T}$ contains u and v, and the same holds in $\mathcal{T}'$. Then, for each edge uv of H, there must be at least one edge xy of G such that $x \in \mathcal{W}(u)$, $y \in \mathcal{W}(v)$. There must be a bag in $\mathcal{T}$ containing both x and y. By definition, this bag contains u and v in $\mathcal{T}'$, therefore representing the edge.

By definition of $\mathcal{T}'$, each of its bag Z_t is at most twice as big as X_t, because each $x \in X_t$ enforces the addition of at most one new vertex in Z_t, namely the $u \in V(H)$ that is the representative of its witness set. Therefore the width of $\mathcal{T}'$ is at most $2tw(G)$. ◀

On the Complexity of Finding 1-Center Spanning Trees

Pin-Hsian Lee
National Taiwan Normal University, Taipei, Taiwan

Meng-Tsung Tsai
Academia Sinica, Taipei, Taiwan

Hung-Lung Wang
National Taiwan Normal University, Taipei, Taiwan

Abstract

We consider the problem of finding a spanning tree T of a given undirected graph G such that any other spanning tree can be obtained from T by removing k edges and subsequently adding k edges, where k is minimized over all spanning trees of G. We refer to this minimum k as the *treeradius* of G.

Treeradius is an interesting graph parameter with natural interpretations: (1) It is the smallest radius of a Hamming ball centered at an extreme point of the spanning tree polytope that covers the entire polytope. (2) Any graph with bounded treeradius also has bounded treewidth. Consequently, if a problem admits a fixed-parameter algorithm parameterized by treewidth, it also admits a fixed-parameter algorithm parameterized by treeradius.

In this paper, we show that computing the exact treeradius for n-vertex graphs requires $2^{\Omega(n)}$ time under the Exponential Time Hypothesis (ETH) and does not admit a PTAS, with an inapproximability bound of 1153/1152, unless P = NP. This hardness result is surprising, as treeradius has significantly higher ETH complexity compared to analogous problems on shortest path polytopes and star subgraph polytopes.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Treeradius, Spanning tree polytope, Shortest s,t-path polytope

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.43

Funding This research was supported in part by the National Science and Technology Council under contract NSTC 113-2628-E-003-001-MY3 and 113-2221-E-001-026.

Acknowledgements We want to thank the anonymous reviewers for their valuable comments.

1 Introduction

We consider the problem of finding a most representative spanning tree of a given undirected connected simple graph $G = (V, E)$. A spanning tree T of G is considered *most representative* if any other spanning tree of G can be obtained from T by removing k edges and subsequently adding k edges, where k is minimized over all spanning trees of G. We refer to this minimum k as *treeradius*. We show that if the treeradius of G is bounded, then the treewidth of G also is bounded. Consequently, if a problem admits a fixed-parameter algorithm parameterized by treewidth, it also admits a fixed-parameter algorithm parameterized by treeradius. Moreover, such a spanning tree has applications in devising algorithms for computing multi-objective spanning trees, which we discuss in Section 1.1.

The most representative spanning tree of G corresponds to a discrete 1-center among the extreme points in the spanning tree polytope of G, a concept that is well-established in the literature. Formally, for each spanning tree T of G, we define its indicator vector $v(T) \in 2^E$, where the ith coordinate of $v(T)$ is 1 if the ith edge in E is included in T, and 0 otherwise.

© Pin-Hsian Lee, Meng-Tsung Tsai, and Hung-Lung Wang;
licensed under Creative Commons License CC-BY 4.0

Placing the starting points of these indicator vectors at the origin in the space 2^E, the convex hull of their endpoints is the *spanning tree polytope*. Since all indicator vectors have the same number of coordinates with value 1, their endpoints are precisely the extreme points of the polytope. A discrete (resp. continuous) 1-*center* among these extreme points is an extreme point p (resp. any point p in the space) that minimizes the radius of the Hamming ball centered at p and covering the entire polytope. If the representing point of a spanning tree corresponds to a discrete 1-center of the polytope, we refer to it as a 1-*center spanning tree* of G. Clearly, the notion of a 1-center spanning tree is equivalent to that of the most representative spanning tree. An illustration of the spanning tree polytope is depicted in Figure 1.

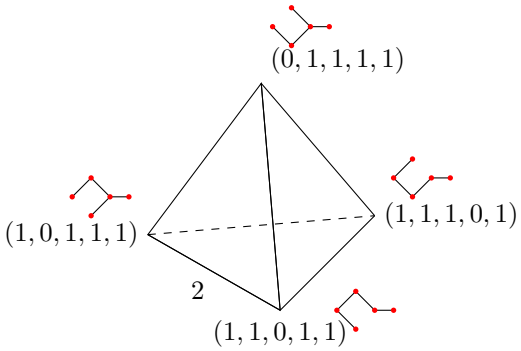


Figure 1 The spanning tree polytope of the banner graph, a 5-vertex pseudotree with a 4-cycle, contains the maximum number of vertices among all spanning tree polytopes in three-dimensional space, as shown in Lemma 26.

Though a discrete 1-center in a set of n points can be found in $O(n^2)$ time using a naive algorithm and cannot be computed in $O(n^{2-\varepsilon})$ time for any constant $\varepsilon > 0$ [1] unless the Hitting Set Conjecture [3] fails, finding a discrete 1-center among the extreme points in the spanning tree polytope is a fundamentally different and potentially more computationally intensive, as the number of spanning trees in G can be exponential in $|V|$. Assuming the Exponential Time Hypothesis (ETH) [16, 17], we have the following result:

▶ **Theorem 1.** *Given an n-vertex undirected connected simple graph G, finding a 1-center spanning tree of G requires $2^{\Omega(n)}$ time unless ETH fails and can be solved in $2^{\tilde{O}(n)}$ time. Moreover, computing the treeradius of G is APX-complete with inapproximability constant* 1153/1152.

We say that a problem has *ETH complexity* $2^{\tilde{\Theta}(n^c)}$ if there exists an algorithm that solves it in $2^{\tilde{O}(n^c)}$ time, and assuming ETH every algorithm solving this problem requires at least $2^{\tilde{\Omega}(n^c)}$ time, where $\tilde{\Omega}(f(n)) = \Omega(f(n)\text{polylog } n)$ and $\tilde{O}(f(n)) = O(f(n)\text{polylog } n)$. We find the ETH complexity $2^{\tilde{\Theta}(n^1)}$ of 1-center spanning trees interesting when compared to the computation of discrete 1-centers among the extreme points of other polytopes associated with G, as summarized in Theorem 2. Finding a discrete 1-center among the extreme points naturally generalizes to any graph problem where the solution is chosen from a collection of edge sets of equal size, ensuring a one-to-one correspondence between solutions and the extreme points of the polytope in 2^E. However, the polytopes we considered are not arbitrary and must satisfy the following conditions.

- We exclude the polytopes where outputting an extreme point is intractable. This is because outputting a discrete 1-center among the extreme points is no easier, so the hardness result for outputting an extreme point already implies the hardness of finding a discrete 1-center among the extreme points.

Typical examples include finding a discrete 1-center among the extreme points in the Hamiltonian path polytopes and the k-edge dominating set polytopes [13].

- We exclude the polytopes where enumerating all extreme points can be solved in polynomial time. This is because outputting a discrete 1-center among the extreme points is easy if all extreme points are given.
  Typical examples include finding a discrete 1-center among the extreme points of the minimum cut polytope, which has $O(n^2)$ minimum cuts [15], and the k-cycle polytope for any fixed k, which contains $O(n^k)$ k-cycles. In both cases, all extreme points can be identified efficiently.

▶ **Theorem 2.** *The ETH complexities of finding a discrete 1-center among the extreme points in the spanning tree polytope, the shortest s,t-path polytope, and the star subgraph S_r polytope of an n-vertex graph are*

$2^{\tilde{\Theta}(n^1)}, 2^{\tilde{\Theta}(n^{1/2})}, 2^{\tilde{\Theta}(n^0)}$, *respectively.*

In addition, we note that finding a continuous 1-center in a set of n points is known to be NP-hard [12, 20] but admits a PTAS [21]. This complements the APX-hardness result in Theorem 1, as both problems involve finding a 1-center and have exponentially large solution spaces relative to their input sizes.

1.1 Applications

Finding a 1-center spanning tree has applications in devising algorithms for computing multi-objective spanning trees, such as max-color spanning trees [8, 23] and max-leaf spanning trees [7, 10, 19, 25].

For graph classes with treeradius at most k, a 1-center spanning tree $T_{1\text{-}center}$ is a good approximation of a max-color spanning tree $T_{max\text{-}color}$ because $T_{max\text{-}color}$ can be obtained from $T_{1\text{-}center}$ by removing k edges and subsequently adding k edges. So the number of colors in $T_{1\text{-}center}$ cannot be less than that in $T_{max\text{-}color}$ by k.

Similarly, for such graph classes, a 1-center spanning tree $T_{1\text{-}center}$ is also a good approximation of a max-leaf spanning tree $T_{max\text{-}leaf}$. Because deleting k edges and subsequently adding k edges can change the degrees of at most $4k$ nodes, the number of leaves in $T_{1\text{-}center}$ cannot be less than that in $T_{max\text{-}leaf}$ by $4k$.

1.2 Related Work

Finding a discrete (resp. continuous) 1-center has been widely studied when the input consists of a collection of strings. While related to our work, these studies differ in that our polytope is defined by specific edge subsets of an underlying graph, serving as a *succinct representation* of the polytope. In contrast, the strings in these related works are given explicitly and do not allow for such a compact representation. This distinction makes proving hardness in our setting more challenging.

The task in the previous work is defined as follows: given a set S of n length-d strings over an alphabet Σ, find a string s^* that minimizes the radius of the Hamming ball centered at s^* enclosing all strings in S. There are two major variants:

- the continuous closest string problem [2, 5, 12, 14, 20, 21, 24], where the string s^* can be chosen from anywhere in Σ^d, and
- the discrete closest string problem [1, 2], where the string s^* can be chosen only from S.

1.3 Paper Organization

The paper is organized as follows. In Section 2, we establish our results on 1-center spanning trees, proving Theorem 1. Next, in Section 3 and Section 4, we analyze the problem of finding a discrete 1-center among the extreme points of the shortest s,t-path polytope and the star subgraph polytope, thereby proving Theorem 2. In Section 5, we present an extremal bound on the number of vertices in a spanning tree polytope in three-dimensional space using 2-distance sets. Finally, in Section 6, we explore the relationship between treewidth and treeradius.

2 1-Center Spanning Tree

The graph under consideration is assumed to be connected, simple, and undirected. We use $G - F$ to denote the spanning subgraph of G with edge set $E(G) \setminus F$. For the difference $A \setminus B$ of two sets A and B, when B is a singleton, say $\{b\}$, we write $A - b$. The number of components of a graph H is denoted by $c(H)$.

Let G be a graph, and let $\Gamma(G)$ be the set of spanning trees of G. The 1-center spanning tree of G is a spanning tree with minimum *eccentricity*, where the eccentricity of a spanning tree T is defined as

$$\operatorname{ecc}(T) = \max_{T' \in \Gamma(G)} \left(|E(T) \setminus E(T')| + |E(T') \setminus E(T)| \right).$$

Notice that the eccentricity of a 1-center spanning tree is exactly twice the treeradius of the graph. We start with a simple but essential observation regarding the eccentricity of a spanning tree.

▶ **Proposition 3.** *Let G be a graph of order n, and let T be a spanning tree of G. Then* $\operatorname{ecc}(T) = 2(n-k)$*, where* $k = c(G - E(T))$.

Proof. Take $k-1$ edges from $E(T)$ to connect the k components of $G - E(T)$. Along with a spanning forest of $G - E(T)$, we have a spanning tree that shares exactly $k-1$ edges with T, and thus $\operatorname{ecc}(T) \geq 2(n-k)$. On the other hand, every spanning tree of G takes at least $k-1$ edges from $E(T)$ to connect the components of $G - E(T)$, implying $\operatorname{ecc}(T) \leq 2(n-k)$. ◀

▶ Remark 4. As an immediate result, computing the eccentricity of a spanning tree can be done in linear time.

A cut is a set of edges whose removal disconnects the graph[1], and a k-cut for any integer $k \geq 2$ is a cut whose removal results in at least k components. From Proposition 3 we know that the edge set of a 1-center spanning tree T is a cut that induces no cycle with $c(G-E(T))$ as large as possible. It is worth noting that every connected graph G contains a spanning tree T with $c(G - E(T)) \geq 2$. A natural relaxation is to ask how a cut that induces no cycle can be computed, with the resulting components as many as possible. A cut that induces no cycle is called *cycle-free*.

▶ **Observation 5.** *For $n \geq 2$, complete graph K_n has a cycle-free k-cut if and only if $k = 2$. Moreover, any subgraph of K_n induced by a cycle-free 2-cut is isomorphic to $K_{1,n-1}$.*

[1] In the literature, a cut typically refers to a partition of the vertices into two subsets. Here, we slightly override this definition to better align it with the notion of a k-cut. Specifically, a cut in our context refers to a superset of the standard cut-set, the set of edges with end-vertices in different subsets of the partition.

We relate the eccentricity of a spanning tree with a cycle-free cut as follows.

▶ **Proposition 6.** *Let G be a graph of order n. There is a spanning tree T with* $\mathrm{ecc}(T) \leq 2(n-k)$ *if and only if G has a cycle-free k-cut.*

Proof. By Proposition 3, a spanning tree T with $\mathrm{ecc}(T) \leq 2(n-k)$ is a k-cut. Thus the necessity follows. For sufficiency, let F be a cycle-free k-cut. Since F induces no cycle, there is a spanning tree T of G such that $F \subseteq E(T)$. It follows immediately that $k = c(G-F) \leq c(G-E(T))$. By Proposition 3, $\mathrm{ecc}(T) = 2\left(n - c(G - E(T))\right) \leq 2(n-k)$. ◀

By Propositions 3 and 6, we have the following equivalent definitions for a 1-center spanning tree.

▶ **Definition 7** (Restatements of 1-Center Spanning Trees)**.** *Let T be a spanning tree of a graph G. The following statements are equivalent:*

- **(i)** *T is a spanning tree with minimum eccentricity.*
- **(ii)** *T is a spanning tree that maximizes the number of components in $G - E(T)$.*
- **(iii)** *T is a spanning tree consisting of the edges in a cycle-free k-cut C with the largest possible k and the edges in a spanning tree of each component in $G - E(C)$.*

We adopt these alternative definitions in the following sections. Our first main result, Theorem 1, is developed based on

- Lemma 14, in Section 2.2, for the lower bound of $2^{\Omega(n)}$ on the time complexity, assuming ETH;
- Corollary 19, in Section 2.3, for the APX-completeness and the inapproximability factor.

The problems involved in the subsequent discussion, including the sources of hardness, are named below.

- CenterSpanningTree: Given a graph G, find a spanning tree of G with minimum eccentricity. The optimal eccentricity is denoted by $\mathsf{OPT}_{\mathrm{ecc}}$.
- CycleFreeCut: Given a graph G, find a cycle-free cut F that maximizes $c(G-F)$.
- CycleFreeCut-D: This is the decision variant of CycleFreeCut. Namely, for a given graph G and an integer k, determine whether there is a cycle-free k-cut of G.
- B-MaxIndepSet: Given a graph with maximum degree at most B, find an independent set of maximum size. The optimal size is denoted by $\mathsf{OPT}_{\mathsf{ind}}$.
- B-IndependentSet: Given a graph G with maximum degree at most B and an integer k, determine whether G contains an independent set of size k. We write IndependentSet if the graph under consideration has no degree constraint.

The following hardness results are known:

▶ **Theorem 8** (Johnson and Szegedy [18])**.** *There is no algorithm with running time $2^{o(n)}$ to solve 3-MaxIndepSet unless the Exponential Time Hypothesis (ETH) fails.*

▶ **Theorem 9** (Berman and Fujito [6])**.** *It is APX-hard to approximate 3-MaxIndepSet.*

▶ **Theorem 10** (Chlebík and Chlebíková [11])**.** *It is NP-hard to approximate 4-MaxIndepSet within a factor of $\frac{48}{47}$.*

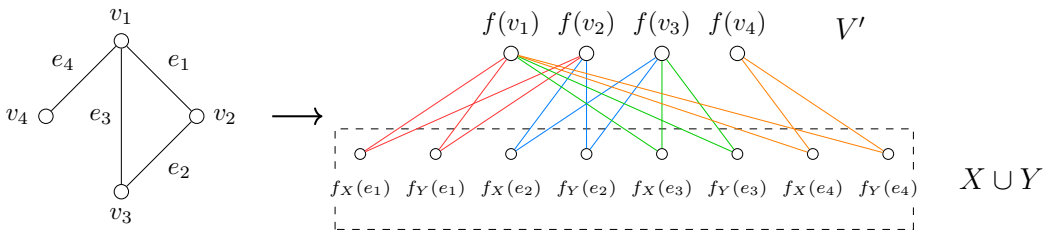


Figure 2 The reduction given in the proof of Theorem 11. The left is a graph G, which is the graph in an IndependentSet instance, while the right is the graph G' in the reduced instance of CycleFreeCut-D. The gadget for each edge of G is a 4-cycle in G'. The vertex set $X \cup Y$ forms a clique.

2.1 NP-Hardness

A *split graph* is a graph whose vertex set can be partitioned into two sets I and K, where I is an independent set and K is a clique.

▶ **Lemma 11.** *Given a graph G and a positive integer k, it is NP-hard to determine whether G has a cycle-free k-cut, even when G is a split graph.*

Proof. We reduce IndependentSet to CycleFreeCut-D. Precisely, for an instance (G, k) of IndependentSet, we construct an instance $(G', k+1)$ of CycleFreeCut-D as follows (see Fig. 2). The vertex set of G' is $V' \cup X \cup Y$, where $|V'| = |V(G)|$ and $|X| = |Y| = |E(G)|$. Each member of V' corresponds to a vertex of G, and each member of X or Y corresponds to an edge of G. We use the bijections $f\colon V(G) \to V'$, $f_X\colon E(G) \to X$, and $f_Y\colon E(G) \to Y$ to describe the correspondences, and for convenience let $g = f^{-1}$, $g_X = f_X^{-1}$, and $g_Y = f_Y^{-1}$. Two vertices a and b of G' are adjacent if

- a and b are incident in G; namely, one of them corresponds to an edge of G and the other corresponds to an endpoint of the edge, or
- $\{a, b\} \subseteq X \cup Y$.

We claim the following.

▷ Claim 12. G has an independent set of size at least k if and only if G' has a cycle-free $(k+1)$-cut.

For $k = 1$, the claim holds due to Observation 5 so we assume that $k \geq 2$. For necessity, let I be an independent set of size k and let $I' = f(I)$. Consider the set of edges $F = \{e \in E(G')\colon e \text{ has an endpoint in } I'\}$. Clearly, $G' - F$ contains at least $k+1$ components. Moreover, every member x in $X \cup Y$ has at most one neighbor in I' since otherwise the two neighbors a and b of x have the corresponding vertices $g(a)$ and $g(b)$ adjacent in G. Thus, there is no cycle between I' and $(V' \setminus I') \cup X \cup Y$, showing that F is cycle-free.

For sufficiency, let F be a cycle-free $(k+1)$-cut of G'. We show that

- $X \cup Y$ is contained in a component of $G' - F$, and
- the other components, each consisting of a single vertex, together correspond to an independent set of G.

First, $G' - F$ contains at least three components. If the clique $X \cup Y$ is not contained in a component, then by Observation 5 there exists $x \in X \cup Y$ with x in a component and $(X \cup Y) - x$ contained in another. Without loss of generality, assume that $x \in X$. The third

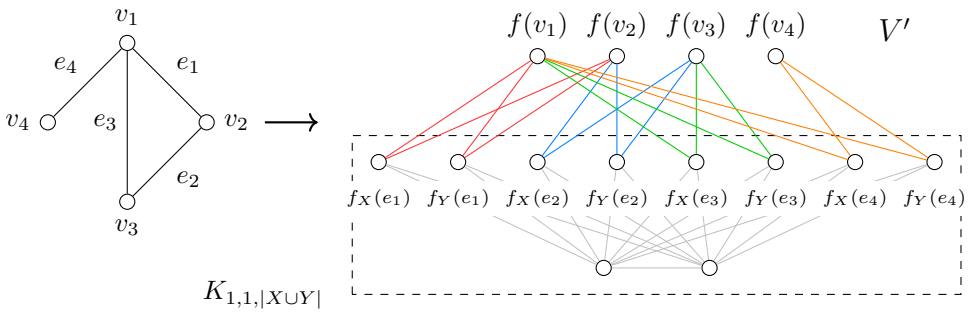


Figure 3 The reduction from IndependentSet to CycleFreeCut-D on sparse graphs.

component contains a vertex, say v', in V'. Let $v = g(v')$, $e = g_X(x)$, and $y = f_Y(e)$. If v is an endpoint of e, then x, y, and v' form a triangle contained in F. Otherwise, v is an endpoint of an edge e' of G. Let $x' = f_X(e')$ and $y' = f_Y(e')$. Then v', x', y', and x form a 4-cycle contained in F. Therefore, $X \cup Y$ is contained in a component of $G' - F$.

For the second, except the component containing $X \cup Y$, all components consist of vertices from V'. Since V' induces an independent set of G', each of these components consists of a single vertex. In addition, let u and v be the endpoints of an edge e in G, and let $u' = f(u)$, $v' = f(v)$, $x = f_X(e)$ and $y = f_Y(e)$ be the corresponding vertices in G'. If both u' and v' are in components not containing $X \cup Y$, then u', v', x and y form a 4-cycle, violating that F is cycle-free. Thus, the components not containing $X \cup Y$ correspond to an independent set of G of size at least k.

Since V' induces an independent set and $X \cup Y$ induces a clique, the graph G' is a split graph. Thus, the theorem follows. ◀

2.2 Exact 1-Center Spanning Trees in Sparse Graphs

Assuming ETH, solving 3-MaxIndepSet takes $2^{\Omega(n)}$ time, where n is the number of vertices of the input graph [18], as stated in Theorem 8. The same result can be obtained for CenterSpanningTree; that is, finding a 1-center spanning tree in a graph takes $2^{\Omega(N)}$ time, assuming ETH, where N is the number of vertices of the graph. This follows from the reduction given in Lemma 11 with the hardness source replaced with a 3-MaxIndepSet instance, making $N = O(n)$. Below, we strengthen the result, showing that the ETH lower bound of $2^{\Omega(n)}$ ($2^{\Omega(N)}$) still holds for finding a 1-center spanning tree in a sparse graph.

We modify the reduction given in the proof of Lemma 11. Recall that we reduce an independent set instance (G, k) to a cycle-free cut instance $(G', k+1)$, with $V(G') = V' \cup X \cup Y$ (see Fig. 2). Instead of making $X \cup Y$ a clique, we substitute it with a sparse structure, a complete 3-partite graph $K_{1,1,|X\cup Y|}$ (see Fig. 3), to ensure the sparsity of the whole graph. Like a complete graph, a complete 3-partite graph admits cycle-free k-cuts only if $k = 2$, as shown in Lemma 13. With this property, an argument similar to that given in the proof of Lemma 11 can be obtained. Details are given in the proof of Lemma 14.

▶ **Lemma 13.** *For $s \geq 3$ and $k \geq 3$, any complete s-partite graph has no cycle-free k-cut.*

Proof. Suppose to the contrary that there is a complete s-partite graph G that has a cycle-free k-cut F, for $k \geq 3$. Observe that there is a component of $G - F$ that intersects at least two partite sets since otherwise F induces a C_3. Let Q be such a component. We claim that there is a C_3 or a C_4 connecting Q and two other components Q' and Q''.

If Q intersects only two partite sets, we take Q' as a component that intersects another partite set, say U, and Q'' an arbitrary component other than Q and Q''. Let q' and q'' be two vertices from Q' and Q'', respectively, with $q' \in U$. If q' and q'' are adjacent, then there is a C_3 induced by q', q'', and a vertex in Q; otherwise, both q' and q' belong to U, and there is a C_4 formed by q', q'', and two vertices from different partite sets in Q.

If Q intersects three or more partite sets, Q' and Q'' can be arbitrary. Then an argument similar to the above applies. Consequently, F is not cycle-free in either case, which is a contradiction. ◀

▶ **Lemma 14.** *Assuming ETH, there is no $2^{o(N)}$-time algorithm to compute a 1-center spanning tree of an N-vertex sparse graph.*

Proof. We reduce 3-IndependentSet to CycleFreeCut-D. Let (G, k) be an instance of 3-IndependentSet. The reduced instance, $(G', k+2)$, is constructed like the reduction given in the proof of Lemma 11. The vertex set $V(G')$ is $V' \cup X \cup Y \cup \{a, b\}$, where $|V'| = |V(G)|$, $|X| = |Y| = |E(G)|$. The correspondences are the following bijections:

- $f\colon V(G) \to V'$,
- $f_X\colon E(G) \to X$, and
- $f_Y\colon E(G) \to Y$.

Also, for convenience, let $g = f^{-1}$, $g_X = f_X^{-1}$, and $g_Y = f_Y^{-1}$. The adjacency in G' is defined as follows. The subgraph induced by $X \cup Y \cup \{a, b\}$ is a complete 3-partite graph, with partite sets $\{a\}$, $\{b\}$, and $X \cup Y$. In addition, for every edge e of G with endpoints u and v, we make a 4-cycle $f(u) \to f_X(e) \to f(v) \to f_Y(e) \to f(u)$. See Fig. 3 for an illustration.

We claim that G has an independent set of size k if and only if G' has a cycle-free $(k+2)$-cut. For sufficiency, let F be a cycle-free $(k+2)$-cut. Then $G' - F$ has at least $k+2$ components. By Lemma 13, $X \cup Y \cup \{a, b\}$ intersects at most two components so at least k components consist of vertices from V'. Since V' induces an independent set of G', each of these components consists of a single vertex. Let u and v be two such components. There is no edge e of G with endpoints $g(u)$ and $g(v)$ since otherwise u, $f_X(e)$, v, and $f_Y(e)$ form a 4-cycle. Thus, the components consisting of vertices in V' correspond to an independent set of G, with size at least k.

For necessity, let I be an independent set of G with $|I| \geq k$. Let F be the cut of G' such that the components $G' - F$ are the subgraphs induced by the following vertex subsets:

- $\{f(u)\}$, for each $u \in I$,
- $(X \cup Y \cup \{a, b\} \cup (V' \setminus f(I))) - f_X(e)$, where e is an arbitrary edge of G, and
- $\{f_X(e)\}$.

Clearly, F is a $(k+2)$-cut. We show that F is cycle-free below. Let C be the component induced by $(X \cup Y \cup \{a, b\} \cup (V' \setminus f(I))) - f_X(e)$. Each vertex of C has at most one neighbor outside C, which implies that F induces no cycle passing through C. The remaining subgraph, induced by $f_X(e) \cup f(I)$, contains at most two edges so there is no cycle, either. Thus, F is a cycle-free $(k+2)$-cut.

The graph G' in the reduced instance is sparse. The reason is that it has $n + 2m + 2$ vertices and $8m + 1$ edges, where n and m are the numbers of vertices and edges of G, respectively. In addition, when the maximum degree of G is no more than three, we have $N = O(n)$, and by Theorem 8 the theorem follows. ◀

Since the number of spanning trees in an N-vertex sparse graph is of $2^{O(N)}$, the naive method of enumerating all spanning trees and keeping the one with minimum eccentricity is optimal for sparse graphs.

▶ **Corollary 15.** *To compute a 1-center spanning tree of a sparse graph G, the naive method of checking all spanning trees of G is optimal, assuming ETH.*

2.3 Approximation

We show that there is a PTAS reduction from B-MaxIndepSet to CenterSpanningTree, with the error parameter linearly preserved.

▶ **Lemma 16.** *Let G and G' be instances of B-MaxIndepSet and CenterSpanningTree, respectively, with G' reduced from G by the reduction given in the proof of Lemma 11. For $0 < \delta \leq (B^2+2B)^{-1}$ and $\varepsilon > 0$, given a spanning tree T of G' satisfying $\mathrm{ecc}(T) \leq (1+\delta\varepsilon)\mathsf{OPT}_{\mathrm{ecc}}$, an independent set I of G satisfying $|I| \geq (1-\varepsilon)\mathsf{OPT}_{\mathsf{ind}}$ can be computed in polynomial time.*

Proof. Recall that for a given graph G, the requested graph G' has $V(G') = V' \cup X \cup Y$, where there are one-to-one correspondences between $V(G)$ and V', $E(G)$ and X, and $E(G)$ and Y, respectively. Each edge of G corresponds to a 4-cycle in G', and $X \cup Y$ forms a clique (see Fig. 2). We show

- **(i)** how a spanning tree T of G' corresponds to an independent set I of G, and
- **(ii)** the error parameter is linearly preserved; i.e., if $\mathrm{ecc}(T) \leq (1+\delta\varepsilon)\mathsf{OPT}_{\mathrm{ecc}}$, then $|I| \geq (1-\varepsilon)\mathsf{OPT}_{\mathsf{ind}}$.

Let $N = |V(G')|$. For (i), we choose I to be the set of vertices that correspond to the single-vertex components of $G' - T$. Moreover, we claim the following.

▷ Claim 17. G has an independent set of size k if and only if G' has a spanning tree of eccentricity $2(N-(k+1))$.

Since every spanning tree is a cycle-free cut, the sufficiency follows immediately from Claim 12. For necessity, by Claim 12 there is a cycle-free $(k+1)$-cut, say F, with $c(G'-F) = k+1$. Specifically, there is a subset U' of V' forming k of the components, where every single vertex forms one. The remaining component, say C, is the subgraph induced by $X \cup Y \cup (V' \setminus U')$. Since $X \cup Y$ is a clique and every vertex of $V' \setminus U'$ has degree at least two in C, there is a spanning tree T_F of G' containing all edges in F without disconnecting C. By Proposition 3, $\mathrm{ecc}(T_F) = 2\,(N-(k+1))$. According to the analysis above, it can be further derived that

$$\mathrm{ecc}(T) = 2(N-(|I|+1)), \tag{1}$$

and

$$\mathsf{OPT}_{\mathrm{ecc}} = 2(N-(\mathsf{OPT}_{\mathsf{ind}}+1)). \tag{2}$$

For (ii), assume that $\mathrm{ecc}(T) \leq (1+\delta\varepsilon)\mathsf{OPT}_{\mathrm{ecc}}$. We claim that $|I| \geq (1-\varepsilon)\mathsf{OPT}_{\mathsf{ind}}$. Let $n = |V(G)|$. Since the maximum degree of G is upper bounded by B, we have

$$N = |V(G)| + 2|E(G)| \leq (B+1)n. \tag{3}$$

The chromatic number of a graph is upper bounded by the maximum degree plus one [9]. It follows that

$$\mathsf{OPT}_{\mathsf{ind}} \geq \frac{n}{B+1}. \tag{4}$$

Along with Eq. (2),

$$\frac{\mathsf{OPT}_{\mathsf{ind}}}{\mathsf{OPT}_{\mathrm{ecc}}/2} \underset{(4)(2)}{\geq} \frac{n/(B+1)}{N-\mathsf{OPT}_{\mathsf{ind}}-1} \underset{(3)}{\geq} \frac{n/(B+1)}{(B+1)n-n/(B+1)-1} = \frac{1}{B^2+2B-o(1)}. \quad (5)$$

Thus, we have

$$|I| \underset{(1)}{=} N - \frac{\mathrm{ecc}(T)}{2} - 1 \geq N - \frac{1}{2}(1+\delta\varepsilon)\mathsf{OPT}_{\mathrm{ecc}} - 1 \underset{(2)}{\geq} \mathsf{OPT}_{\mathsf{ind}} - \frac{\delta\varepsilon}{2}\mathsf{OPT}_{\mathrm{ecc}}$$
$$\underset{(5)}{\geq} \mathsf{OPT}_{\mathsf{ind}} - \left(B^2+2B\right)\delta\varepsilon\mathsf{OPT}_{\mathsf{ind}} \geq (1-\varepsilon)\mathsf{OPT}_{\mathsf{ind}}. \quad (6)$$

◀

▶ **Corollary 18.** CenterSpanningTree *is APX-complete even when the input is restricted to split graphs. Moreover, it is NP-hard to approximate within a factor of* 1153/1152.

Proof. The APX-hardness follows immediately from Lemma 16. For the inapproximability factor, by Theorem 10 we have that 4-MaxIndepSet is NP-hard to approximate within a factor of 48/47. Taking $B = 4$, $\delta = (B^2+2B)^{-1}$, and $\varepsilon = 1/48$ in Lemma 16, the corollary is established. ◀

Analogous to how we modify the proof of Lemma 11 to obtain Lemma 14, by replacing G' in Lemma 16 with the sparse graph in Lemma 14, we have the following results.

▶ **Corollary 19.** CenterSpanningTree *is APX-complete even when the input is restricted to sparse graphs. Moreover, it is NP-hard to approximate within a factor of* 1153/1152.

3 1-Center Shortest Path

Let G be an undirected simple graph, and let s and t be two distinct vertices of G. The set of shortest s,t-paths is denoted by $\mathcal{P}_{s,t}$. For a path P and two vertices u and v on P, the subpath running between u and v is denoted by $u \overset{\rightsquigarrow}{}_{P} v$; if u and v are adjacent on P, we write $u \to v$ when depicting the subpath. When referring to a path P, we use the same symbol P to denote the set of edges on the path. Analogous to the eccentricity of a spanning tree, the eccentricity of a shortest s,t-path P is defined to be

$$\mathrm{ecc}(P) = \max_{Q \in \mathcal{P}_{s,t}} (|P \setminus Q| + |Q \setminus P|).$$

A shortest s,t-path P^* is said to be *1-center* if

$$\mathrm{ecc}(P^*) = \min_{P \in \mathcal{P}_{s,t}} \mathrm{ecc}(P).$$

The *distance* between two vertices u and v is denoted by $d(u,v)$, which is the number of edges on a shortest u,v-path. For an edge with endpoints u and v on a shortest s,t-path, we denote it by uv if $d(s,u) < d(s,v)$.

Notice that the two conditions for the polytopes we consider are satisfied since a shortest s,t-path can be computed in polynomial time, whereas enumerating all shortest s,t-paths requires exponential time in the worst case.

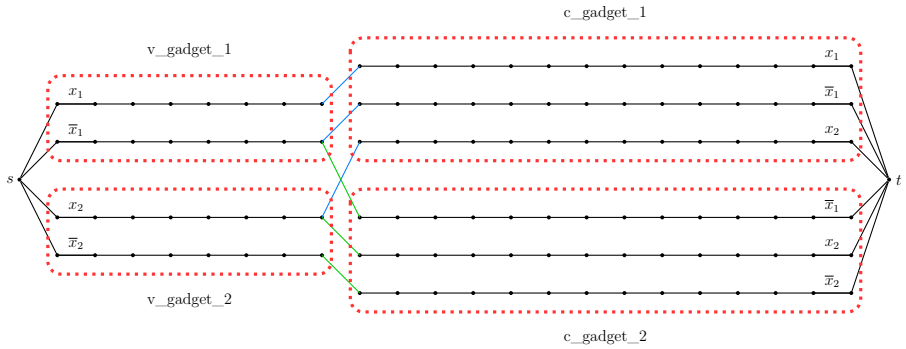


Figure 4 The variable gadgets and clause gadgets. The instance of the 3SAT has two variables x_1 and x_2, and two clauses c_1 and c_2. In this example, $c_1 = \{x_1, \overline{x_1}, x_2\}$ and $c_2 = \{\overline{x_1}, x_2, \overline{x_2}\}$. The upper and the lower paths in a variable gadget correspond to the positive and the negative literals of the variable, respectively.

3.1 NP-Hardness

A *double star* $S_{a,b}$, with $a \geq 2$ and $b \geq 2$, consists of two stars, $K_{1,a}$ and $K_{1,b}$, along with an additional edge, called the critical edge, connecting the two internal nodes. We distinguish the two stars, the $K_{1,a}$ and the $K_{1,b}$, of a double star as the left star and the right star, respectively. For two sets $\mathcal{P} = \{P_1, \ldots, P_p\}$ and $\mathcal{Q} = \{Q_1, \ldots, Q_q\}$ of disjoint paths, *bridging* $\mathcal{P}$ and $\mathcal{Q}$ using a double star $S_{p,q}$ is to identify the p leaves of the left star of $S_{p,q}$ with designated vertices on $P_1, \ldots, P_p$, respectively, and to identify the q leaves of the right star with designated vertices on $Q_1, \ldots, Q_q$, respectively. For two paths P and Q, we say that P *hits* Q if $P \cap Q \neq \emptyset$, and P *touches* Q if $|P \cap Q| = 1$.

▶ **Lemma 20.** *Given a graph G and two distinct vertices s and t, it is NP-hard to determine whether G contains a shortest s,t-path that hits all shortest s,t-paths in G; i.e., to determine if there is a path P such that $ecc(P) < 2d(s,t)$.*

Proof. We give a reduction from 3SAT. We assume that each clause contains exactly three literals and each variable appears in some clauses. Consider a 3SAT instance Φ with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$. In the reduced instance (G, s, t), there are

- two distinguished vertices, one for s and the other for t;
- n variable gadgets, $\mathsf{v_gadget}_1, \ldots, \mathsf{v_gadget}_n$, each consisting of two P_{4n}s;
- m clause gadgets, $\mathsf{c_gadget}_1, \ldots, \mathsf{c_gadget}_m$, each consisting of three P_{8m-2}s;
- an assignment gadget, consisting of $n + 2m - 1$ double stars; for each double star, the internal nodes u and v of the left and the right stars are connected to s and t, respectively, each with a path of specific length. We denote the s,u-path by $R_s(u)$ and the v,t-path by $R_t(v)$.

Each path in the variable gadgets and the clause gadgets is called a *literal path*. For the two literal paths in a variable gadget, one corresponds to the positive literal and the other the negative one. For the three literal paths in a clause gadget, there is a one-to-one correspondence to the literals in the clause. To ease the presentation, we embed each literal path in the plane horizontally, with a left end and a right end. Vertices on a literal path are numbered from left to right. Let $D(X, i) = \{v\colon\ v \text{ is the } i\text{th vertex on a path in gadget } X\}$, where X is a variable gadget or a clause gadget. The gadgets are interrelated as follows.

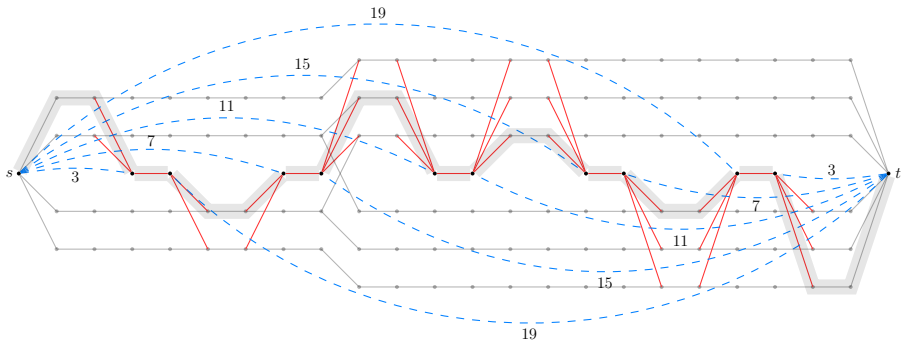


Figure 5 The assignment gadget. In this example, $n = 2$ and $m = 2$. There are five double stars, one $S_{2,2}$, one $S_{2,3}$, and three $S_{3,3}$s, induced by the red edges. The dotted arcs are the paths $R_s(u)$ and $R_v(v)$, for uv the critical edges. The number beside each arc indicates the length, i.e. the number of edges, of the path. The shaded path corresponds to the truth assignment $(x_1, x_2) = (\mathsf{true}, \mathsf{true})$. For each clause, the shaded path passes through two literal paths of the three, excluding one that corresponds to a true literal.

The left end of each literal path in a variable gadget is made adjacent to s; the right end of each literal path in a clause gadget is made adjacent to t; for each pair of literal paths (P, Q), with P in a variable gadget and Q in a clause gadget, if both P and Q correspond to an identical literal, concatenate these two paths by making the right end of P and the left end of Q adjacent. We call the resulting graph H. See Fig. 4 for an example.

Consecutive variable gadgets, consecutive clause gadgets, and the literal paths in a clause gadget are bridged using the double stars in the assignment gadget. Details are given below.

- Each of the first $n - 1$ double stars is an $S_{2,2}$, bridging a pair of consecutive variable gadgets. To bridge $\mathsf{v_gadget}_i$ and $\mathsf{v_gadget}_{i+1}$, for $i \in [n-1]$, the designated sets of vertices are $D(\mathsf{v_gadget}_i, 4i-2)$ and $D(\mathsf{v_gadget}_{i+1}, 4i+1)$.
- The nth double star is an $S_{2,3}$, bridging the nth variable gadget with the first clause gadget. The designated sets of vertices are $D(\mathsf{v_gadget}_n, 4n-2)$ and $D(\mathsf{c_gadget}_1, 1)$.
- The following $2m - 1$ double stars are $S_{3,3}$s. Except for the last one, every two of them bridge a clause gadget with itself, and the clause gadget with the next one; the last double star bridges the last clause gadget with itself only. For $i \in [m]$, to bridge $\mathsf{c_gadget}_i$ with itself, the designated sets of vertices are $D(\mathsf{c_gadget}_i, 8i-6)$ and $D(\mathsf{c_gadget}_i, 8i-3)$; to bridge $\mathsf{c_gadget}_i$ with $\mathsf{c_gadget}_{i+1}$, the designated sets of vertices are $D(\mathsf{c_gadget}_i, 8i-2)$ and $D(\mathsf{c_gadget}_{i+1}, 8i+1)$.

For a critical edge uv of a double star in an assignment gadget, the numbers of vertices on $R_s(u)$ and $R_t(v)$ are chosen so that the path $s \underset{R_s(u)}{\rightsquigarrow} u \to v \underset{R_t(v)}{\rightsquigarrow} t$ has length equal to a shortest s,t-path in H and remains a shortest s,t-path in G. This completes the construction of (G, s, t). See Fig. 5 for an example.

▶ **Observation 21.** *If a shortest s,t-path passes through no critical edges, then the subpath excluding s and t consists of two literal paths, one from a variable gadget and the other from a clause gadget. Moreover, the two literal paths correspond to the same literal.*

We now show that Φ is satisfiable if and only if there is a shortest s,t-path that hits all shortest s,t-paths in G. For necessity, given a satisfiable assignment, we consider a shortest s,t-path P such that

- P passes through all critical edges in the assignment gadget;
- P touches the path corresponding to a true literal of every variable gadget;
- P touches two of the three paths of every clause gadget, excluding an arbitrary one corresponding to a true literal in the clause.

If there is a path Q not hit by P, then Q contains no critical edges. By Observation 21, Q contains two disjoint subpaths corresponding to the same literal. However, the one from the variable gadget corresponds to a false literal, and the other, which is from a clause gadget, corresponds to a true literal. This leads to a contradiction, so P is the requested path. For sufficiency, observe that the path P contains all critical edges. In addition, P touches exactly one literal path in each variable gadget. Let A be the truth assignment such that true literals have their corresponding literal paths in the variable gadgets touched by P. We claim that A satisfies Φ. Suppose to the contrary that there is a clause that contains no true literal. Then, there is a literal whose corresponding literal paths in the variable gadget and the clause gadget are not hit by P, and so is the shortest s,t-path that passes through these two literal paths, a contradiction. ◀

▶ Remark 22. For a 3SAT instance of n variables and m clauses, the reduced instance in the proof above has $\Theta((n+m)^2)$ vertices. It follows that computing a 1-center shortest path cannot be done in $2^{o(\sqrt{n})}$ time, assuming ETH.

▶ **Corollary 23.** *Let G be an n-vertex graph, and let s and t be two vertices of G. Assuming ETH, there is no $2^{o(\sqrt{n})}$-time algorithm to compute a 1-center shortest s,t-path of G.*

3.2 An Exact Algorithm to Compute a 1-Center Shortest s,t-Path

A naive way to compute a 1-center shortest s,t-path is to enumerate all shortest s,t-paths, and keep the one with minimum eccentricity. The eccentricity of a given shortest s,t-path P can be computed in polynomial time by finding a shortest s,t-path that uses the edges on P as few as possible. Thus, this naive method takes $2^{O(n)}$ time. In the following, we aim for developing a $2^{\tilde{O}(\sqrt{n})}$-time algorithm. To achieve the reduced running time, we consider another way, a dynamic programming algorithm, to compute the eccentricity of a shortest s,t-path, say P. The vertex set of the graph is partitioned into layers, depending on the distances from s. For each vertex u in layer i, we record the least number of edges that an s,u-path shares with P. This value, which we call the intersection index of u with respect to P, can be computed recursively by finding an optimal path of the form $s \rightsquigarrow u' \to u$, where u' is some vertex in layer $i-1$.

To compute the eccentricity of a 1-center shortest s,t-path, it suffices to compute for every shortest s,t-path P the intersection index of t with respect to P and maintain the minimum. We treat the intersection indices of the vertices in a layer with respect to P as a function, called the *index function* of P to the layer. A key observation is that for any layer, the number of different index functions is upper bounded by $(d(s,t)+1)^{|V_i|}$. The idea of our algorithm is to distinguish the layers by their "thickness", the numbers of vertices in the layers. For thin layers, we maintain all possible index functions. On the other hand, we enumerate all subpaths passing through the thick layers and compute its eccentricity using the naive method mentioned above. These two parts can be combined and balanced so that the requested running time $2^{\tilde{O}(\sqrt{n})}$ is achieved.

Precisely, let $l = d(s,t)$, and let $V_i = \{v \in V(G) \colon d(s,v) = i, d(v,t) = l-i\}$ for $i \in \{0,\dots,l\}$. The *layered subgraph* H of G with respect to s and t is defined as

- $V(H) = V_0 \cup \cdots \cup V_l$ and
- $E(H) = \{uv \in E(G)\colon\ u \in V_{i-1}, v \in V_i, i \in [l]\}$.

Each V_i is called a *layer*. A path $u_0, \ldots, u_k$ is said to be *forward* if for all $i \in [k]$ there exists some $j \in [l]$ such that $u_{i-1} \in V_{j-1}$ and $u_i \in V_j$. Notice that there is a one-to-one correspondence between the set of shortest s,t-paths of G and that of forward s,t-paths of H. In the following we show how a 1-center forward s,t-path of H is computed.

A layer V_i is called *thin* if $|V_i| \leq \sqrt{n}$ and *thick* otherwise. Let S be an *exact hitting set* of the thick layers; namely for all $i \in [l]$

$$|S \cap V_i| = \begin{cases} 1, & \text{if } V_i \text{ is thick} \\ 0, & \text{otherwise.} \end{cases}$$

A forward path P is *S-governed* if every node u on P that belongs to a thick layer is in S. Let $\mathcal{P}_{u,v}$ be the set of all forward u,v-paths in H, and, for a given exact hitting set S of thick layers, let $\mathcal{S}_{s,t}$ be the set of S-governed forward s,t-paths in H.

For every possible S, we compute an S-governed forward s,t-path P^* with minimum eccentricity; namely, such a path P^* satisfies

$$\min_{Q \in \mathcal{P}_{s,t}} |P^* \cap Q| = \max_{P \in \mathcal{S}_{s,t}} \min_{Q \in \mathcal{P}_{s,t}} |P \cap Q|.$$

Let $U_0, \ldots, U_k$ be the thin layers. For each S-governed forward s,t-path P, we associate $k+1$ functions $g_i|_P\colon\ U_i \to \{0, \ldots, l\}$, the index function of P to layer U_i, for $i \in \{0, \ldots, k\}$, and k functions $h_i|_P\colon\ U_{i-1} \times U_i \to \{0, \ldots, l\}$, for $i \in [k]$, where

$$g_i|_P(v) = \min_{Q \in \mathcal{P}_{s,v}} |P \cap Q|$$

and

$$h_i|_P(u,v) = \begin{cases} \infty, & \text{if } \mathcal{P}_{u,v} = \emptyset \\ \min_{Q \in \mathcal{P}_{u,v}} |P \cap Q| & \text{otherwise.} \end{cases}$$

Notice that $g_0|_P = \mathbf{0}$, the zero function, and $g_k|_P(t) = \min_{Q \in \mathcal{P}_{s,t}} |P \cap Q|$. For $i \in [k]$, observe that

$$g_i|_P(v) = \min_{u \in U_{i-1}} \left(g_{i-1}|_P(u) + h_i|_P(u,v)\right). \tag{7}$$

Let

$$\mathcal{G}_i|_x = \{g_i|_P\colon\ P \in \mathcal{S}_{s,t}, P \text{ passes through } x, x \in U_i\}$$

and

$$\mathcal{H}_i|_{x,y} = \{h_i|_P\colon\ P \in \mathcal{S}_{s,t}, P \text{ passes through } x \text{ and } y, x \in U_{i-1}, y \in U_i\}.$$

▶ **Lemma 24.** *Let $f\colon\ U_i \to \{0, \ldots, l\}$ and $y \in U_i$. Then, $f \in \mathcal{G}_i|_y$ if and only if there exists $x \in U_{i-1}$, $g \in \mathcal{G}_{i-1}|_x$, and $h \in \mathcal{H}_i|_{x,y}$ such that*

$$f\colon\ v \mapsto \min_{u \in U_{i-1}} (g(u) + h(u,v)).$$

Proof. The necessity follows immediately from the definition. For sufficiency, we claim that there is an S-governed forward s,t-path P such that $g = g_{i-1}|_P$ and $h = h_i|_P$, and then by Eq. (7)

$$f(v) = \min_{u \in U_{i-1}} (g(u) + h(u,v)) = \min_{u \in U_{i-1}} (g_{i-1}|_P(u) + h_i|_P(u,v)) \underset{(7)}{=} g_i|_P(v),$$

for all $v \in U_i$. Such a path P can be constructed in a straightforward manner. Let $\{P', P''\} \subseteq \mathcal{S}_{s,t}$ such that $g = g_{i-1}|_{P'}$ and $h = h_i|_{P''}$. Clearly, the path $s \underset{P'}{\rightsquigarrow} x \underset{P''}{\rightsquigarrow} y \underset{P''}{\rightsquigarrow} t$ is the requested one. ◀

Algorithm 1 An exact algorithm to find a 1-center shortest s,t-path of a graph G.

Input: (G, s, t)
Output: The minimum eccentricity of a shortest s,t-path
$H \leftarrow$ the layered subgraph of G w.r.t s and t
for *each exact hitting set S of the thick layers* **do**
  for $x \in S$ **do**
    $\mathcal{G}_0|_x \leftarrow \{\mathbf{0}\}$
  for $i \leftarrow 1$ **to** k **do**
    for $x \in U_{i-1}, y \in U_i$ **do**
      compute $\mathcal{H}_i|_{x,y}$
    for $y \in U_i$ **do**
      $\mathcal{F} \leftarrow \emptyset$
      for $x \in U_{i-1}$ **do**
        for $g \in \mathcal{G}_{i-1}|_x, h \in \mathcal{H}_i|_{x,y}$ **do**
          for $v \in U_i$ **do**
            $f(v) \leftarrow \min_{u \in U_{i-1}} (g(u) + h(u,v))$
          $\mathcal{F} \leftarrow \mathcal{F} \cup \{f\}$
      $\mathcal{G}_i|_y \leftarrow \mathcal{F}$
return $\max_S \max_{g \in \mathcal{G}_k|_t} g(t)$

Time complexity. A thick layer has size greater than $\sqrt{n}$, so there are no more than $\sqrt{n}$ ones. Since each thick layer has size at most n, there are $O(n^{\sqrt{n}})$ different exact hitting sets of the thick layers. For each exact hitting set S of the thick layers, since all S-governed forward s,t-paths P coincide on S, there is exactly one function in $\mathcal{H}_i|_{x,y}$, given $x \in U_{i-1}$ and $y \in U_i$. This function can be computed in polynomial time by finding a forward x,y-path that uses the edges on P as little as possible. For $x \in U_{i-1}$, the set $\mathcal{G}_{i-1}|_x$ contains $O\left((l+1)^{\sqrt{n}}\right)$ functions so along with Lemma 24 computing $\mathcal{G}_i|_y$ for all $y \in U_i$ takes $O\left((|U_{i-1}| \cdot |U_i|)^2 \cdot (l+1)^{\sqrt{n}}\right)$ time. The overall running time is $O\left(n^{\sqrt{n}} \cdot \left(\sqrt{n} + \sum_{i=1}^{k} \left(\mathsf{poly}(n) + (|U_{i-1}| \cdot |U_i|)^2 \cdot (l+1)^{\sqrt{n}}\right)\right)\right) = 2^{\tilde{O}(\sqrt{n})}$. See Algorithm 1 for the pseudocode.

4 1-Center Star Subgraph S_r

We consider the problem of finding a discrete 1-center among the extreme points in the star subgraph polytope of an undirected graph. Given an undirected graph $G = (V, E)$ and an integer $r \geq 2$, one can identify all the star subgraphs with r leaves. Each star subgraph

S_r corresponds to an indicator vector $v(S_r) \in 2^E$, where the ith coordinate of $v(S_r)$ is 1 if the ith edge in E is included in S_r, and 0 otherwise. Placing the starting points of these indicator vectors at the origin in the space 2^E, the convex hull of their endpoints is the *star subgraph polytope*. Note that the indicator vector of each star subgraph S_r corresponds to an extreme point in the star subgraph polytope because every S_r contains exactly r edges.

The two conditions for the polytopes we considered are clearly satisfied since a star subgraph can be output in time linear to the input size, whereas enumerating all star subgraphs requires $\Omega(n^r)$ time in the worst case, which is superpolynomial for superconstant r.

We show that:

▶ **Lemma 25.** *It takes linear time to find a star subgraph S_r such that every other star subgraph S'_r can be obtained from S_r with removing k edges and subsequently adding k edges and k is minimized over all choices of star subgraphs with r leaves.*

Proof. Assume that $x \in V$ is the center node of the target S^*_r. Classify the star subgraph S_r other than S^*_r into the following three categories. Let $\deg(x)$ denotes the degree of node x in G.

- S_r shares the same center node with S^*_r. Thus, $\deg(x) \geq r$. Then the largest Hamming distance from any such S_r to S^*_r is $2r$ if $\deg(x) \geq 2r$ or $2\deg(x) - 2r$ otherwise. Let k_1 be the largest Hamming distance in this case.
- S_r has the center node y and $y \neq x$, and G does not contain $\{x, y\}$ as an edge. Thus, $\deg(x) \geq r$ and $\deg(y) \geq r$. Then the largest Hamming distance from any such S_r to S^*_r is $2r$. Let k_2 be the largest Hamming distance in this case.
- S_r has the center node y and $y \neq x$, and G contains $\{x, y\}$ as an edge. Thus, $deg(x) \geq r$ and $\deg(y) \geq r$. If $\deg(y) \geq r + 1$, then the largest Hamming distance from any such S_r to S^*_r is $2r$. Otherwise $\deg(y) = r$, such a center node y is the only chance that the choices of leaves of S^*_r matter. If there are more than r such center nodes y, then the largest Hamming distance from S^*_r to some star subgraph is $2r$. Otherwise, set k_3 as $2r - 1$ in this case.

Taking the maximum of k_1, k_2, and k_3 gives the desired k under the assumption that x is the center node of the target S^*_r. To find the optimal center, we iterate over all possible nodes as candidates. Since each category above requires $O(1)$ time, and the process is performed for each of the $O(m)$ edges in G, the total running time remains linear in the input size. ◀

Lemma 25 proves the results on star subgraph polytopes stated in Theorem 2.

5 An Extremal Bound

In this section, we show that the polytope depicted in Figure 1 contains the maximum number of vertices in three-dimensional space.

▶ **Lemma 26.** *Any spanning tree polytope that is embeddable in three-dimensional space contains at most four vertices.*

Proof. We say a point set P is a *k-distance set* if the number of distinct pairwise distances between points in P is exactly k. In three-dimensional space, any 1-distance set contains at most 4 points and any 2-distance set contains at most 6 points [22].

Let G be an undirected simple graph whose spanning tree polytope is embeddable in 3D. Then G cannot contain any of the following graphs as a subgraph.

- Any cycle of length $\ell \geq 5$. Here is why. Let H be a spanning pseudotree of G that contains the cycle C_ℓ. Then the number of vertices on the spanning tree polytope of H is ℓ and each pair of the vertices has distance 2. This forms a 1-distance set of $\ell \geq 5$ points, but every 1-distance set in 3D contains at most four points. Because these vertices is a subset of the vertices on the spanning tree polytope of G, then the spanning tree polytope of G cannot be embedded in 3D.
- Any pair of edge-disjoint simple cycles. Suppose that G contains a pair of edge-disjoint simple cycles C_ℓ and C_t for some $\ell, t \geq 3$. Let H be a connected spanning subgraph of G that contains only the two cycles C_ℓ and C_t. Then the number of vertices on the spanning tree polytope of H is ℓt and each pair of the vertices has distance either 2 or 4. This forms a 2-distance set of $\ell t \geq 9$ points, but every 2-distance set in 3D contains at most 6 points. By a similar reason, the spanning tree polytope of G cannot be embedded in 3D.
- The diamond graph. Let H be a connected spanning subgraph of G that contains only the three simple cycles on the diamond. Then the number of vertices on the spanning tree polytope of H is 8 and each pair of the vertices has distance either 2 or 4. This forms a 2-distance set of 8 points, but every 2-distance set in 3D contains at most 6 points. By a similar reason, the spanning tree polytope of G cannot be embedded in 3D.
- The $K_{2,3}$ complete bipartite graph. Let H be a connected spanning subgraph of G that contains only the three simple cycles on the $K_{2,3}$. Then the number of vertices on the spanning tree polytope of H is 12 and each pair of the vertices has distance either 2 or 4. This forms a 2-distance set of 12 points, but every 2-distance set in 3D contains at most 6 points. By a similar reason, the spanning tree polytope of G cannot be embedded in 3D.

Thus, to have the spanning tree polytope of G embeddable in 3D, G contains at most one cycle of length 3 or 4. The latter attains the claimed extremal bound. ◀

6 Concluding Remarks

As a final remark, we relate the two graph parameters: treeradius and treewidth, as stated in Theorem 28. Consequently, if a problem admits a fixed-parameter algorithm parameterized by treewidth, it also admits a fixed-parameter algorithm parameterized by treeradius.

▶ **Lemma 27** ([4]). *For any vertex v in a graph G, we have*

$$\text{treewidth}(G) \leq \text{treewidth}(G - v) + 1.$$

▶ **Theorem 28.** *For any graph G,*

$$\text{treewidth}(G) \leq 2 \cdot \text{treeradius}(G) + 1.$$

Proof. Let F be an edge subset that induces a forest in G, and let $\rho = n - c(G - F)$, where n is the number of vertices of G and $c(G - F)$ is the number of components of $G - F$. By Proposition 3, it suffices to show that $\text{treewidth}(G) \leq 2\rho + 1$. Denote by $C_1, \ldots, C_r$ the vertex sets of the *nontrivial components* of $G - F$, i.e., components of at least 2 vertices. Then

$$\rho = n - c(G - F) = \Big(c(G - F) - r + \sum_{i=1}^{r} |C_i|\Big) - c(G - F) = \sum_{i=1}^{r} \Big(|C_i| - 1\Big),$$

Since $|C_i| - 1 \geq 1$ for $i \in [r]$, we have $r \leq \rho$. Hence $\sum_{i=1}^{r} |C_i| = \rho + r \leq 2\rho$.

Let H be the graph obtained by removing all vertices contained in the nontrivial components of G. As the remaining vertices in H can only be joined by edges in F, the graph H is a forest and thus has treewidth 1. Applying Lemma 27 repeatedly yields $\text{treewidth}(G) \leq 2\rho + 1$. ◀

▶ Remark 29. Theorem 28 shows that if the treeradius of a graph is bounded then its treewidth is also bounded. Nonetheless, the converse does not hold in general. For example, as shown in Lemma 13, the complete tripartite graph $K_{1,1,n}$ has treeradius of n, which grows unboundedly with n, even though its treewidth is 2.

References

1 Amir Abboud, MohammadHossein Bateni, Vincent Cohen-Addad, Karthik C. S., and Saeed Seddighin. On complexity of 1-center in various metrics. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 275 of *LIPIcs*, pages 1:1–1:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.APPROX/RANDOM.2023.1.

2 Amir Abboud, Nick Fischer, Elazar Goldenberg, Karthik C. S., and Ron Safier. Can you solve closest string faster than exhaustive search? In *31st Annual European Symposium on Algorithms (ESA)*, volume 274 of *LIPIcs*, pages 3:1–3:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.3.

3 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.CH28.

4 Lowell W. Beineke, Robin J. Wilson, Ortrud R. Oellermann, Dirk Meierling, Lutz Volkmann, Matthias Kriesell, Kiyoshi Ando, R. J. Faudree, Michael Ferrara, Ronald J. Gould, Dieter Rautenbach, Bruce Reed, Ian Anderson, Keith Edwards, Graham Farr, Béla Bollobás, Oliver Riordan, F. T. Boesch, A. Satyanarayana, C. L. Suffel, Abdol-Hossein Esfahanian, R. A. Bailey, and Peter J. Cameron, editors. *Topics in Structural Graph Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge, UK, 2012.

5 Amir Ben-Dor, Giuseppe Lancia, Jennifer Perone, and R. Ravi. Banishing bias from consensus sequences. In *Combinatorial Pattern Matching, 8th Annual Symposium (CPM)*, volume 1264 of *Lecture Notes in Computer Science*, pages 247–261. Springer, 1997. doi:10.1007/3-540-63220-4_63.

6 Piotr Berman and Toshihiro Fujito. On approximation properties of the independent set problem for degree 3 graphs. In *Algorithms and Data Structures*, pages 449–460, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

7 Paul S. Bonsma. Spanning trees with many leaves in graphs with minimum degree three. *SIAM J. Discret. Math.*, 22(3):920–937, 2008. doi:10.1137/060664318.

8 Hajo Broersma and Xueliang Li. Spanning trees with many or few colors in edge-colored graphs. *Discuss. Math. Graph Theory*, 17(2):259–269, 1997. doi:10.7151/DMGT.1053.

9 R. L. Brooks. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37(2):194–197, April 1941.

10 Yi-Jun Chang, Martin Farach-Colton, Tsan-sheng Hsu, and Meng-Tsung Tsai. Streaming complexity of spanning tree computation. In *37th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 154 of *LIPIcs*, pages 34:1–34:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.STACS.2020.34.

11 Miroslav Chlebík and Janka Chlebíková. Inapproximability results for bounded variants of optimization problems. In *Fundamentals of Computation Theory, 14th International Symposium (FCT)*, volume 2751 of *Lecture Notes in Computer Science*, pages 27–38. Springer, 2003. doi:10.1007/978-3-540-45077-1_4.

12 Moti Frances and Ami Litman. On covering problems of codes. *Theory Comput. Syst.*, 30(2):113–119, 1997. doi:10.1007/S002240000044.

13 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

14 Leszek Gasieniec, Jesper Jansson, and Andrzej Lingas. Efficient approximation algorithms for the hamming center problem. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 905–906. ACM/SIAM, 1999. URL: http://dl.acm.org/citation.cfm?id=314500.315081.

15 R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

16 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.

17 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/JCSS.2001.1774.

18 David S. Johnson and Mario Szegedy. What are the least tractable instances of max independent set? In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, pages 927–928, USA, 1999. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=314500.315093.

19 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.

20 J. Kevin Lanctôt, Ming Li, Bin Ma, Shaojiu Wang, and Louxin Zhang. Distinguishing string selection problems. *Inf. Comput.*, 185(1):41–55, 2003. doi:10.1016/S0890-5401(03)00057-9.

21 Ming Li, Bin Ma, and Lusheng Wang. On the closest string and substring problems. *J. ACM*, 49(2):157–171, 2002. doi:10.1145/506147.506150.

22 Petr Lisonek. New maximal two-distance sets. *J. Comb. Theory A*, 77(2):318–338, 1997. doi:10.1006/JCTA.1997.2749.

23 Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

24 Nikola Stojanovic, Piotr Berman, Deborah Gumucio, Ross C. Hardison, and Webb Miller. A linear-time algorithm for the 1-mismatch problem. In *Algorithms and Data Structures, 5th International Workshop (WADS)*, volume 1272 of *Lecture Notes in Computer Science*, pages 126–135. Springer, 1997. doi:10.1007/3-540-63307-3_53.

25 Bang Ye Wu and Kun-Mao Chao. *Spanning trees and optimization problems*. Discrete mathematics and its applications. Chapman & Hall, 2004.

Streaming Algorithms for Conflict-Free Coloring

Rogers Mathew
Department of Computer Science and Engineering, IIT Hyderabad, India

Fahad Panolan
School of Computer Science, University of Leeds, UK

Seshikanth
Department of Computer Science and Engineering, IIT Hyderabad, India

Abstract

Conflict-free coloring of a hypergraph $\mathcal{H} = (V, \mathcal{E})$ using k colors is a function $f : V \to \{1, 2, \ldots, k\}$ such that for all $E \in \mathcal{E}$, there exists a vertex $v \in E$ with a unique color. That is, $f(v) \neq f(u)$ for all $u \in E \setminus \{v\}$. The minimum k for which $\mathcal{H}$ has a conflict-free coloring using k colors is called the *conflict-free chromatic number* of $\mathcal{H}$. For a simple graph G, a conflict-free coloring of the hypergraph with vertex set $V(G)$ and edge set being the set of all closed neighborhoods of the vertices in G is called a *conflict-free closed neighborhood (CFCN) coloring* of G. *CFCN chromatic number*, denoted by $\chi_{CN}(G)$, is the minimum number of colors used in a conflict-free closed neighborhood coloring of G. Analogously, we define conflict-free open neighborhood (CFON) coloring and CFON chromatic number, $\chi_{ON}(G)$, of a graph G.

There are various works on proving upper and lower bounds of $\chi_{ON}(G)$ and $\chi_{CN}(G)$. In this work, we develop streaming algorithms for CFCN and CFON coloring of a graph where the number of colors used matches the best-known upper bounds of $\chi_{ON}(G)$ and $\chi_{CN}(G)$. Our algorithms use as input an edge stream of the graph G in the insertion-only model. Our results and the best-known bounds for $\chi_{ON}(G)$ and $\chi_{CN}(G)$ are given below.

1. Pach and Tardos [Combinatorics, Probability and Computing, 2009] showed that, for any n vertex graph G, $\chi_{CN}(G) = O(\ln^2 n)$. Glebov, Szabó and Tardos [Combinatorics, Probability and Computing, 2014] showed the existence of graphs G with $\chi_{CN}(G) = \Omega(\ln^2 n)$. We design a randomized single-pass semi-streaming algorithm (i.e., it uses $O(n \ln n)$ space[1] that, given an n-vertex graph G, outputs a CFCN coloring of G using $O(\ln^2 n)$ colors with probability at least $(1 - \frac{2}{n})$.
2. Bhyravarapu, Kalyanasundaram, Mathew [Journal of Graph Theory, 2021] showed that for a graph G with maximum degree Δ, $\chi_{CN}(G) = O(\ln^2 \Delta)$. The methods used by our algorithms give rise to a simpler, alternate proof for this bound.
3. It is known that $\chi_{ON}(G) \leq 1/2 + \sqrt{2n + 1/4}$ (See Pach and Tardos [Combinatorics, Probability and Computing, 2009] and Ph.D. thesis of Cheilaris). This bound is asymptotically tight.
   - We design a deterministic single-pass $O(n\sqrt{n})$ space streaming algorithm that, given a graph G on n vertices, finds a CFON coloring using $2\sqrt{n}$ colors.
   - We design a randomized, single-pass, semi-streaming algorithm to find a CFON coloring of a graph G using $O(\sqrt{n} \ln^2 n)$ colors with success probability at least $(1 - \frac{2}{n})$.
4. It is known that $\chi_{ON}(G) \leq \Delta + 1$, where Δ is the maximum degree of a vertex in G. Further, there are graphs G known with $\chi_{ON}(G) = \Delta + 1$. We design a randomized two-pass semi-streaming algorithm (uses $O(\frac{1}{\epsilon^2} n \ln^3 n)$ space) that outputs a CFON coloring of G using $(1+\epsilon)\Delta$ colors, for any $\epsilon > 0$, with a probability at least $(1 - \frac{1}{n})$.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Streaming algorithm, conflict-free coloring, vertex coloring, randomized algorithms

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.44

[1] The space complexity is in terms of number of words.

© Rogers Mathew, Fahad Panolan, and Seshikanth;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Conflict-Free coloring (or CF coloring) of a hypergraph is a coloring of its vertices in which each hyperedge sees some color exactly once. The minimum number of colors required for the CF coloring of a hypergraph $\mathcal{H}$, denoted by $\chi_{CF}(\mathcal{H})$, is called the *conflict-free chromatic number* of $\mathcal{H}$. Motivated by its application in a frequency assignment problem in cellular networks, conflict-free coloring was introduced in 2002 by Even, Lokter, Ron, and Smorodinsky [17].

Cellular networks consist of base stations and mobile agents. Base stations, represented as vertices, are stationary and serve as the network's core, while mobile agents are clients served by base stations. Each base station operates on a fixed frequency, represented by a color. To establish a connection with a base station, an agent's device must tune itself to that base station's frequency. Agents can be within the range of multiple base stations. The range of communication for each agent is defined by a hyperedge, which represents the set of base stations it is capable of communicating with. To prevent interference, frequencies must be assigned to base stations in a conflict-free manner as described below. Every mobile agent should be able to find some base station in its vicinity that operates in a frequency that is distinct from that of all the other base stations in its vicinity. In combinatorial terms, this is about coloring the base stations in such a way that every agent sees a base station of a unique color in the hyperedge associated with it. While assigning n different frequencies to n base stations can solve the problem, it is expensive. Therefore, a scheme that allows the reusing of frequencies, whenever possible is preferred to minimize expenses.

Recently, conflict-free coloring and its variants were used in [19] to get improved results on a problem in coding theory, namely the Pliable Index Coding problem. Also, CF coloring has found applications in battery consumption analysis in sensor networks, in certain problems related to RFID protocols, in the vertex ranking problem (also known as ordered coloring, which finds applications in various fields, such as VLSI design and operations research), etc. See [14–19, 21, 23–25] for more details.

We define below two popular variants of the conflict-free coloring of graphs that have been extensively studied. Given a graph G, the *open neighborhood* of a vertex v in G, denoted by $N_G(v)$, is the set of vertices adjacent to v in G. The *closed neighborhood* of a vertex v in G, denoted by $N_G[v]$, is defined as $\{v\} \cup N_G(v)$. A coloring of the vertices of G using k colors is a *Conflict-Free Closed Neighborhood coloring* (*CFCN coloring*) if every vertex v sees some color exactly once in its closed neighborhood $N_G[v]$. The minimum k for which such a coloring exists is called the *Conflict-Free Closed Neighborhood chromatic number* (or *CFCN chromatic number*) of G. It is denoted by $\chi_{CN}(G)$. Analogously, one can define Conflict-Free Open Neighborhood coloring (CFON coloring) and Conflict-Free Open Neighborhood chromatic number (CFON chromatic number) of a graph by replacing "closed neighborhood" with "open neighborhood" in the above definitions. It is denoted by $\chi_{ON}(G)$. It is easy to see that for any graph G, $\chi_{CN}(G)$ is at most its chromatic number as in any proper coloring of G, every vertex v sees its color exactly once in its closed neighborhood $N_G[v]$. However, there are graphs G for which $\chi_{ON}(G)$ is arbitrarily larger than its chromatic number. For any positive integer r, Example 4 gives bipartite graphs (and hence its CFCN chromatic number is at most 2) with CFON chromatic number at least r. It is known that for any graph G with maximum degree Δ, (i) $\chi_{CN}(G) = O(\ln^2 \Delta)$ (see [12]), and (ii) $\chi_{ON}(G) \leq \Delta + 1$ (see [23]). Both these bounds are asymptotically tight (see Section 2 for more details).

In this paper, we focus on semi-streaming algorithms (Refer [22] for more information on graph streams) for conflict-free coloring of graphs. We consider only graph streams that are insert-only. Below, we define an *insert-only graph stream*. We consider streams consisting of a sequence of undirected edges $e = (u, v)$, where $u, v \in [n] = \{1, 2, \ldots, n\}$. Such a stream, $S = \langle e_1, e_2, \ldots, e_m \rangle$ naturally defines an undirected graph $G = (V, E)$, where $V = [n]$ and $E = \{e_1, \ldots, e_m\}$. We assume the stream elements are distinct, and therefore, the resulting graph is a simple graph. An algorithm for graphs is a *semi-streaming graph algorithm* if it takes as input a graph stream and does all the computations using $O(n \cdot \text{polylog}(n))$ bits of space, where n is the number of vertices of the graph. The efficiency of a graph algorithm in a streaming model is measured by the space it uses, the time it requires to process each edge, and the number of passes it makes over the graph stream. Extensive research has been conducted on semi-streaming algorithms for proper coloring of graphs, as demonstrated in several notable works [1, 3–5, 8, 9]. Similarly, edge coloring has been thoroughly explored in the context of streaming algorithms, with notable contributions found in [2, 7]. However, to the best of our knowledge, there is no streaming algorithm known for conflict-free coloring of graphs. We describe our results and methods in the section below.

2 Our contributions

First, we state a few results from the literature. Then we explain our results and briefly describe the key ideas involved. Please note that the space used by the algorithms mentioned below is in terms of the number of words.

Auxiliary results from the literature. The following results on conflict-free coloring are due to Pach and Tardos [23].

▶ **Proposition 1** (Theorem 1.1 in [23]). *Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph, where every vertex is present in at most Δ hyperedges. Then $\chi_{CF}(\mathcal{H}) \leq \Delta + 1$. Moreover, there is a deterministic $O(n\Delta)$ time (and hence $O(n\Delta)$ space) algorithm that obtains such a coloring.*

▶ **Proposition 2** (Theorem 1.2 in [23]). *For any positive integers t and Γ, the conflict-free chromatic number of any hypergraph in which each edge is of size at least $2t - 1$ and each edge intersects at most Γ others is $O(t\Gamma^{\frac{1}{t}} \ln \Gamma)$.*

As a corollary of Proposition 2, it is proved that $\chi_{ON}(G) \leq \frac{1}{2} + \sqrt{2n + \frac{1}{4}}$. Cheilaris gave an alternate proof to to this result [14].

▶ **Proposition 3** (Proposition 4.40 in [14]). *For every graph G, $\chi_{ON}(G) \leq 2\sqrt{n}$.*

From Example 4, we infer that the bound in Proposition 3 is asymptotically tight.

▶ **Example 4.** Let K_r^* denote the graph obtained from a complete graph on r vertices by subdividing every edge exactly once. This graph with $\binom{r}{2} + r$ vertices and with a maximum degree of $r - 1$ is known to have a CFON chromatic number of r. Here, subdividing an edge uv means introducing a new vertex w and replacing the edge uv with edges uw and wv.

Our methods and results. Pach and Tardos [23] showed that, for any graph G, $\chi_{CN}(G) = O(\ln^2 n)$ where n is the number of vertices in G. Glebov, Szabó and Tardos [18] showed the existence of graphs G with $\chi_{CN}(G) = \Omega(\ln^2 n)$.

Result 1: There is a randomized, single-pass, $O(n \ln n)$ space streaming algorithm that, given an n-vertex graph G, outputs a CFCN coloring of G using $O(\ln^2 n)$ colors with probability at least $(1 - \frac{2}{n})$.

Our algorithm partitions the vertex set into two parts: Part A, which is made of vertices of degree at least $2c \ln n$ for some constant c, and Part B, which is the set of all the remaining vertices (which are of degree less than $2c \ln n$). Part A is further partitioned into two parts, namely A' and A''. The set A'' is made of all vertices in A that have no neighbor in B. We choose a random coloring for the vertices in A from a geometric distribution. Then, we will show that, with high probability, every vertex in A'' sees some color exactly once in its open and closed neighborhood.

Next, we construct a hypergraph $\mathcal{H} = (B, \mathcal{E})$, where $\mathcal{E} = \{N_G[v] \cap B \ : \ v \in A' \cup B\} \cup \{N_G(v) \cap B \ : \ v \in A' \cup B, N_G(v) \cap B \neq \emptyset)\}$. Observe that the maximum degree of $\mathcal{H}$ is at most $2c \ln n$ and hence can be CF colored by the greedy procedure given in Proposition 1 using $2c \ln n + 1$ colors. Finally, we observe that the union of the two colorings explained above (let this coloring be c) is a CFCN coloring of G and obtain Result 1. Moreover, we observe that for many vertices, there is a unique color in its open neighborhood as well. More specifically, let $B_1 = \{b \in B : N_G(b) \cap B = \emptyset\}$. We prove that for any vertex $v \in V(G) \setminus B_1$, there is a unique colored vertex in $N_G(v)$. Also, notice that for all $b \in B_1$, $|N_G(b)| = O(\ln n)$ and $N_G(b) \subseteq A'$. So in the semi-streaming algorithm, we store the neighborhood information of the vertices in B_1 and obtain a CFON coloring c' of the graph induced on $A' \cup B_1$ using at most $O(\sqrt{n})$ colors. Then, the coloring c_o defined as $c_o(v) = (c(v), c'(v))$ for all v, is a CFON coloring of G using $O(\sqrt{n} \ln^2 n)$ colors. Here, for convenience, assume that $c'(v) = 1$ if $v \notin A' \cup B_1$. This gives us the following result on CFON coloring.

Result 2: There is a randomized, single-pass, semi-streaming algorithm that, given a graph G on n vertices, finds a CFON coloring using $O(\sqrt{n} \ln^2 n)$ colors and in polynomial time with probability at least $(1 - \frac{2}{n})$.

From Proposition 3, we know that $\chi_{ON}(G) \leq 2\sqrt{n}$. This bound is asymptotically tight due to Example 4. Our next result, whose proof has been moved to the appendix due to the paucity of space, is the following.

Result 3: There is a deterministic, single-pass, $O(n\sqrt{n})$ space streaming algorithm that, given a graph G on n vertices, finds a CFON coloring using $2\sqrt{n}$ colors.

For a graph G, with maximum degree Δ, Pach and Tardos [23] in the year 2009 showed that $\chi_{CN}(G) = O(\ln^{2+\epsilon} \Delta)$ for any $\epsilon > 0$. As mentioned above, in 2014, Glebov, Szabó, and Tardos [18] proved that there exist graphs G on n vertices with $\chi_{CN}(G) = \Omega(\ln^2 n)$ (and thereby $\Omega(\ln^2 \Delta)$). In the year 2021, Bhyravarapu, Kalyanasundaram, and Mathew [12] tightened the upper bound to show that $\chi_{CN}(G) = O(\ln^2 \Delta)$. The methods we use in proving Result 1 give us an alternate, shorter proof (which is given below).

Proof of $\chi_{CN}(G) = O(\ln^2 \Delta)$. We partition $V(G) = A \uplus B$, where $A = \{v \in V(G) : deg_G(v) \geq 2t - 1\}$, $B = V(G) \setminus A$ and $t = \ln \Delta$. Further, let $A = A_1 \uplus A_2$ where $A_1 = \{a \in A : (N_G(a) \cap B) \neq \emptyset\}$ and $A_2 = A \setminus A_1$. We define a hypergraph $\mathcal{H}_2 = (A, \mathcal{E}_2)$ where $\mathcal{E}_2 = \{N_G[a] : a \in A_2\}$. From the definition of $A_2, N_G[a] \subseteq A, \forall a \in A_2$. Applying Proposition 2

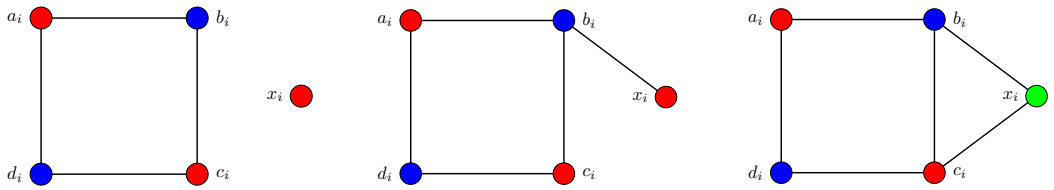
with $t = \ln \Delta, \Gamma = \Delta^2$, we get $\chi_{CN}(\mathcal{H}_2) = O(\log^2 \Delta)$. This coloring of the vertices of A ensures that every vertex in A_2 sees some color exactly once in its closed neighborhood in G. We define another hypergraph $\mathcal{H}_1 = (B, \mathcal{E}_1), \mathcal{E}_1 = \{N_G[v] \cap B : v \in A_1 \cup B\}$. Since the degree of a vertex in B in graph G is at most $2 \ln \Delta - 2$, we can conclude that every vertex in $\mathcal{H}_1$ is present in at most $2 \ln \Delta - 1$ hyperedges. Applying Proposition 1 on $\mathcal{H}_1$, we get $\chi_{CN}(\mathcal{H}_1) \leq 2 \ln \Delta$. We make sure that the set of colors used to color $\mathcal{H}_2$ is disjoint from the set of colors used to color $\mathcal{H}_1$. This coloring of the vertices of B ensures that every vertex in $A_1 \uplus B$ sees some color exactly once in its closed neighborhood in G. ◀

It is known that $\chi_{ON}(G) \leq \Delta + 1$, where Δ is the maximum degree of a vertex in G (follows from Proposition 1). From Example 4, we know that this bound is tight. Our next result is the following.

Result 4: There is a two-pass semi-streaming algorithm (uses $O(\frac{1}{\epsilon^2} n \ln^3 n)$ space) that outputs a CFON coloring of G using $(1 + \epsilon)\Delta$ colors, for any $\epsilon > 0$, with probability at least $(1 - \frac{1}{n})$.

In our algorithm of Result 4, we use the pallete sparsification lemma of Assadi, Chen, and Khanna [4]. For each vertex $v \in V(G)$, we sample a set $L(v)$ of $O(\frac{1}{\epsilon} \ln n)$ colors from $[(1 + \epsilon)\Delta]$. In the first pass of the stream, for each vertex v, we identify a special vertex $s(v)$ that will get a unique color among $N_G(v)$ in our coloring. In the second pass, using the list of colors and special vertices, we find a sketch H of G of size $O(\frac{1}{\epsilon^2} n \ln^2 n)$. Finally, we do a greedy procedure on H to get a CFON coloring.

We observe that any $O(1)$-pass streaming algorithm that determines if the CFCN (CFON) chromatic number of an input graph is at most 2 or not, requires $\Omega(n)$ space. The proofs of both the results are by reductions from the two player communication problem Disjointness (refer [20]). In this problem, Alice and Bob are given two strings of length n (say the strings given to them are $p_1 \dots p_n$ and $q_1 \dots q_n$, respectively), and they are disjoint if there is no $i \in [n]$ such that $p_i = 1$ and $q_i = 1$. It is known that any protocol (even randomized) solving Disjointness requires at least $\Omega(n)$ space.



$p_i = 0$, $q_i = 0$, $\chi_{\text{CN}} = 2$.

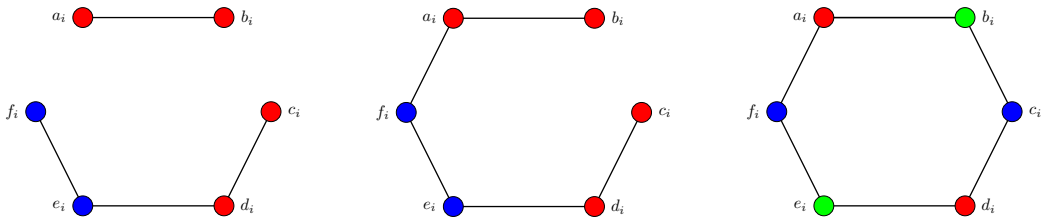
$p_i = 1$, $q_i = 0$, $\chi_{\text{CN}} = 2$. We get similar graph for $p_i = 0$, $q_i = 1$.

$p_i = 1$, $q_i = 1$, $\chi_{\text{CN}} = 3$.

Figure 1 An illustration of the cases in the construction of the graph $G(I)$. The CFCN chromatic number of the rightmost graph is 3, and the CFCN chromatic number of the other two graphs is 2.

Proof of lower bound for CFCN. Consider an instance I of the Disjointness Problem where Alice is given an n-bit string $p_1 \dots p_n$ and Bob is given an n-bit string $q_1 \dots q_n$. Together they construct a graph $G(I)$ on $5n$ vertices whose vertices are a_i, b_i, c_i, d_i, x_i, where $1 \leq i \leq n$. For every i, Alice adds the edges $a_i b_i$, $b_i c_i$, $c_i d_i$, $d_i a_i$ to form n 4-cycles. Further, for every i, (i) if $p_i = 1$, then Alice adds the edge $b_i x_i$, and (ii) if $q_i = 1$, then Bob adds the

edge $c_i x_i$. For each i, they create one or two components of $G(I)$ which is isomorphic to one of the graphs in Figure 1. The graph when both the corresponding bits intersect (i.e., $p_i = q_i = 1$) require at least 3 colors for any CFCN coloring. It can be verified that the graph $G(I)$ thus constructed has its CFCN chromatic number at most 2 if and only if I is a YES instance of the DISJOINTNESS problem. ◀



$p_i = 0$, $q_i = 0$, $\chi_{\text{ON}} = 2$.

$p_i = 1$, $q_i = 0$, $\chi_{\text{ON}} = 2$. We get similar graph for $p_i = 0$, $q_i = 1$.

$p_i = 1$, $q_i = 1$, $\chi_{\text{ON}} = 3$.

Figure 2 An illustration of the cases in the construction of the graph $H(I)$. The CFON chromatic number of the rightmost graph is 3, and the CFON chromatic number of the other two graphs is 2.

Proof of lower bound for CFON. For this purpose, given an instance I, Alice and Bob together construct a graph $H(I)$ on $6n$ vertices, namely $a_i, b_i, c_i, d_i, e_i, f_i$, $1 \leq i \leq n$, as described below: For every i, Alice adds the edges $a_i b_i, c_i d_i, d_i e_i, e_i f_i$. Further, for every i, (i) if $p_i = 1$, then Alice adds the edge $a_i f_i$, and (ii) if $q_i = 1$, the Bob adds the edge $b_i c_i$. If there exists i such that $p_i = q_i = 1$, then $H(I)$ has a connected component which is a cycle of length 6 and we need at least 3 colors in any CFON coloring of it. Otherwise each connected component of $H(I)$ is a path and its CFON chromatic number is at most 2. See Figure 2 for an illustration. ◀

3 Semi-streaming algorithms for CFCN and CFON colorings

In this section, we prove the following theorems.

▶ **Theorem 5.** *There exists a randomized, single-pass, semi-streaming algorithm that, given a graph G with n vertices, with probability at least $\left(1 - \frac{2}{n}\right)$, finds a CFCN coloring using $O(\ln^2 n)$ colors, using $O(n \ln n)$ space and in polynomial time.*

▶ **Theorem 6.** *There exists a randomized, single-pass, semi-streaming algorithm that, given a graph G on n vertices, with probability at least $(1 - \frac{2}{n})$, finds a CFON coloring using $O(\sqrt{n} \ln^2 n)$ colors, using $O(n \ln n)$ space and in polynomial time.*

Towards that, we first prove the following lemma about conflict-free coloring. The bound obtained on the number of colors used to do a conflict-free coloring of the hypergraph defined in this lemma can be obtained by directly substituting the values $t = 2 \ln r$ and $\Gamma = 2n$ in the statement of Proposition 2 that was proved in [23]. However, Proposition 2 would only ensure that such a coloring exists with positive probability. In the lemma below, we modify the proof of this proposition to obtain such a coloring with high probability. For the sake of completeness, we provide the proof of this lemma in the appendix.

▶ **Lemma 7** (⋆). [2] *Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph with $|V| = n$, $|\mathcal{E}| \leq 2n$, and $|E| \geq 4 \ln r - 1$ for all $E \in \mathcal{E}$, where $r \geq max\{7, n\}$ is a positive integer. Then, Algorithm 1, which takes as input V runs in $O(n \ln^2 r)$ time, uses $O(n)$ space and with probability greater than $\left(1 - \frac{2}{r}\right)$ returns a conflict-free coloring of $\mathcal{H}$ that uses at most $\lceil 16e \ln^2 r \rceil$ colors.*

Algorithm 1 Randomized Algorithm for CF coloring of $\mathcal{H}$.

Input: The vertex set of hypergraph $\mathcal{H} = (V, \mathcal{E})$, where each hyperedge E satisfies $|E| \geq 4 \ln r - 1, \forall E \in \mathcal{E}$, and $r \geq n = |V|$.
Output: A CF coloring function $c : V \to [\lceil 16e \ln^2 r \rceil]$

```
Let T = ⌈16e ln² r⌉ and let c(v) = 0, ∀v ∈ V
Set i := 0
while i < T do
    i := i + 1;
    for each vertex v ∈ V with c(v) = 0 do
        Independently toss a coin that gives Head with probability 1/(8e ln r)
        if the outcome is Head then
            c(v) := i
return c
```

Next, we prove the following lemma that gives a coloring such that a subset of the vertices with "large" degree gets a unique color in its open and closed neighborhoods.

▶ **Lemma 8.** *Let G_1 be a graph on n_1 vertices, where $V(G_1) = X \uplus Y$. Suppose for all $x \in X, deg_{G_1}(x) \geq 4 \ln r_1 - 1$, where r_1 is a positive integer with $r_1 \geq max\{n_1, 7\}$. Let c_1 be the coloring $c_1 : V(G_1) \to [\lceil 16e \ln^2 r_1 \rceil]$ obtained by invoking Algorithm 1 with $V(G_1)$ as input. Then, with probability $1 - \frac{2}{r_1}$, every vertex in X sees some color exactly once in its open and closed neighborhoods.*

Proof. Proof follows from the application of Lemma 7 with $\mathcal{H}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ defined as $\mathcal{V}_1 = V(G_1)$ and $\mathcal{E}_1 = \{N_{G_1}[v] : v \in X\} \cup \{N_{G_1}(v) : v \in X\}$. ◀

▶ **Lemma 9.** *Let $r_2 \in \mathbb{Z}^+$ and let G_2 be a graph on n_2 vertices, where $V(G_2) = X \uplus Y$ with the following properties: (i) X is an independent set, (ii) for all $x \in X, N_{G_2}(x) \cap Y \neq \emptyset$, and (iii) for all $y \in Y, deg_{G_2}(y) < 4\lceil \ln r_2 \rceil - 1$. Then, Algorithm 2 is a greedy, deterministic algorithm that runs in $O(n_2 \ln r_2)$ time and $O(n_2 \ln r_2)$ space, and outputs a coloring of vertices in Y using at most $(8\lceil \ln r_2 \rceil)$ colors with the following properties.*

(a) *Every vertex in $V(G_2)$ sees some color exactly once in its closed neighborhood.*
(b) *Every vertex $v \in V(G_2)$ with $N_{G_2}(v) \cap Y \neq \emptyset$ sees some color exactly once in its open neighborhood.*

Proof. The hypergraph $\mathcal{H}_2$ constructed in step 1 of Algorithm 2 takes $O(n_2 \ln r_2)$ space because the degree of this hypergraph is at most $8\lceil \ln r_2 \rceil$. It is easy to see that a conflict-free coloring of $\mathcal{H}_2$ will ensure properties (a) and (b). Proposition 1 gives a conflict-free coloring of $\mathcal{H}_2$ in $O(n_2 \ln r_2)$ time and $O(n_2 \ln r_2)$ space using at most $8\lceil \ln r_2 \rceil$ colors. ◀

[2] Proof of results marked with ⋆ are deferred to the appendix.

Algorithm 2 Deterministic Algorithm for conflict-free coloring of G_2.

Input: The vertex set and edge set of the graph G_2 with
1. Partition $V(G_2) = X \uplus Y$, where X is an independent set,
2. $\forall x \in X, N_{G_2}(x) \cap Y \neq \emptyset$ and
3. $\forall y \in Y, \deg_{G_2}(y) < (4\lceil \ln r_2 \rceil - 1)$

Output: A coloring function $c_2 : Y \to [8\lceil \ln r_2 \rceil]$ that satisfies properties (a) and (b) mentioned in Lemma 9.

1 Construct a hypergraph $\mathcal{H}_2 = (Y, \mathcal{E})$ with

$$\mathcal{E} = \{N_{G_2}[v] \cap Y : v \in V(G_2)\} \cup \{N_{G_2}(v) \cap Y : v \in V(G_2), N_{G_2}(v) \cap Y \neq \emptyset\}.$$

Note that the degree of this hypergraph is at most $8\lceil \ln r_2 \rceil$ because for all $y \in Y$, $\deg_{G_2}(y) < 4\lceil \ln r_2 \rceil$. Use the algorithm given in Proposition 1 to get a coloring $c_2 : Y \to 8[\lceil \ln r_2 \rceil]$ of the vertices of $\mathcal{H}_2$ such that every hyperedge sees some color exactly once.

2 Return the coloring c_2.

▶ **Lemma 10.** *Let G be a graph on n vertices, where*
- *$V(G) = A \uplus B$, $A = A' \uplus A''$, $A' = \{a \in A : N_G(a) \cap B \neq \emptyset\}, A'' = A \setminus A'$,*
- *for all $a \in A$, $deg_G(a) \geq (4\lceil \ln n \rceil - 1)$, and*
- *for all $b \in B$, $deg_G(b) < (4\lceil \ln n \rceil - 1)$.*

Algorithm 3 takes as input (i) the sets A', A'', B, (ii) $N_G(a') \cap B$ for all $a' \in A'$, and (iii) $N_G(b)$ for all $b \in B$, runs in $O(n \cdot \ln^2 n)$ time, uses $O(n \ln n)$ space and gives a coloring c of $V(G)$ using $O(\ln^2 n)$ colors such that with probability greater than $\left(1 - \frac{2}{n}\right)$ the following holds.

(i) *c is a CFCN coloring of G.*

(ii) *For every $v \in V(G) \setminus \{b \in B : N_G(b) \cap B = \emptyset\}$, v sees some color exactly once in its open neighbourhood.*

Proof. We apply Lemma 8 with the following inputs: $G_1 = (A, E_1)$, where $E_1 = \{(u, v) : u, v \in A\}$, $X = A''$, $Y = A'$, and $r_1 = n$. The algorithm mentioned in Lemma 7, runs in $O(n \ln^2 n)$ time, as $n_1 \leq r_1 = n$. It uses $O(n)$ space and outputs the coloring function $c_1 : A \to [16\lceil e \ln^2 n \rceil]$. With probability greater than $(1 - \frac{2}{n})$, every vertex in $A''(= X)$ will see some color exactly once in its open and closed neighborhoods under the coloring c_1 (as well as in c).

Next, we invoke Algorithm 2 with the following inputs: the graph $G_2 = (A' \uplus B, E_2)$, where $X = A'$, $Y = B$, $r_2 = n$, and $E_2 = E(G) \setminus E_1$. It is important to note that (i) G_2 is well-defined as every edge in E_2 has both its endpoints in $A' \cup B$, and (ii) A' is an independent set in G_2. According to Lemma 9, Algorithm 2 outputs the coloring $c_2 : B \to \{16\lceil e \ln^2 n \rceil + 1, \ldots, 16\lceil e \ln^2 n \rceil + 8\lceil \ln n \rceil\}$ in $O(n \ln n)$ time, using $O(n \ln n)$ space, since $n_2 \leq r_2 = n$. By property (a) of Lemma 9, every vertex in $A' \cup B$ sees some color exactly once in its closed neighborhood under the coloring c_2 (as well as in c). Therefore, we have proved that c is a CFCN coloring of G.

Furthermore, based on the reasoning above, our algorithm takes $O(n \ln^2 n)$ time and $O(n \ln n)$ space. We have already proved that with probability greater than $(1 - \frac{2}{n})$, every vertex in A'' will see some color exactly once in its open neighborhoods under the coloring c. By the property (b) of Lemma 9, we know that every vertex v in $\{u \in V(G_2) : N_{G_2}(u) \cap Y \neq \emptyset\} = A' \cup \{b \in B : N_G(b) \cap B \neq \emptyset\}$, sees some vertex exactly once in its open neighborhood under the coloring c_2 (as well as in c). This proves property (ii) mentioned in the lemma. ◀

Algorithm 3 Algorithm for conflict free coloring.

Input: (i) vertex set $V(G)$ of the graph G on n vertices and the partition $V(G) = A \uplus B$, where $A = A' \uplus A''$, (ii) $N_G(a') \cap B$ for all $a' \in A'$, and (iii) $N_G(b)$ for all $b \in B$. The input satisfies the conditions of Lemma 10.

Output: A coloring function $c : V(G) \to [16\lceil e \ln^2 n \rceil + 8\lceil \ln n \rceil]$.

1 Consider the graph $G_1 = (A, E_1)$, where $E_1 = \{(u, v) : u, v \in A\}$. We apply the algorithm mentioned in Lemma 8 with G_1 as the input. Let $X = A''$, $Y = A'$, and $r_1 = n$. Notice that for all $x \in X = A''$, $\deg_{G_1}(x) \geq (4\lceil \ln n \rceil - 1)$. Let $c_1 : A \to [16\lceil e \ln^2 n \rceil]$ be the coloring returned by the algorithm mentioned in Lemma 8.

2 Consider the graph $G_2 = (A' \uplus B, E_2)$, where $E_2 = E(G) \setminus E_1$. Note that A' is an independent set in G_2. Invoke Algorithm 2 with G_2 as the input. Set $X = A'$, $Y = B$, and $r_2 = n$. Let the coloring function returned by Algorithm 2 be $c_2 : B \to \{16\lceil e \ln^2 n \rceil + 1, \ldots, 16\lceil e \ln^2 n \rceil + 8\lceil \ln n \rceil\}$.

3 Return $c : V(G) \to [16\lceil e \ln^2 n \rceil + 4\lceil \ln n \rceil - 1]$, defined as follows:

$$c(v) = \begin{cases} c_1(v), & \text{if } v \in A \\ c_2(v), & \text{if } v \in B \end{cases}$$

Proof of Theorem 5. The proof follows from the application of Lemma 10 and the following partitioning of the vertex set $V := V(G)$. We start by partitioning the vertex set V into two sets, namely A and B, where $A = \{v \in V : \deg_G(v) \geq (4\lceil \ln n \rceil - 1)\}$ and $B = V \setminus A$ using the following procedure . Here, A represents the set of high-degree vertices, while B represents the set of low-degree vertices.

To facilitate the partitioning process, we maintain a list $L(v) = \{u : u \in N_G(v)\}$. Initially, we set $A := \emptyset$ and $L(v) = \emptyset$. As each edge (u, v) passes through the stream, we update $L(u) = L(u) \cup \{v\}$ if $u \in V \setminus A$ and $L(v) = L(v) \cup \{u\}$ if $v \in V \setminus A$. Additionally, if a vertex w satisfies $|L(w)| \geq (4\lceil \ln n \rceil - 1)$, then we update $A := A \cup \{w\}$. At the end of the stream, we define $B := V \setminus A$. It is important to note that for any vertex $b \in B$, $|L(b)| < (4\lceil \ln n \rceil - 1)$. Up to this point, the storage requirement for $\bigcup_{v \in V} L(v)$ is $O(n \ln n)$ space.

Next, for all $a \in A$, update $L(a)$ as $L(a) = \{b \in B \ : \ a \in L(b)\}$. That is $L(a)$ is the set $N_G(a) \cap B$. Observe that updating $L(a)$'s can only double the total space used so far. After updating $L(a)$, for all $a \in A$, we define $A' = \{a \in A \ : \ L(a) \neq \emptyset\}$ and $A'' = A \setminus A'$. We have thus obtained (i) the sets A', A'', B, (ii) $N_G(a') \cap B$, for all $a' \in A'$, and (iii) $N_G(b)$ for all $b \in B$. We can now invoke Algorithm 3 with the required inputs to obtain a CFCN coloring function $c : V(G) \to [16\lceil e \ln^2 n \rceil + 8\lceil \ln n \rceil]$. From Lemma 10, we know that Algorithm 3 requires only $O(n \ln n)$ space and runs in polynomial time. ◀

Proof of Theorem 6. Given a graph $G = (V, E)$, we follow the same partitioning procedure mentioned in Theorem 5. That is, $A = \{v \in V : \deg_G(v) \geq (4\lceil \ln n \rceil - 1)\}$, and $B = V(G) \setminus A$. We again partition $B \subseteq V$ into two sets: $B = B_1 \uplus B_2$, where $B_1 = \{b \in B : N_G(b) \cap B = \emptyset\}$ and $B_2 = B \setminus B_1$ That is, using similar steps as in the case of Theorem 5, we get the partition of $V(G)$ into $A' \uplus A'' \uplus B_1 \uplus B_2$. Moreover, we also get $L(b) = N_G(b)$ for all $b \in B$ in $O(n \ln n)$ space. Now we invoke Algorithm 3 with the required inputs to obtain a coloring $c : V(G) \to [16\lceil e \ln^2 n \rceil + 8\lceil \ln n \rceil]$. From Lemma 10, we know that Algorithm 3 requires only

$O(n \ln n)$ space and runs in polynomial time. Moreover, with probability greater than $1 - \frac{2}{n}$, for every $v \in V(G) \setminus \{b \in B : N_G(b) \cap B = \emptyset\}$, v sees some color exactly once in its open neighbourhood.

We now invoke Proposition 3 (or an algorithmic version of it (see Theorem 18)) on the graph $G' = (A' \cup B_1, E')$ where $E' = \{(a, b) \in E(G) : a \in A', b \in B_1\}$ and get a CFON coloring $c' : V(G') \to [\lceil 2\sqrt{n} \rceil]$ of G'. Notice that $|E'| = O(n \ln n)$ and hence c' can be obtained in $O(n \ln n)$ space and polynomial time (see Theorem 18). Now, consider the coloring c_o on $V(G)$ defined as follows.

$$c_o(v) = \begin{cases} (c(v), c'(v)), & \text{if } v \in A' \cup B_1 \\ (c(v), 1), & \text{otherwise} \end{cases}$$

Now it is easy to see that c_o is a CFON coloring because for each $v \in V(G) \setminus B_1$, there is a unique color in the open neighborhood of v due to the coloring c and for each $v \in B_1$, there is a unique color in the open neighborhood of v due to the coloring c'. Notice that the number of colors used in c_o is $O(\sqrt{n} \ln^2 n)$. ◀

4 Semi-streaming algorithm for CFON coloring using $(1 + \epsilon)\Delta$ colors

In this section we prove a two-pass randomized semi-streaming algorithm for finding a CFON coloring of the input graph G using $(1 + \epsilon)\Delta$ colors, where Δ is the maximum degree of G. The theorem is formally stated below.

▶ **Theorem 11.** *There is a two-pass randomized semi-streaming algorithm (uses $O(\frac{1}{\epsilon^2} n \ln^3 n)$ space) that given a graph G and $\epsilon > 0$, outputs a CFON coloring of G using $(1 + \epsilon)\Delta$ colors or reports fail. It outputs a CFON coloring with probability at least $(1 - \frac{1}{n^3})$. Here, Δ is the maximum degree of G.*

We would like to mention that prior knowledge of Δ is not required. During the first pass of the edge stream, we find the maximum degree Δ of G, and for each vertex $u \in V(G)$, a special vertex $s(u)$ as explained below. Let $V(G) = \{v_1, \ldots, v_n\}$ and let Π be a fixed order $v_1, v_2, \ldots, v_n$. For each $u \in V(G)$, $s(u) = v_i$, where i is the least index such that $v_i \in N_G(u)$. We would like to mention that if our algorithm outputs a coloring, then for each vertex $u \in V(G)$, $s(u)$ will get a unique color among $N_G(u)$. It is easy to note that one can find the maximum degree Δ from the first pass using $O(n)$ space by keeping n counters, one per vertex. One can find out the special vertices from the first pass of the edge stream using the following procedure. We maintain an array (call it "a") of size n in memory. The array keeps track of the special vertices, i.e., the minimum indexed vertex in the open neighborhood of every vertex in the graph seen until now. Initially, all the elements in the array are set to NULL. When an edge (v_i, v_j) comes in the stream, we update $a[i]$ as follows.

- If $a[i]$ =NULL, then set $a[i] = v_j$.
- If $a[i] = w$ for some vertex $w \in V(G)$, then we set $a[i] = v_j$ iff $v_j <_\Pi w$.
- Similarly, we update the entry $a[j]$.

Notice that the space used in the first pass is $O(n \ln n)$. In the second pass of the stream, we construct a sketch H of G. Before the beginning of the second pass, for each vertex $v \in V(G)$, we sample a set of colors $L(v)$ of size $t = \frac{4}{\epsilon} \ln n$ from $[(1 + \epsilon)\Delta]$ uniformly and independently at random with replacement. Note that $L(v)$ is a multiset as the colors are sampled at random with replacement.

Algorithm 4 Algorithm for CFON coloring of graph G.

Data: Stream of edges of a graph G and $\epsilon > 0$
Result: coloring $c : V(G) \to [(1+\epsilon)\Delta]$ where Δ is the maximum degree of G

1 Fix an ordering $\Pi = v_1, \ldots, v_n$ of $V(G)$.
2 For all $v \in V$, sample a set $L(v)$ of size $t = \frac{4}{\epsilon} \ln n$ colors from $[(1+\epsilon)\Delta]$ uniformly and independently at random with replacement.
3 In the first pass of the edge stream, compute the maximum degree Δ of G and special vertices $s(u)$ for all $u \in V(G)$.
4 In the second pass of the edge stream, construct the sketch H of G with respect L (see Definition 12).
5 Do coloring $c : V(G) \to [(1+\epsilon)\Delta]$ using a greedy procedure on H as explained below.
6 In the i^{th} step, we have colored all the vertices $v_1, \ldots, v_{i-1}$. Now we color v_i as follows.
- Let $S_i = \{s(u) \ : \ u \in N_H(v_i)\} \cap \{v_1, \ldots, v_{i-1}\}$ and $Z_i = L(v_i) \setminus \{c(w) \mid w \in S_i\}\}$
- If $Z_i = \emptyset$, then report FAIL.
- Otherwise $c(v_i) = \min Z$.

▶ **Definition 12** (Sketch of G with respect to L)**.** *A sketch of G with respect to L is the subgraph H of G defined as follows: $V(H) = V(G)$, and*

$$E(H) = \{(u,v) : (L(u) \cap L(s(v)) \neq \emptyset) \vee ((L(s(u)) \cap L(v)) \neq \emptyset))\}$$

It is easy to store the sketch H of G in the second pass. For each edge (u,v) encountered in the edge stream, we check if $(u,v) \in E(H)$ by checking the condition $(L(u) \cap L(s(v)) \neq \emptyset) \vee ((L(s(u)) \cap L(v)) \neq \emptyset))$. Thus, by the end of the second pass, we have the sketch H of G with respect to L.

Finally, we do a coloring $c : V(G) \to [(1+\epsilon)\Delta]$ using a greedy procedure on H explained as follows. We color the vertices in the order $v_1, \ldots, v_n$. In the ith step we color the vertex v_i as follows. Let $S_i = \{s(u) \ : \ u \in N_G(v_i)\} \cap \{v_1, \ldots, v_{i-1}\}$.

Note that for each $u \in N_G(v_i), s(u) \leq_\Pi v_i$ and S_i is the set of special vertices, of "all" the neighbors of v_i, that are on the left side of v_i in the ordering Π. Let $Z_i = L(v_i) \setminus \{c(w) \mid w \in S_i\}$. That is, Z_i is the set of colors from $L(v_i)$ that are not used to color the vertices in S_i. Now we color v_i as per the following.

- If $Z_i = \emptyset$, then report FAIL. That is no color is available in the list $L(v_i)$ that is not equal to the color of the special vertices in S_i.
- Otherwise $c(v_i) = \min Z_i$.

For convenience, the pseudocode of the algorithm is available in Algorithm 4. Next we prove the following lemma which will be used to prove Theorem 11.

▶ **Lemma 13** (⋆)**.** *With probability at least* 0.99, $|E(H)| \leq \frac{1600}{\epsilon^2} n \ln^2 n$.

▶ **Lemma 14.** *If Algorithm 4 produces a coloring c then c is a CFON coloring of G. Moreover, Algorithm 4 produces a CFON coloring with probability at least* $(1 - \frac{1}{n^3})$.

Proof. Suppose Algorithm 4 outputs a coloring c. To prove that it is a CFON coloring of G, we need to prove that for each $v \in V(G)$, there is a vertex in $N_G(v)$ with unique color. We will prove that for each vertex $v \in V(G)$, $s(v)$ becomes the uniquely colored vertex in $N_G(v)$. Recall that $s(v)$ is the first vertex in Π among $N_G(v)$. So Algorithm 4 colors $s(v)$ before coloring the vertices in $N_G(v) \setminus \{s(v)\}$. Step 6 of Algorithm 4 implies that each vertex in $N_G(v) \setminus \{s(v)\}$ gets a color different from the color of $s(v)$. So, c is the CFON coloring of G.

We now find the probability that Algorithm 4 reports fail. Suppose Algorithm 4 reports FAIL in the ith iteration of step 6. It means that $Z_i = \emptyset$. That is, $L(v_i) \subseteq \{c(x) \mid x \in S_i\}$. We know that $|S_i| \leq |N_G(v_i)| \leq \Delta$. Here, we want to calculate $\Pr[L(v_i) \subseteq S_i]$. This is calculated as follows.

$$\Pr[L(v_i) \subseteq S_i] = \left(\frac{|S_i|}{((1+\epsilon)\Delta)}\right)^t \leq \left(\frac{\Delta}{(1+\epsilon)\Delta)}\right)^t = \frac{1}{(1+\epsilon)^t} = \frac{1}{(1+\epsilon)^{\frac{4}{\epsilon}\cdot \ln n}} \leq \frac{1}{n^4}$$

Here, the second inequality follows from the fact that $|S_i| \leq \Delta$. Notice that there are n iterations, and by the union bound, the probability that the algorithm reports FAIL in any one of these iterations is at most $\frac{1}{n^3}$. ◀

Proof of Theorem 11. We run Algorithm 4 $\ln n$ times parallelly. In each execution of the algorithm, if the space used by it exceeds $\frac{1600}{\epsilon^2} n \ln n$, then we abort it. Finally, if any of the execution outputs a coloring, we output one such coloring. Otherwise, we output FAIL. The probability that each execution is either aborted or output FAIL is upper bounded by $\frac{1}{100} + \frac{1}{n^3} \leq 0.02$. The probability that all the executions fail with probability is at most $0.02^{\ln n} \leq \frac{1}{n^3}$. This implies that the overall probability that Algorithm 4 succeeds is at least $(1 - \frac{1}{n^3})$. Notice that each execution of Algorithm 4 takes $O(\frac{1}{\epsilon^2} n \ln^2 n)$ space and hence the total space is $O(\frac{1}{\epsilon^2} n \ln^3 n)$. ◀

5 Concluding remarks

Most of the study on CFCN and CFON coloring so far has been towards proving tighter bounds in terms of various graph parameters like maximum degree, order of the graph, pathwidth, etc. Showing improved bounds for special graph classes like planar graphs, line graphs, unit-disc graphs, etc. has also been explored. CFCN and CFON coloring problems have also been explored from a parameterized complexity setting [10, 11, 13]. To the best of our knowledge, this is the first paper exploring semi-streaming algorithms for these coloring problems. The bounds (on the number of colors used) we obtain in Theorems 5, 11, and 18 are asymptotically tight. Yet these leave us with many more intriguing open questions, some of which are listed below. Given as input an edge stream of an n-vertex graph G with maximum degree Δ,

1. Is it possible to obtain a deterministic single pass semi-streaming algorithm that does a CFCN coloring using $O(\ln^2 n)$ colors? The algorithm given by Theorem 5 is randomized.
2. Is it possible to improve Theorem 6 to obtain a semi-streaming (deterministic or randomized) algorithm that gives a CFON coloring using $O(\sqrt{n})$ colors?
3. Is it possible to obtain a semi-streaming algorithm that outputs a CFON coloring of G using $\Delta + 1$ colors? Theorem 11 gives a two-pass randomized semi-streaming algorithm that outputs a CFON coloring using at most $(1+\epsilon)\Delta$ colors. And, Theorem 18 can be modified to obtain a deterministic single-pass streaming algorithm for CFON coloring of G using $\Delta + 1$ colors that uses $O(n\sqrt{n})$ space (the algorithm given by the theorem works, if $\Delta \geq 2\sqrt{n} - 1$. Otherwise, we store the entire graph and color it offline). As far as classical coloring is concerned, there are randomized single-pass semi-streaming algorithms that output a $\Delta + 1$-coloring with high probability (see [6]). At the same time, it is known (see [8]) that any single-pass streaming algorithm that outputs a $(d+1)$-coloring of G requires at least $\Omega(n^2)$ space, where d is the degeneracy of G (the graph G is k-degenerate if its vertices can be arranged in a line such that every vertex has at most k neighbors to its left. The minimum k for which G is k-degenerate is called its degeneracy. Clearly, this parameter is at most Δ). Coming back to the CFON coloring problem, any CFON

coloring of G would want some vertex, say $s(v)$, in the neighborhood of each vertex v to receive a color distinct from that of the other neighbors of v. If we construct an auxiliary graph of G by adding edges between $s(v)$ and all the other neighbors of v, for every v, such a graph would have a maximum degree of $\Theta(\Delta^2)$ and degeneracy of Δ. And a classical coloring of this auxiliary graph would yield a CFON coloring of G. This makes the question of obtaining a semi-streaming algorithm for $(\Delta+1)$-CFON coloring of G all the more interesting.

References

1. Noga Alon and Sepehr Assadi. Palette sparsification beyond $(\Delta + 1)$ vertex coloring. *International Workshop and International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques 2020*, 2020. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.6.
2. Mohammad Ansari, Mohammad Saneian, and Hamid Zarrabi-Zadeh. Simple streaming algorithms for edge coloring. In *30th Annual European Symposium on Algorithms (ESA 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.8.
3. Sepehr Assadi, Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl. Coloring in graph streams via deterministic and adversarially robust algorithms. *PODS '23: Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 2022.
4. Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\Delta+1)$ vertex coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 767–786. SIAM, 2019. doi:10.1137/1.9781611975482.48.
5. Sepehr Assadi, Pankaj Kumar, and Parth Mittal. Brooks' theorem in graph streams: a single-pass semi-streaming algorithm for Δ-coloring. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 234–247, 2022.
6. Sepehr Assadi and Helia Yazdanyar. Simple sublinear algorithms for $(\Delta+1)$ vertex coloring via asymmetric palette sparsification. In *2025 Symposium on Simplicity in Algorithms (SOSA)*, pages 1–8. SIAM, 2025. doi:10.1137/1.9781611978315.1.
7. Soheil Behnezhad and Mohammad Saneian. Streaming edge coloring with asymptotically optimal colors. *arXiv preprint arXiv:2305.01714*, 2023. doi:10.48550/arXiv.2305.01714.
8. 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)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:21, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.11.
9. Anup Bhattacharya, Arijit Bishnu, Gopinath Mishra, and Anannya Upasana. Even the easiest(?) graph coloring problem is not easy in streaming! In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 15:1–15:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ITCS.2021.15.
10. Sriram Bhyravarapu, Tim A. Hartmann, Subrahmanyam Kalyanasundaram, and I. Vinod Reddy. Conflict-free coloring: Graphs of bounded clique width and intersection graphs. In Paola Flocchini and Lucia Moura, editors, *Combinatorial Algorithms*, pages 92–106, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-79987-8_7.
11. Sriram Bhyravarapu and Subrahmanyam Kalyanasundaram. A tight bound for conflict-free coloring in terms of distance to cluster. *Discrete Mathematics*, 345(11):113058, 2022. doi:10.1016/j.disc.2022.113058.
12. Sriram Bhyravarapu, Subrahmanyam Kalyanasundaram, and Rogers Mathew. A short note on conflict-free coloring on closed neighborhoods of bounded degree graphs. *Journal of Graph Theory*, 97(4):553–556, 2021. doi:10.1002/JGT.22670.

13 Hans L. Bodlaender, Sudeshna Kolay, and Astrid Pieterse. Parameterized complexity of conflict-free graph coloring. *SIAM J. Discret. Math.*, 35(3):2003–2038, 2021. doi:10.1137/19M1307160.

14 Panagiotis Cheilaris. *Conflict-free coloring*. City University of New York, 2009.

15 Michał Dębski and Jakub Przybyło. Conflict-free chromatic number versus conflict-free chromatic index. *Journal of Graph Theory*, 99(3):349–358, 2022. doi:10.1002/JGT.22743.

16 Khaled M. Elbassioni and Nabil H. Mustafa. Conflict-free colorings of rectangles ranges. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*, volume 3884 of *Lecture Notes in Computer Science*, pages 254–263. Springer, 2006. doi:10.1007/11672142_20.

17 Guy Even, Zvi Lotker, Dana Ron, and Shakhar Smorodinsky. Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. *SIAM Journal on Computing*, 33(1):94–136, 2003. doi:10.1137/S0097539702431840.

18 Roman Glebov, Tibor Szabó, and Gábor Tardos. Conflict-free colouring of graphs. *Combinatorics, Probability and Computing*, 23(3):434–448, 2014. doi:10.1017/S0963548313000540.

19 Prasad Krishnan, Rogers Mathew, and Subrahmanyam Kalyanasundaram. Pliable index coding via conflict-free colorings of hypergraphs. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 214–219. IEEE, 2021. doi:10.1109/ISIT45174.2021.9518120.

20 E Kushilevitz and N Nisan. Communication complexity, cambridge univ. *Press, Cambridge, UK*, 1997.

21 Nissan Lev-Tov and David Peleg. Conflict-free coloring of unit disks. *Discrete Applied Mathematics*, 157(7):1521–1532, 2009. doi:10.1016/j.dam.2008.09.005.

22 Andrew McGregor. Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20, 2014. doi:10.1145/2627692.2627694.

23 János Pach and Gábor Tardos. Conflict-free colourings of graphs and hypergraphs. *Combinatorics, Probability and Computing*, 18(5):819–834, 2009. doi:10.1017/S0963548309990290.

24 Shakhar Smorodinsky. *Combinatorial problems in computational geometry*. PhD thesis, Tel-Aviv University, 2003.

25 Shakhar Smorodinsky. Conflict-free coloring and its applications. *Geometry – Intuitive, Discrete, and Convex: A Tribute to László Fejes Tóth*, pages 331–389, 2013.

A Proof of Lemma 7

In any given iteration of the "while" loop, the "for" loop in Step 5 runs at most n times. The while loop in Step 3 runs at most $T = \lceil 16e \ln^2 r \rceil$ times. Thus it is easy to see that the algorithm terminates in $O(nT) = O(n \ln^2 r)$ steps.

Note that the vertex set V of $\mathcal{H}$ can be stored in $O(n)$ space. The algorithm does not need to store the edges regardless of the structure of the hypergraph $\mathcal{H}$. The algorithm does a randomized coloring of the vertices in V, which is explained in steps 5-8. This can be done in $O(n)$ workspace. Thus, the total space required is $O(n)$.

Consider an execution of Algorithm 1. Let A_v denote the event of v not receiving any non-zero color.

$$\Pr[A_v] = (1-p)^T, \text{ where } p \text{ is the probability for Head and } p = \frac{1}{8e \ln r}$$
$$= \left(1 - \frac{1}{8e \ln r}\right)^T \leq e^{-\frac{T}{8e \ln r}} \leq \frac{1}{r^2} \qquad (\because T = \lceil 16e \ln^2 r \rceil)$$

Let A denote the event of some vertex in V not receiving a non-zero color. Then, $\Pr(A) = \Pr(\bigcup_{v \in V} A_v) \leq \sum_{v \in V} \Pr(A_v) \leq n \cdot \frac{1}{r^2} \leq r \cdot \frac{1}{r^2} (\because n \leq r) = \frac{1}{r}$. Thus, we have $\Pr(A) \leq \frac{1}{r}$.

Let $\bar{A}$ denote the complement of event A. For a hyperedge $E \in \mathcal{E}$, let B_E denote the event of E not seeing a uniquely colored vertex. And, let B denote the event of some hyperedge $E \in \mathcal{E}$ not seeing a unique color. That is, B denotes the event of the coloring function not being a conflict-free coloring of $\mathcal{H}$. From the definition of B, $B = \bigcup_{E \in \mathcal{E}} B_E$. Let $\bar{B}$ denote the complement of the event B. Then we know that,

$$\Pr(\bar{A} \cap \bar{B}) = \Pr(\bar{A}) \cdot \Pr(\bar{B}|\bar{A}) \geq (1 - \frac{1}{r}) \cdot \Pr(\bar{B}|\bar{A}) \tag{1}$$

We are left with the task of estimating $P(\bar{B}|\bar{A})$. We have,

$$\Pr(\bar{B}|\bar{A}) = 1 - \Pr(B|\bar{A}) = 1 - \Pr(\bigcup_{E \in \mathcal{E}} B_E|\bar{A}) \geq 1 - \sum_{E \in \mathcal{E}} \Pr(B_E|\bar{A}) \tag{2}$$

To estimate $\Pr(B_E|\bar{A})$, we use the following lemma from [23].

▶ **Lemma 15** (Lemma 3.1 in [23]). *Let V be a set of elements and let $E \subseteq V$ with $|E| \geq 2t-1$ for a positive integer t. Let us color each element of V independently, according to the geometric distribution with parameter p. Then the probability that no element of E receives a unique color (one that is not received by any other vertex of E) is at most $2(etp)^t$.*

With respect to the execution of Algorithm 1, given that the event $\bar{A}$ has happened (that is, the algorithm has assigned a non-zero color for every vertex $v \in V$), the coloring c returned by the algorithm is a coloring based on a geometric distribution with parameter $p = \frac{1}{8e \ln r}$. Thus, Lemma 15 gives an upper bound to $P(B_E|\bar{A})$. Applying Lemma 15 with $p = \frac{1}{8e \ln r}$ and $t = 2 \ln r$, we have

$$\Pr(B_E|\bar{A}) \leq 2 \left(e \cdot 2 \ln r \cdot \frac{1}{8e \ln r} \right)^{2 \ln r} \leq \frac{2}{r^{2.773}} \tag{3}$$

From Equations (2) and (3), we get

$$\Pr(\bar{B}|\bar{A}) \geq 1 - |\boldsymbol{\mathcal{E}}| \cdot \frac{2}{r^{2.773}} = 1 - \frac{4}{r^{1.773}} \qquad \text{(Because } |\mathcal{E}| \leq 2n \leq 2r\text{)} \tag{4}$$

From Equations (4) and (1), we have

$$\Pr(\bar{A} \cap \bar{B}) \geq \left(1 - \frac{1}{r}\right) \left(1 - \frac{4}{r^{1.773}}\right) \geq \left(1 - \frac{1}{r}\right)^2 > \left(1 - \frac{2}{r}\right) \tag{5}$$

Here, the second inequality follows from the fact that $1 - \frac{4}{r^{1.773}} > 1 - \frac{1}{r}$ when $r \geq 7$. This completes the proof of the lemma.

B Proof of Lemma 13

To prove Lemma 13, first we prove the following lemma and then use this lemma.

▶ **Lemma 16.** *Let $(u, v) \in E(G)$. Then $\Pr[(u, v) \in E(H)] \leq \frac{2t^2}{(1+\epsilon)\Delta}$.*

Proof. Recall that for all $w \in V(G)$, $L(w)$ denotes the pallete of colors for the vertex w and $|L(w)| = t$. The colors are sampled uniformly and independently at random with replacement from the set of colors $[(1+\epsilon)\Delta]$. Let E_1 be the event $(L(u) \cap L(s(v))) \neq \emptyset$ and let E_2 be the event $(L(s(u)) \cap L(v)) \neq \emptyset$. Now, we have $\Pr[(u, v) \in E(H)] = \Pr[E_1 \vee E_2] \leq \Pr[E_1] + \Pr[E_2]$. We now calculate $\Pr[E_1]$. Let $L(s(v)) = \{r_1, \ldots, r_t\}$. Recall that $L(s(v))$ is a multiset.

$$\Pr[E_1] = \Pr\left[\bigcup_{i\in[t]} r_i \in L(u)\right] \leq \sum_{i\in[t]} \Pr(r_i \in L(u)) = \sum_{i\in[t]} \frac{t}{(1+\epsilon)\Delta} = \frac{t^2}{(1+\epsilon)\Delta}$$

Similarly, $\Pr[E_2] \leq \frac{t^2}{(1+\epsilon)\Delta}$. Thus, we get $\Pr[(u,v) \in E(H)] \leq \Pr[E_1] + \Pr[E_2] \leq \frac{2t^2}{(1+\epsilon)\Delta}$. ◀

Below we state Markov's Inequality.

▶ **Proposition 17** (Markov's inequality). *If X is a nonnegative random variable and $a > 0$, then $\Pr[X \geq a] \leq \frac{E(X)}{a}$.*

Now, we are ready to prove Lemma 13.

Proof of Lemma 13. Let $\mathbb{X}$ be the random variable that denotes $|E(H)|$. For each $v \in V(H)$, let $\mathbb{X}_v$ denote the degree of vertex v in H. For each $(v,u) \in E(G)$, $\mathbb{X}_{vu}$ is the indicator random variable defined as follows: $\mathbb{X}_{vu} = 1$ if and only if $(v,u) \in E(H)$. Then, for any $v \in V(G)$, $\mathbb{X}_v = \sum_{u\in N_G(v)} \mathbb{X}_{vu}$, and $E[\mathbb{X}_v]$ is calculated as follows.

$$E[\mathbb{X}_v] = \sum_{u\in N_G(v)} E[\mathbb{X}_{vu}] \leq \Delta \frac{2t^2}{(1+\epsilon)\Delta} = \frac{2t^2}{(1+\epsilon)} \tag{6}$$

Here, the second inequality follows from Lemma 16. Notice that $\mathbb{X} = \frac{1}{2}\sum_{v\in V(G)} \mathbb{X}_v$. Thus, by Equation (6)

$$E[\mathbb{X}] = \frac{1}{2}\sum_{v\in V(G)} E[\mathbb{X}_v] \leq \frac{n}{2} \cdot \frac{2t^2}{(1+\epsilon)} \leq \frac{16n\ln^2 n}{\epsilon^2}.$$

By Markov's inequality, $\Pr[\mathbb{X} > \frac{1600n\ln^2 n}{\epsilon^2}] \leq \frac{1}{100}$. Thus, $|E(H)| \leq \frac{1600}{\epsilon^2} n\ln^2 n$ with probability at least 0.99. ◀

C Streaming algorithm for CFON coloring using $O(\sqrt{n})$ colors

Cheilaris proved that, for a graph G on n vertices, $\chi_{ON}(G) \leq 2\sqrt{n}$ [14]. Pach and Tardos improved this to $\chi_{ON}(G) \leq \sqrt{2n}$ [23]. The above upper bounds are asymptotically tight due to Example 4. Cheilaris's result in Proposition 3 uses a deterministic algorithm to do a coloring of the vertices as follows. When a vertex is present in at least $\sqrt{n}$ neighborhoods, then we color that vertex with a new distinct color. Note that this vertex will be a unique color for at least $\sqrt{n}$ neighbourhoods. Next, we consider another vertex that is present in at least $\sqrt{n}$ neighborhoods with no vertex in this neighborhood colored so far. Then, we color that vertex with a new distinct color and continue this process. At the end of this process, we have used at most $\sqrt{n}$ colors because there are only n neighborhoods. Also, at the end of this process, each vertex will be part of at most $\sqrt{n}$ neighborhoods with no vertex in these neighborhoods colored so far. Then we use Proposition 1 to color the remaining vertices using at most $\sqrt{n}$ new colors. Clearly, this algorithm requires to see the entire input to do the coloring.

In this section, we give a streaming algorithm to do a CFON coloring of G using $2\sqrt{n}$ colors. We prove the following result.

▶ **Theorem 18.** *There is a deterministic single-pass $O(n\sqrt{n})$-space streaming algorithm that, given a graph G on n vertices, finds a CFON coloring using at most $2\sqrt{n}$ colors.*

Algorithm 5 Algorithm for CFON coloring of a graph G on n vertices using $2\sqrt{n}$ colors.

Input: Stream of edges of G, where $V(G) = [n]$.
Output: coloring function $f : V(G) \to [\lceil 2\sqrt{n} \rceil]$.

Set $\mathcal{V} := V(G)$;
Set $L(v) := \emptyset$, for all $v \in V(G)$;
Set $i := 1$;
Set marked$[v]$:= false, for all $v \in V(G)$;
`/* marked[v] denotes if` $N_G(v)$ `has a unique colored member or not */`
while *an edge* $e = (u, v)$ *arrives in the stream* **do**
  if v *is colored* **then**
    marked$[u]$:= true;
    $L(u) := \emptyset$;
    `/*` v `is a uniquely colored vertex in` $N_G(u)$. `*/`
  if u *is colored* **then**
    marked$[v]$:= true ;
    $L(v) := \emptyset$;
    `/*` u `is a uniquely colored vertex in` $N_G(v)$. `*/`
  `/* Below, we update the lists` $L(u)$ `and` $L(v)$ `based on whether` u `and` v `are unmarked or not. */`
  if *marked*$[v]$ = *false* **then**
    $L(v) := L(v) \cup \{u\}$;
  if *marked*$[u]$ = *false* **then**
    $L(u) := L(u) \cup \{v\}$;
  if v *is not colored and* $|\{w \ : \ v \in L(w)\}| \geq \sqrt{n}$ **then**
    `/* Below, we color` v `*/`
    $f(v) := i$;
    $i := i + 1$;
    $\mathcal{V} := \mathcal{V} \setminus \{v\}$;
    $L(z) := \emptyset$ and *marked*$[z]$:= true, for all z such that $v \in L(z)$;
    `/*` v `is a uniquely colored vertex in` $N_G(z)$. `*/`
  if u *is not colored and* $|\{w \ : \ u \in L(w)\}| \geq \sqrt{n}$ **then**
    `/* Below, we color` u `*/`
    $f(u) := i$;
    $i := i + 1$;
    $\mathcal{V} := \mathcal{V} \setminus \{u\}$;
    $L(z) := \emptyset$ and *marked*$[z]$:= true, for all z such that $v \in L(z)$;
    `/*` u `is a uniquely colored vertex in` $N_G(z)$. `*/`
Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be the hypergraph, where $\mathcal{E} = \{L(v) : v \in V(G),\ marked[v] =$ false$\}$. Note that the maximum degree of any vertex in $\mathcal{H}$ is at most $\sqrt{n} - 1$. Apply the greedy coloring algorithm given by Proposition 1 to do a CF coloring of $\mathcal{H}$ using $\sqrt{n}$ colors from the set $\{i + 1, \ldots, i + \sqrt{n}\}$.
Output f.

Proof. The pseudocode of our algorithm is given in Algorithm 5. We explain the algorithm here. We use $\mathcal{V}$ to denote the set of all so-far uncolored vertices. Initially, as all the vertices in $V(G)$ are uncolored, we initialize $\mathcal{V} := V(G)$. We "mark" a vertex v if it sees some unique

coloring in $N_G(v)$. Initially, all the vertices are unmarked. For an unmarked vertex v, we shall use the list $L(v)$ to denote the set of neighbors of v that we have seen so far. From the moment v is marked, we maintain $L(v)$ to be an empty set as the "marked" status of v implies it is seeing a uniquely colored vertex in its open neighborhood. When a new edge (u, v) arrives in the stream, the algorithm executes Steps 6 to 27 which are explained below. Note that in these steps while coloring vertices we do not use the same color twice. If u (or v) is already colored then we mark v, (resp., u) as u (resp., v) will act as the uniquely colored vertex in the open neighborhood of v (resp., u). As mentioned above, as soon as a vertex is marked, its list (in this case, $L(u)$ or $L(v)$ or both) is reset to the empty set. These steps are executed in Lines 6 to 11. We update the list of u (or v), if it is unmarked, in Lines 12 to 15. If v is not colored and is present in at least $\sqrt{n}$ lists, then we assign a new color to v. We remove it from $\mathcal{V}$ which maintains the set of all uncolored vertices so far. We mark all vertices z that have v in $L(z)$, as v will act as the uniquely colored neighbor of such a vertex z. As mentioned before, we immediately reset $L(z)$ to the empty set. These steps are implemented in Lines 16 to 21. Now, in a similar fashion, if u is not colored and is present in at least $\sqrt{n}$ lists, we execute similar steps for the vertex u in Lines 22 to 27.

We claim that we color at most $\sqrt{n}$ vertices in total during the various iterations of the **while** loop (Lines 6 to 27). Every time we color a new vertex in Line 18 or Line 24, we update the status of least $\sqrt{n}$ new vertices from unmarked to marked and set their lists to $\emptyset$. Since there are only n vertices in total and since we do not unmark a vertex that is already marked, our claim holds. Next, we claim that every vertex that is marked sees some color exactly once in its open neighborhood. Every time a vertex is marked in Lines 10, 21, or 27, it has some colored vertex in its open neighborhood (see the comments below these lines). Since no two vertices receive the same color inside the **while** loop, our claim holds. To summarise, (i) we have used at most $\sqrt{n}$ colors so far, and (ii) every marked vertex is seeing some color exactly once in its open neighborhood.

When we exit the **while** loop, it is only the unmarked vertices that are yet to see a unique color in their open neighborhood. We take care of them in Line 28. In this step, we construct a hypergraph $\mathcal{H}$ whose vertex set is the set of so far uncolored vertices (maintained by $\mathcal{V}$) and whose edge set $\mathcal{E}$ is all the lists $L(v)$ for which v is unmarked. In other words, $\mathcal{E}$ is the set of all lists $L(v)$ that are not empty sets. Clearly, such lists $L(v)$ contain only uncolored vertices (that is, vertices in $\mathcal{V}$); else v would have been marked and $L(v)$ would have been reset to $\emptyset$. Further, for any unmarked vertex v, $L(v) = N_G(v)$ as our algorithm added every neighbor of v into $L(v)$ as and when they were unveiled (see Lines 13 and 15). Thus, any conflict-free coloring of $\mathcal{H}$ using a set of new colors would result in every unmarked vertex seeing a unique color in its open neighborhood in G. We claim that the maximum degree of any vertex in $\mathcal{H}$ (that is, the maximum number of hyperedges any vertex in $\mathcal{V}$ is present in) is at most $\sqrt{n} - 1$ and then use Proposition 1 to do a greedy conflict-free coloring of $\mathcal{H}$ using at most $\sqrt{n}$ new colors. We prove the claim by contradiction. Suppose there existed a vertex $v \in \mathcal{V}$ which was present in $\sqrt{n}$ or more hyperedges (or lists) $L(w)$ in $\mathcal{H}$. But, then such a vertex v would satisfy the conditions of the **if** statement in Line 16. Line 20 would then lead to the contradictory situation that v is not a member of $\mathcal{V}$. This proves the claim.

We have thus shown that the algorithm does give a CFON coloring of G using at most $2\sqrt{n}$ colors. It is easy to see that it is a single-pass, deterministic algorithm. Finally, the amount of memory required is the space required to store the lists $L(u)$ for all u. Notice that each time during the execution of the algorithm each vertex v is present in at most $\sqrt{n}$ lists. This implies that the space complexity is $O(n\sqrt{n})$. ◀

Fast Kd-Trees for the Kullback–Leibler Divergence and Other Decomposable Bregman Divergences

Tuyen Pham
University of Florida, Gainesville, FL, USA

Hubert Wagner
University of Florida, Gainesville, Fl, USA

Abstract

The contributions of the paper span theoretical and implementational results. First, we prove that Kd-trees can be extended to $\mathbb{R}^d$ with the distance measured by an arbitrary Bregman divergence. Perhaps surprisingly, this shows that the triangle inequality is not necessary for correct pruning in Kd-trees. Second, we offer an efficient algorithm and C++ implementation for nearest neighbour search for decomposable Bregman divergences.

The implementation supports the Kullback–Leibler divergence (relative entropy) which is a popular distance between probability vectors and is commonly used in statistics and machine learning. This is a step toward broadening the usage of computational geometry algorithms.

Our benchmarks show that our implementation efficiently handles both exact and approximate nearest neighbour queries. Compared to a linear search, we achieve two orders of magnitude speedup for practical scenarios in dimension up to 100. Our solution is simpler and more efficient than competing methods.

2012 ACM Subject Classification Theory of computation → Computational geometry; Information systems → Data structures; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Kd-tree, k-d tree, nearest neighbour search, Bregman divergence, decomposable Bregman divergence, KL divergence, relative entropy, cross entropy, Shannon's entropy

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.45

Funding 2022 Google Research Scholar Award in Algorithms and Optimization.

1 Motivation

Nearest neighbour search is a fundamental method offered by computational geometry, with applications in a wide range of fields. Bentley's k-dimensional tree [11], Kd-tree for short, is among the simplest and most practical data structures for this task.

Like many other computational geometry techniques, Kd-trees were initially designed for Euclidean space and later extended to more general metric spaces. However, many modern geometric problems, particularly in data science, use distances that are not proper metrics. For example, it is common to represent data as probability vectors and use specialized dissimilarities to measure distances between them. While standard geometric algorithms do not work with non-metric distances, they can often be extended to such settings.

Indeed, it is interesting that many computational geometry algorithms – that are typically assumed to require a metric – can work with significantly weaker assumptions. Specifically, we may omit the requirement of symmetry or the triangle inequality. We will mention prominent examples in Section 2, and prove that the above statement extends to Kd-trees. In particular, correctness and efficiency guarantees can be retained.

Applications. One practical example of such a non-metric distance is the Kullback–Leibler (KL) divergence [28]. Originating in information theory [47], the KL divergence is a standard way of comparing discrete probability distributions (probability vectors). For example, it

© Tuyen Pham and Hubert Wagner;
licensed under Creative Commons License CC-BY 4.0

is used as a loss function minimized in machine learning [34, 37, 50], such as in variational auto-encoders [26]. Approximate nearest neighbours queries are becoming an increasingly important component of modern machine learning. In particular, retrieval-augmented generation (RAG) aims to improve large language models by searching for existing documents to support generated answers. This is done by a nearest neighbour search within probability vectors. Typically, heuristic methods, such as the small world graphs [30], which lack performance guarantees, are used [33, 35, 51]. Recently, Indyk and Xu [24] warned that the methods used in such contexts can catastrophically fail. Regardless, there is a growing field of *vector databases* focusing on supporting nearest neighbour queries [23].

Problem statement. We revisit the topic of nearest neighbour search, focusing on algorithms that provide exact answers as well as approximate results with guarantees. In particular, we investigate non-metric geometries induced by Bregman divergences – of which the KL divergence is a prominent member.

Given a finite collection of points $X \subset \Omega \subset \mathbb{R}^d$ and a query point $q \in \Omega$, select k points from X with the smallest distance to q. Specifically, the distance will be measured using a Bregman divergence, which we discuss in Section 3. We will design and implement a modified Kd-tree for answering queries in this setting.

Contributions. We list the contributions of this paper:

1. Theoretical results on correctness and efficiency of Kd-tree queries in the setting of Bregman divergences.
2. The first implementation of Kd-trees for Bregman divergences. It is currently the fastest method for exact k-nearest neighbour queries in the Bregman setting and works for arbitrary decomposable divergences[1].
3. Benchmarks showing the method is usable in practical situations with real datasets.

2 Related work

Kd-Trees were introduced by Jon Bentley in 1975 [11]. Further improvements were made by him, Friedman and Finkel [22] and many others. Bregman divergences were introduced by Lev Bregman in 1967 [16].

Many computational geometry techniques have been extended to operate with Bregman divergences instead of a metric. In the context of nearest neighbour search, Cayton first extended ball-trees [19], implementing prototype software for the KL and Itakura–Saito divergences. Cayton also proved theoretical results towards extending Kd-trees [17], which we strengthen as well as provide algorithms and an efficient implementation. In turn, Nielsen, Piro, and Barlaud extended Vantage point trees [40, 41]. The same authors further improved Bregman ball-trees [18, 42]. These methods rely on a bisection search and in some cases a preprocessing transformation. We show that Kd-trees are able to perform exact nearest neighbour searches without relying on these computations.

Rectangle-trees (R-trees) and vector approximation files (VA-files) were extended by Zhang and collaborators [53]. These implementations inspired Song and collaborators to develop BrePartitions [48]. Ring-trees combined with a quad-tree decomposition have been proven to work sublinearly for finding approximate nearest neighbours by Abdullah, Moeller,

[1] We are working on incorporating our implementation into the popular scikit-learn [44] library, which will increase both the generality and efficiency of its existing Kd-trees implementation.

and Venkatasubramanian [2]. In 2013, Boytsov and Naidan developed their own Bregman VP-trees extension [15] for approximate nearest neighbours. Naidan later incorporated his VP-trees and Cayton's ball-trees into the Non-Metric Space Library (NMSLIB) [14]. This library also includes other approximate Bregman similarity searches including small world graphs [32]. The hierarchical navigable small world graph has been a popular choice for similarity searches in vector databases [33, 35, 51] and performs well in benchmarks for metrics [8]. However, its implementation in NMSLIB is currently experimental for Bregman divergences [31]. Recently Abdelkader, Arya, da Fonseca and Mount proposed an approach to proximity search in non-metric settings, which includes Bregman divergences [1]; as we understand it, this has not yet been implemented.

More broadly, Banerjee and collaborators extended k-means clustering [9] to arbitrary Bregman divergences – with the surprising twist that the existing algorithm works without changes. Coresets have also been extended to the Bregman setting by Ackermann and Blömer [3]. Nielsen, Boissonnat, and Nock developed Bregman Voronoi diagrams and Delaunay triangulations [13]. Edelsbrunner and Wagner extended topological data analysis methods to the Bregman case [21].

In the Euclidean case, robust software is available for all of these techniques. One popular package in the Euclidean case is the ANN library by Mount and Arya [5–7, 36]. Our current implementation is inspired by this library.

3 Background on Bregman divergences

We begin by setting up definitions for Bregman divergences [16], which we use as a measure of distance. These divergences are usually asymmetric and do not generally satisfy the triangle inequality – and as such do not define a proper metric. Despite this limitation, decomposable Bregman divergences will efficiently work with Kd-trees with minimal changes.

Each Bregman divergence is parametrized by a convex function with particular properties [10]. We set the stage by letting $\Omega \subseteq \mathbb{R}^d$ be an open convex set. Next, we define a **function of Legendre type** [46] as a function $F : \Omega \to \mathbb{R}$ that is:

- **I** differentiable and
- **II** strictly convex.
- **III** We additionally require that $\lim_{x \to \partial\Omega} \|\nabla F(x)\| = \infty$, provided $\partial\Omega$ is nonempty.

The third requirement is often omitted, but will prove important in Section 5.

Given a function F of Legendre type, the **Bregman divergence generated** by F is a function $D_F : \Omega \times \Omega \to \mathbb{R}$. The value of the divergence between x and y is the difference between $F(x)$ and the best affine approximation of F at y also evaluated at x, or simply

$$D_F(x\|y) = F(x) - (F(y) + \langle \nabla F(y), x - y \rangle). \tag{1}$$

See Figure 1 for an illustration. We refer to $D_F(x\|y)$ as the divergence in the **direction from x to y**. Due to the lack of symmetry, we will be mindful about the direction in which we compute it.

All Bregman divergences fulfill the following:

▶ **Property 1** (Bregman Nonnegativity). *$D_F(x\|y) \geq 0$ for each $x, y \in \Omega$, with equality if and only if $x = y$.*

Despite failing to satisfy the requirements for a metric, various computational geometry algorithms extend to Bregman divergences. Also, despite the seemingly simple definition, the resulting divergences have interesting properties and interpretations.

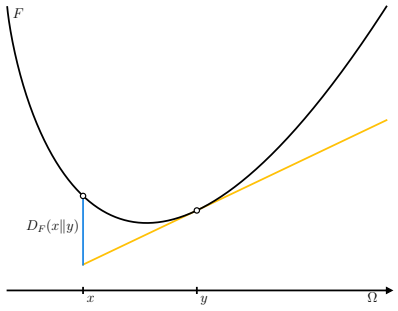


Figure 1 Visualization of a Bregman divergence construction for a one-dimensional domain.

Table 1 List of common decomposable Bregman divergences.

Domain	Legendre type function	Divergence	Name
$\mathbb{R}^d$	$\sum_{i=1}^d x_i^2$	$\sum_{i=1}^d (x_i - y_i)^2$	Squared Euclidean (SE)
$\mathbb{R}^d_+$	$-\sum_{i=1}^d x_i \log_2 \frac{1}{x_i}$	$\sum_{i=1}^d x_i \log_2 \frac{x_i}{y_i} + \frac{y_i - x_i}{\ln 2}$	Generalized Kullback–Leibler (GKL)
$\triangle^{d-1}$	$-\sum_{i=1}^d x_i \log_2 \frac{1}{x_i}$	$\sum_{i=1}^d x_i \log_2 \frac{x_i}{y_i}$	Kullback–Leibler (KL)
$\mathbb{R}^d_+$	$-\sum_{i=1}^d \log x_i$	$\sum_{i=1}^d \frac{x_i}{y_i} - \log \frac{x_i}{y_i} - 1$	Itakura–Saito (IS)
$\mathbb{R}^d_+$	$-\sum_{i=1}^d -\sqrt{x_i}$	$\sum_{i=1}^d \frac{\sqrt{y_i}}{2} + \frac{x_i}{2\sqrt{y_i}} - \sqrt{x_i}$	Bhattacharyya-Like
$\Omega_1 \cap \Omega_2$	$\lambda F_1 + (1-\lambda)F_2$, $\lambda \in [0,1]$	$\lambda D_{F_1} + (1-\lambda) D_{F_2}$	Interpolated divergence

Decomposable Bregman divergences. Our focus is on a sub-family of Bregman divergences called **decomposable Bregman divergence** [38, 52]. They are generated by a function $F = \sum_{i=1}^d f_i$, where each f_i is a univariate function of Legendre type. The function, F, generates a Bregman divergence of the form $D_F(x\|y) = \sum_{i=1}^d D_{f_i}(x_i\|y_i)$ for x_i, y_i (namely the components of x and y) lying in the domain of f_i [39]. Most divergences used in practice belong to this family.

We list common decomposable Bregman divergences in Table 1. The most commonly used decomposable Bregman divergence is the **squared Euclidean distance** (SE). Of particular interest is the **generalized Kullback–Leibler divergence** (GKL) defined over $\mathbb{R}^d_+$. It reduces to the standard KL divergence for points on the **open standard simplex**, $\triangle^{d-1} = \{x \in \mathbb{R}^d : \sum_{i=1}^d x_i = 1,\ x_i > 0\}$. Another example is the **Itakura–Saito** (IS) divergence [25], which is useful for working with speech and sound data [20]. There are many other decomposable Bregman divergences, some inspired by popular tools in statistics such as the Bhattacharyya distance. Additionally, given functions of Legendre type, F_1 and F_2, we can form a new divergence generated by the weighted sum $\lambda F_1 + (1-\lambda)F_2$ for $\lambda \in [0,1]$ [46]. This new Bregman divergence which can be viewed as an interpolation between the two Bregman divergences D_{F_1} and D_{F_2}.

One outlier is the squared Mahalanobis distance [29] which is popular in statistics. While not decomposable, nearest neighbour problems involving this divergence can be reduced to the squared Euclidean distance that is decomposable [27]. Another one is the KL divergence on the *closed simplex*. While it does not fall under our definition, it can be treated as a limit case of the KL on the open simplex.

Overall, as the prominent Bregman divergences are decomposable, restricting our attention to decomposable divergences over open domains will not limit the choice of divergences handled in practice.

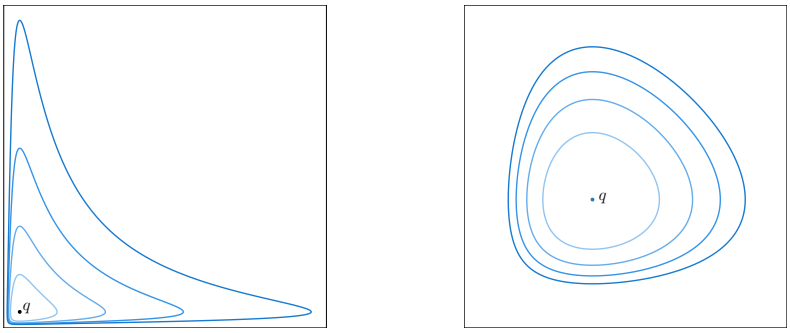


Figure 2 Left: primal Itakura–Saito balls. Right: primal generalized Kullback–Leibler balls.

Bregman balls. Due to the asymmetry, one can define two types of Bregman balls [43]. We start from the **primal Bregman ball** of radius $r \geq 0$ centered at q which is defined as

$$B_F(q; r) = \{y \in \Omega : D_F(q\|y) \leq r\}. \tag{2}$$

Namely, it is the collection of points with Bregman divergence measured *from* the center not exceeding r. See Figure 2 for an illustration.

The **dual Bregman ball** of radius $r \geq 0$ centered at q is defined as

$$B'_F(q; r) = \{y \in \Omega : D_F(y\|q) \leq r\}. \tag{3}$$

As seen in Figure 2, and observed in [13], primal Bregman balls can be non-convex (when viewed as a subset of Euclidean space). It is reasonable to question if all balls are necessarily connected. In Section 5 we will show that the balls are indeed connected, and emphasize why this property is crucial.

Bregman projections. While different in many aspects from metrics, Bregman divergences often exhibit familiar behaviours. We mention standard results related to projections, which we sharpen in the subsequent sections.

Given a Bregman divergence D_F, we consider the **Bregman projection** to a point q from a nonempty $C \subset \Omega$:

$$\mathrm{proj}_F(q, C) = \operatorname*{arginf}_{x \in C} D_F(x\|q).$$

When C is closed and convex, this projection exists and is unique. In this case, this point is declared to be the Bregman projection of q onto C [4]. In analogy with projection distance, we define the Bregman **projection divergence** as $D_F(\mathrm{proj}_F(q, C)\|q)$, the infimum of divergences from C to q. We state the following useful statement [13].

▶ **Lemma 1** (Bregman Projection [13]). *Given a nonempty closed convex set $C \subset \Omega$ and $q \in \Omega$, denote $q_C = \mathrm{proj}_F(q, C)$. For all $x \in C$:*

$$D_F(x\|q) \geq D_F(x\|q_C) + D_F(q_C\|q).$$

If C is an affine subspace, the above is an equality.

However, this definition and theorem only apply when D_F is computed to q from q_C. As $D_F(x\|y)$ may not be convex in the second coordinate, a projection, $\arg\min_{x \in C} D_F(q\|x)$, is not generally defined in Bregman divergence literature. However, by restricting the setup to axis-aligned boxes, we obtain a similar result working in both directions. We provide results for divergences computed *from* a query, but results and proofs for the reverse direction are analogous.

4 Kd-trees

We briefly overview a version of the Kd-tree data structure introduced by Bentley [11], focusing on nearest neighbour queries. It is a binary tree which encodes recursive partitioning of $\mathbb{R}^d$ (in practice: a sufficiently large box contained in it) into axis-aligned boxes. We consider a variant in which each node corresponds to an axis-aligned box and the data points are stored only in the leaves. We highlight the changes required to extend the standard method to the Bregman setting.

Construction. Kd-trees partition the space by cutting it with axis-aligned hyperplanes, often called splitting planes or cutting planes. The details of the construction, i.e. the order and locations of the splits, can have a significant impact on efficiency of Kd-trees [11,12,22,49]. However, the construction does not depend on the choice of a metric, or divergence. We therefore only mention that the standard splitting methods work in the Bregman case. It is worth emphasizing that once the tree is constructed, each query can be made efficiently using any *decomposable* divergence.

Nearest neighbour queries in the Bregman case. Consider a Kd-tree constructed from a finite set of **data points**, $X \subset \Omega \subset \mathbb{R}^d$. To simplify exposition, we focus on finding the single, exact nearest neighbour of the query point q among the points in X. More precisely, we consider the primal Bregman nearest neighbour, $\operatorname{argmin}_{x \in X} D_F(q\|x)$. The proposed algorithm is applicable for any Bregman divergence and is particularly efficient in the decomposable case.

We recall that the query can be performed using a simple recursive procedure. It traverses the tree trying to prune as many subtrees as possible, while guaranteeing that all viable candidates are considered. We overview the algorithm in the Bregman case now, and present an implementation in Section 7.

- The base case: divergences from q to the points stored in a leaf are compared with the divergence, r_{nn}, to the current best candidate which is updated if needed.
- We first visit child nodes based on the relative location of the query point and the splitting plane at this node. As such, this step does not depend on the choice of divergence.
- Moving back to the root, we visit the remaining subtrees only if they cannot be safely pruned. This is decided by what we call the **pruning test**. Conceptually, we rephrase it as an **intersection test** between the axis-aligned box corresponding to the remaining node and a primal Bregman ball. Specifically, we mean the ball centered at q of radius r_{nn}. Clearly, if the box and the ball are disjoint, all data points in the box are in the complement of the ball, and are therefore too far away to contribute.

Details. We mention that finding the dual Bregman nearest neighbour is completely analogous. Finding the k nearest neighbours, is another easy modification involving a priority queue. In this case r is set to the divergence to the current k-th nearest neighbour, or infinity. To allow approximate queries, the radius of the ball is decreased to $\frac{r}{1+\varepsilon}$. Our implementation supports all these options. We skip the details for brevity.

Pruning test in practice. In practice, to determine if a given box can be safely pruned, we will perform a Bregman projection of the query point onto the boundary of the box. Before we describe the implementation, we must prove that – despite the lack of symmetry and triangle inequality – correct and efficient pruning is possible. To this end, we focus on problems related to intersecting a Bregman ball $B \subset \Omega \subset \mathbb{R}^d$ with an axis-aligned box $A \subset \mathbb{R}^d$.

We divide our argument into two parts. Part I is presented in Section 5 and is more topological: we show that intersecting B with the boundary of A is an equivalent test. This argument works for arbitrary Bregman divergences, not only decomposable ones. Part II is presented in Section 6 and is more geometric: we replace the intersection test with a projection and show it can be computed in a simple efficient way. This part is specific to decomposable divergences.

5 Proof of pruning correctness

We consider a Bregman ball $B \subset \Omega \subset \mathbb{R}^d$ and an axis-aligned box $A \subset \mathbb{R}^d$. The intersection test checks if the intersection $A \cap B$ is nonempty. We first prove a crucial result which relies on the Legendre Transform.

Legendre transform. The Legendre transform is a tool from convex geometry [46]. In the context of Bregman divergences, it is used to map a Bregman generator over a domain into another generator over a possibly different domain [13]. In particular, it transforms primal balls in one domain into dual balls in the other domain, and vice versa. We will see one basic application of this tool.

More technically, given a function of Legendre type, $F : \Omega \to \mathbb{R}$, there exists the **Legendre transform** which maps F to a conjugate $F^* : \Omega^* \to \mathbb{R}$, where $\Omega^* = \{\nabla F(x) : x \in \Omega\}$ is the conjugate domain. Under this transformation, F^* is also a function of Legendre type [46] and we can define the Bregman divergence associated to F^*. We now use this result to prove the connectedness of Bregman balls.

▶ **Lemma 2** (Connectedness). *Primal and dual Bregman balls are connected.*

Proof. The dual balls are trivially convex [13], hence connected.

The primal balls are more interesting, so we show an explicit proof. First, recall that F is strictly convex and differentiable, implying it is continuously differentiable [46]. Therefore, the Legendre transform of F induces a homeomorphism $h : \Omega \to \Omega^*$. In particular, h maps dual balls in Ω^* to primal balls in Ω. Since connectedness is a topological property, any primal ball in Ω is connected as the homeomorphic preimage of a connected dual ball in Ω^*. ◀

This particular proof is useful for clarifying the importance of the three assumptions in the definition of the Legendre-type function. (I) and (II) gives *continuous* differentiability, and consequently the crucial homeomorphism. As for (III), let us show how things can go wrong without it. Specifically, if we allowed arbitrary convex restrictions of the domain. Consider Ω' as a restriction of Ω to the preimage under h of a non-convex primal ball in Ω^*. Since Ω' is convex, everything appears to work. However, the restricted h now maps Ω' to a non-convex conjugate domain, where the Legendre transform is not well defined. The above proof would therefore fail if we restricted the domain in this way and we could not rule out the existence of non-connected balls. Requiring condition (III) prevents us from making this mistake. Rockefellar [46] mentions that this is a very common mistake in general – it is also present in the Bregman divergence literature.

We now use connectedness for the following lemma.

▶ **Lemma 3** (Boundary Intersection). *Let $\Omega \subset \mathbb{R}^d$ be the domain for a Bregman divergence, D_F, $A \subset \mathbb{R}^d$ be an axis-aligned box of positive finite volume with boundary ∂A and $q \in \Omega \setminus A$ be the center of a Bregman ball B_F of finite radius r. If $B \cap \partial A = \emptyset$, then $A \cap \Omega$ lies in $\Omega \setminus B$.*

Proof. As A has finite volume, it is a codimension-1 topological sphere, and thus ∂A divides $\mathbb{R}^d$ into the inside and the outside, by the Jordan–Brouwer separation theorem. Because $B \cap \partial A = \emptyset$, $q \notin A$, and B is connected, we have that B is necessarily on the outside of ∂A and so $B \cap A = \emptyset$. Therefore, $A \cap \Omega$ indeed lies in the complement of B in Ω. ◀

Thus $B \cap \partial A = \emptyset$ implies that the divergence from q to each potential *data* point in A exceeds the radius of B, namely r. This means that the intersection test with the boundary of A is sufficient to safely prune points in the Kd-tree query. We omit the analogous case for dual balls. We mention that the finite volume assumption is just a technicality as in practice Kd-trees partition a box of finite volume.

6 Proof of pruning efficiency

In this section we show that the pruning test can be performed efficiently in the case of decomposable Bregman divergences. Specifically, we aim to update the projection in $O(1)$ running time, independently of the dimension of the ambient space. To this aim, we rephrase the pruning test in terms of a Bregman projection onto the *boundary* of the box. We call this the **boundary projection test**.

▶ **Lemma 4.** *Let D'_F be a decomposable Bregman divergence defined on $\Omega' \times \Omega'$. Then D'_F is a restriction of a divergence defined on an axis-aligned box.*

Proof. Let D'_F be a decomposable Bregman divergence. Then $F = \sum_{i=1}^d f_i$, where each f_i is a univariate function of Legendre type. Thus, each f_i has a convex domain in $\mathbb{R}$, ω_i, which must be an interval. Thus, D_F is a Bregman divergence defined on an axis-aligned box. Then, as both are generated by the same function of Legendre type, $D_F|_{\Omega' \times \Omega'} = D'_F$ and thus D'_F is a restriction of D_F. ◀

This lemma further emphasizes the importance of Legendre-type function assumption (III), as a restriction on a Bregman divergence's domain is induced by a domain restriction on the parametrization function F. For example, the GKL divergence on $\mathbb{R}^d_+$ may be restricted to the KL divergence on $\triangle^{d-1}$ but not to $(0,1)^d$.

For a given query and set of data points the nearest neighbours are identical under both D_F and a restricted D'_F, allowing use of either divergence. The assumption on the domain $\Omega = \prod_{i=1}^d \omega_i$ ensures that Lemma 5 and Corollary 1 apply to Kd-trees. This is important because Kd-trees decompose $\mathbb{R}^d$ and not the chosen domain of a Bregman divergence. Additionally, in the unrestricted domain, Lemma 6 enables efficient query processing and ensures that our underlying algorithm remains robust under any domain restriction.

From boxes to hyperplanes. We first consider a simplified problem, namely a Bregman projection onto a single axis-aligned hyperplane.

▶ **Lemma 5** (Axis-Aligned Projection)**.** *Let $F = \sum_{i=1}^d f_i$ be a decomposable function of Legendre type defined on an axis-aligned box, Ω.*

Let $P \subset \mathbb{R}^d$ be an axis-aligned hyperplane such that $P \cap \Omega \neq \emptyset$. Let q_P be the Bregman projection of a point $q \in \Omega$ onto P with respect to D_F, $\arg\min_{x \in P} D_F(x\|q)$. Then q_P coincides with the orthogonal projection of q onto P. (The same is true for $\arg\min_{x \in P} D_F(q\|x)$.)

Proof. Let P be an axis-aligned hyperplane orthogonal to the j-th standard basis vector. Specifically, each point $p \in P$ has its j-th coordinate fixed.

Because: (1) q is fixed; (2) p_j is fixed for each $p \in P$; and (3) D_F is decomposable, we can write $D_F(q\|p) = \sum_{i=1}^{d} D_{f_i}(q_i\|p_i) = D_{f_j}(q_j\|p_j) + \sum_{i\neq j} D_{f_i}(p_i\|q_i)$. To minimize $D_F(q\|p)$, we minimize $\sum_{i\neq j} D_{f_i}(q_i\|p_i)$, and since each D_{f_i} is a Bregman divergence, Bregman Nonnegativity (Property 1) applies. So each $D_{f_i}(q_i\|p_i) \geq 0$, with equality if and only if $p_i = q_i$. Consequently, $\arg\inf_{p\in P\cap\Omega} D_F(q\|p) = (q_1, q_2, \ldots, p_j, \ldots, q_d)$. Therefore, the Bregman projection of q onto P is precisely the orthogonal projection. ◀

Generally our Bregman projection *from* q, $\arg\min_{x\in P} D_F(q\|x)$, would *not* be considered a Bregman projection, and Lemma 1 would not apply to it. In our case, the two projections coincide, so we will refer to the resulting point as the Bregman projection onto the axis-aligned hyperplane. With this we get the following corollary.

▶ **Corollary 1** (Box Projection Divergence). *Let $F = \sum_{i=1}^{d} f_i$ be a decomposable function of Legendre type defined on an axis-aligned box. The Bregman projection divergence of $q \in \Omega$ onto A, with respect to the Bregman divergence generated by F, can be computed as*

$$\sum_{i=1}^{d} D_{f_i}(q_i\|p_i) = D_F(q\|p),$$

where p is the (squared) Euclidean projection of q onto A.

Back to the pruning test. To decide if the input points in the current box can be pruned, we compare two values. One is the divergence to the current best candidate; the second one is the projection divergence of q onto an axis-aligned box A. This also works for divergence computed in the reverse direction.

In the end, the situation is very simple. This simplicity allows us to compute the projection divergence in time $O(d)$ – exactly as in the (squared) Euclidean case.

Efficient projection. We focus on *maintaining* the projection divergence during the course of the query, rather than computing it every time. It turns out a single update can be done in constant time, independent of the dimension.

▶ **Lemma 6** (Updating Projection Divergence in Constant Time). *Let $F = \sum_{i=1}^{d} f_i$ be a decomposable function of Legendre type, where each f_i has domain $\omega_i \subseteq \mathbb{R}$. Then the projection divergence can be updated in constant time.*

Proof. Let $q \in \Omega$ and $B = \prod_{i=1}^{d}[a_i, b_i]$ be a box corresponding to a splitting node of our Kd-tree. By corollary 1, the Bregman projection of q onto B is on the boundary of B. Denote this Bregman projection $x = \arg\inf_{p\in B} D_F(q\|p)$. As x lies on the boundary, x_i is either a_i, q_i, or b_i.

For the box C corresponding to a child node, we change only one wall of B by the construction of the Kd-tree. Without loss of generality, $C = [c, b_1] \times \prod_{i=2}^{d}[a_i, b_i]$, with $a_1 < c < b_1$. For $y = \arg\inf_{p\in C} D_F(q\|p)$ we similarly have $y_i = x_i$ for $i = 2, \ldots, d$.

Since D_F is decomposable, $D_{f_i}(q_i\|x_i) = D_{f_i}(q_i\|y_i)$ for $i = 2, \ldots, d$. Thus $D_F(q\|\rho) = D_F(q\|\omega) - D_{f_1}(q_1\|\omega_1) + D_{f_1}(q_1\|\rho_1)$. This is illustrated in Figure 3.

Thus, as we move from a splitting node to its child, updating the projection divergence is independent of the dimension, d. ◀

Moving from a Kd-tree node to its child, the corresponding box shrinks along a single dimension. The projection divergence can therefore be updated using at most two divergence computations (one negative, one positive) along the same dimension. The update is $O(1)$ and independent of the embedding dimension. See Figure 3.

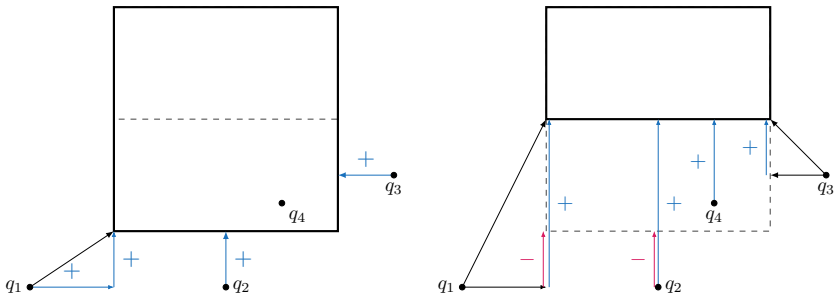


Figure 3 Left: Calculation of projection divergences of each point q_i onto the box decomposed as the sum of divergence computations along individual dimension. **Right:** Efficient update of projection divergence.

As Corollary 1 follows from Lemma 5 and Lemma 6 solely depends on the decomposable structure, the results apply for Bregman divergences computed in either direction.

7 Implementation

Our implementation is based on the ANN (Approximate Nearest Neighbour) C++ library by Mount and Arya [36]. It is an optimized library for Kd-trees.

Bregman query implementation. Algorithm 1 shows a C++ implementation of the Bregman query algorithm using this optimization (modulo unimportant technicalities). The code is structured after the implementation in the ANN library. We show only the part of the code for splitting nodes; handling leaf nodes is straightforward.

We assume that splitting nodes are instances of class `kd_tree_splitting_node`. Leaf nodes store input points and queries are handled with a linear search algorithm. Variable `eps` is used for approximate queries. Finally, `D_f` is assumed to compute the decomposable Bregman divergence *along a selected coordinate* – for all practical decomposable divergences this takes time $O(1)$ by utilizing Lemma 6 at lines 18 and 20. Line 18 adds the new projection divergence, `new_proj_div`, and the old projection divergence is removed in line 20. We remark that many implementations, including KDTree from the popular sklearn library [44], use a slower $O(d)$ approach, adding unnecessary work in higher dimensions.

The variable `knn_priority_queue` is used to maintain the k-nearest neighbours. To perform the dual query, just swap the parameters in the function used to compute `D_f`. The extra argument in `D_f` is just a technicality which allows one to use different 1-dimensional divergence depending on the currently considered dimension.

Expected computational complexity of a query. While we perform a single visit of an internal node in an optimal (constant) time, the expected complexity of the query remains an open problem. In particular, proving the $O(\log n)$ bound for uniformly distributed data is significantly harder in our setting. First, it is not clear what it means to uniformly distribute points with respect to a given divergence. Second, standard proofs rely on volume arguments [11] – but in our case the Euclidean volume of a Bregman ball depends on its location.

Algorithm 1 Bregman Kd-tree query implementation.

```
struct kd_tree_splitting_node : kd_tree_node {
  kd_tree_node *child_lower, *child_higher;
  int cut_dim;
  float cut_val, upper_bound, lower_bound;
  virtual void search(...);
};

using div_t = std::function<float(const float, const float, const int)>;

float D_GKL(const float x_i, const float y_i, const int dim) { // example
  return x_i*log(x_i) - x_i*log(y_i) - x_i + y_i;
}

void virtual kd_tree_splitting_node::search(const point& q,
                float box_proj_div, div_t D_f, float eps=0.0) {
  if (q[cut_dim] < cut_val) { // q lower than the cutting plane
    child_lower->search(q, box_proj_div, D_f, eps); // more promising child
    float new_box_proj_div = box_proj_div+D_f(q[cut_dim], cut_val, cut_dim);
    if (lower_bound > q[cut_dim])
      new_box_proj_div -= D_f(q[cut_dim], lower_bound, cut_dim);
    if (box_div*(1+eps) < knn_priority_queue->max_divergence())
      child_higher->search(q, new_box_proj_div, D_f, eps); // recursive call
  }
  else {/* analogous for q higher than the cutting plane... */ }
}
```

Although the constant time computations only rely on splitting planes being axis aligned, the impact of how splitting planes are chosen in the Bregman case similarly relies on the properties of Bregman balls. While the depth of the trees is not influenced by the choice of metric or divergence, the interactions of Bregman balls and axis-aligned boxes can be highly nonuniform.

These issues necessitate new, significantly more sophisticated, proof techniques. We leave this as an open problem, and show experimentally that the method performs well in practical situations.

8 Experiments

The main practical motivation of our work is to apply computational geometric algorithms to the point clouds produced by machine learning models. In particular, we wish to compare two collections of probabilistic (soft) predictions using the KL divergence. Efficient nearest neighbour queries are useful in this setting, however using the Euclidean tools in this setting can lead to severe discrepancies, seen in Section 8.1. Finally, the dimension is often not overly high (often 10-100) which gives hope that Kd-trees can be efficient.

We will benchmark Bregman Kd-trees for exact and approximate nearest neighbour queries and compare with other methods. Additional results are reported in Section A.

We stress that efficiency is just one aspect determining the practicality of a method. Bregman Kd-trees have several unique advantages which make them practical. For example, once a Bregman Kd-tree is constructed, each query can be performed using a different decomposable Bregman divergence.

Data sets. We use synthetic data, standard datasets, and probabilistic predictions coming from a machine learning model. For all data, we use 50,000 points and 10,000 query points.

In the machine learning setup, we consider popular image datasets CIFAR10 and CIFAR100. Each contains 50,000 training and 10,000 test images, with 10 and 100 different labels respectively. We train two neural networks, M_1 and M_2, on a classification task on CIFAR100. They achieve 80.22%, and 71.74% test accuracy respectively. From each model, we produce two sets of probabilistic predictions: $(\text{trn}_i, \text{tst}_i)$, for $i \in \{1, 2\}$. By $Q \to D$ we mean we query dataset D with queries Q. Since the network is trained to minimize the total KL divergence, these predictions lie on the $\triangle^{99} \subset \mathbb{R}^{100}$ equipped with KL divergence.

We also consider a model trained to 95.2% test accuracy on CIFAR10 and extract its probabilistic predictions on training and test points. These predictions are contained in $\mathbb{R}^{10}$. We also use the standard Corel Image Features data contained in $\triangle^{99}$.

Compiler and hardware. Software was compiled with Clang 14.0.3. The experiments were done on a single core of a 3.5 GHz ARM64-based CPU with 4MB L2 cache using 32GB RAM. We observed similar speed ups on an x86-64 CPU.

8.1 Nearest neighbour comparisons

One may assume that a Euclidean ball and Bregman ball with the same center and radius have large intersections and thus a query would return the same nearest neighbours in both cases. We experimentally show that this is not the case, which necessitates the usage of data structures specialized for the Bregman case.

We first compare the sets of nearest neighbours obtained by using Euclidean distance and the KL divergence. For $\text{tst}_1 \to \text{trn}_1$, we find the 10 nearest for each query in tst_1 with respect to the KL divergence and Euclidean distance. For each query, we compare the two sets of neighbours while disregarding order. Of the 10,000 queries, 9,962 had different sets of nearest neighbours with 134 having no common nearest neighbours. This is expected as the geometry of KL balls can vary depending on the location of the center whereas the Euclidean balls grow more uniformly. A lower dimensional example with three nearest neighbours computed can be seen in Figure 4. When the sets are ordered, the average of number of neighbours with matching indices is 0.8422 of 10, with only two queries having the same nearest neighbours in the same order. In conclusion, the reported nearest neighbours rarely coincide in the same order and often have different neighbours completely. Thus we cannot simply use fast Euclidean algorithms to analyze machine learning models trained using Bregman divergences.

8.2 Baselines

We stress that there are no robust, general libraries for the exact and approximate Bregman nearest neighbour computations. We compare our package to Cayton's experimental implementation of Bregman ball-trees [19] (**BBT**) for exact and approximate nearest neighbours. Two other available (experimental) implementations [40] are not usable, due to severe compilation issues (the code is non-portable) and limited documentation.

We additionally compare our package to the fastest implementation in NMSLIB [14] for Bregman divergences. We stress that these methods are recall-approximate: they may return the correct nearest neighbour, but offer no guarantees (unlike our method). These methods are therefore not in direct competition with our method, but it is interesting to observe the trade-offs between efficiency and guarantees.

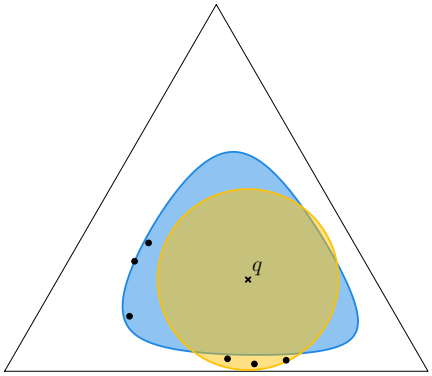


Figure 4 Three nearest neighbours of a query q with respect to the KL divergence and the Euclidean distance on $\triangle^2$. The blue area is the KL ball and the yellow is the Euclidean ball whose radius is determined by the respective third nearest neighbour.

Table 2 Build time comparison between Kd-trees and Bregman ball-trees for different divergences and data sets.

		Bregman Ball Trees [18]		
	Kd-tree	SE	KL	IS
trn_1	0.43s	5.75s	10.05s	7.71s
Corel 64	0.08s	1.30s	6.27s	6.08s
CIFAR10	0.04s	0.20s	1.30s	0.88s

8.3 Exact queries

In Table 2, we measure the construction time of Kd-trees and ball trees. Ball trees are constructed with Cayton's Bregman Ball trees. We note that the same Kd-tree works for any decomposable divergence for either direction, while a ball tree construction depends on the given divergence and direction.

Table 3 shows the speed up in finding nearest neighbours using our method compared to linear search and BBT. We use the KL, IS, BL, and an interpolated divergence. For exact queries with the KL divergence on the 100-dimensional CIFAR data sets we observe $\approx 100\times$ speed compared to the linear search. We compare our speeds to Cayton's Bregman ball trees, achieving minimum $3\times$ speed up with KL divergence and up to $20\times$ speed up for the IS divergence. As BBT has not implemented BL or interpolated divergences, these times are not available.

8.4 Approximate Bregman queries

Given ϵ, an **approximate nearest neighbour query** must return each nearest neighbour x' such that $D_F(q\|x') \leq (1+\epsilon)D_F(q\|x)$, where x is the true nearest neighbour.

To evaluate our method for approximate queries, we compare it with an implementation of Bregman Ball trees (**BB-trees**) by Cayton [18,19]. It is specialized for KL, with *experimental support* for IS. Unlike our method, extending it to other divergences is nontrivial.

Table 3 Runtimes of Kd-trees compared to Bregman ball-trees and linear search. Speed ups of Kd-trees compared to linear and BBT are listed. For example Kd-trees are 92.12× faster than a linear search on $\text{tst}_1 \rightarrow \text{trn}_1$. The interpolated (Int) divergence is 0.9KL + 0.1SE.

		$\text{tst}_1 \rightarrow \text{trn}_1$		$\text{tst}_2 \rightarrow \text{trn}_1$		Corel64		CIFAR10	
Kd-tree	KL	3.06s		3.66s		18.26s		0.30s	
	IS	24.95s		26.65s		67.95s		1.13s	
	BL	4.72s		5.78s		27.05s		0.49s	
	Int	5.55s		6.44s		21.94s		0.50s	
Linear	KL	281.90s	92.12×	286.63s	78.31×	177.77s	9.74×	30.53s	101.77×
	IS	277.87s	11.14×	274.58s	10.30×	173.79s	2.05×	30.20s	23.01×
	BL	88.59s	18.77×	87.84s	15.20×	55.46s	2.05×	8.89s	18.14×
	Int	309.21s	55.71×	311.63s	48.39×	196.99s	8.98×	34.20s	68.40×
BBT	KL	9.62s	3.14×	14.33s	3.92×	99.10s	5.43×	0.91s	3.03×
	IS	507.45s	20.34×	614.97s	23.08×	397.97s	5.86×	3.53s	3.53×
	BL	N/A		N/A		N/A		N/A	
	Int	N/A		N/A		N/A		N/A	

We compare the query times for the Kd-tree search to the Bregman ball tree search for a range of ϵ values in Figure 5. Our method is between 3-5 times faster for KL queries, and between 5-15 times faster for IS queries.

Our Kd-tree method works with arbitrary decomposable divergences (computed in either direction): one can either use a predefined divergence, or implement a custom one. This only requires implementing a single function in the user's code that computes the divergence – no changes to the Kd-tree library are required. This allows the method to work out of the box in various contexts.

The above is in contrast with Bregman Ball trees: they are more general but require tailoring to different divergences [19]. Also, there is a big difference between the simple squared Euclidean case and the Bregman case. Finding a projection onto a (dual) Bregman ball generally requires performing a 1-dimensional convex optimization. In practice, this is done using a binary search, with a full divergence computation at each step, making each step $\Omega(d)$. In our case this entire projection is $O(1)$. As evidenced by BB-tree's relatively lower performance for the IS divergence, extending BB-trees to other divergences poses an algorithmic challenge.

8.5 An unfair comparison with fast heuristics

The Non-Metric Space Library [14] by Naidan has algorithms adapted for working with Bregman divergences and other non-metric distances. Benchmarks for methods have been published specifically for the KL and IS divergences [45]. In particular, the small-world graph (SWG) [30] search is considered a state-of-the-art method. For brevity (and because these heuristic methods do not directly compete with methods for exact and approximate queries), we limit the comparison to this method.

Unlike Kd-trees, the SWG method does not offer guarantees on the number of correct nearest neighbours. While they tend to behave well in practice, Indyk and Xu showed [24] that such methods can fail catastrophically. Generally, recall tends to drop in higher dimensions. In particular, in dimension 100, 15.71% results for the KL divergence contained some incorrect nearest neighbours; in 4.54% of cases, *all* of the reported neighbours were incorrect.

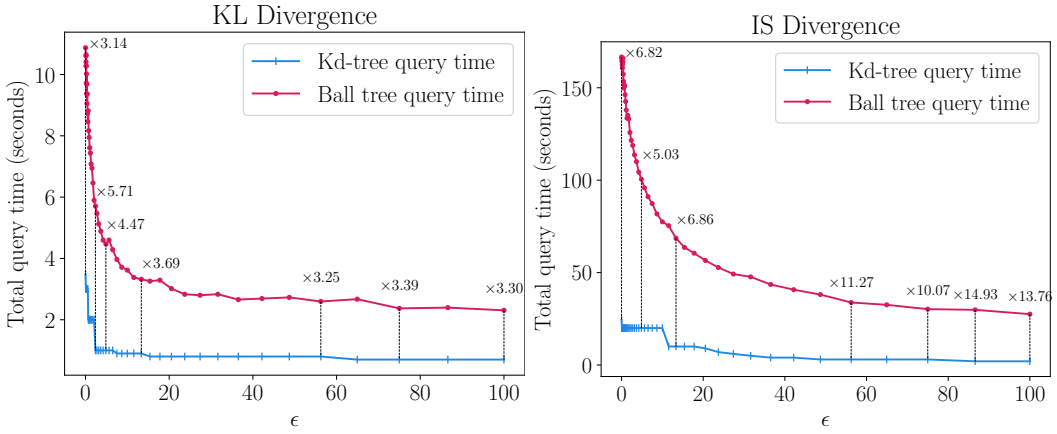


Figure 5 Total query time compared for $(1+\epsilon)$-approximate nearest neighbours for $\text{tst}_1 \rightarrow \text{trn}_2$ (lower is better). Left is KL and right is IS divergence. Starting from $\epsilon = 0.1$. Vertical bars mark the speed up of Kd-trees over ball-trees for a given ϵ.

Table 4 Comparing Kd-tree and SWG method for 10 nearest neighbours. For SWG, error frequency is number of times < 10 correct nearest neighbours are returned. Min recall is the minimum number of correct nearest neighbours. The interpolated (Int) divergence is 0.9KL+0.1Euc.

		Kd-tree Build time	Kd-tree query time	SWG build time	SWG query time	Avg recall	Error freq	Min recall	Min recall freq
KL	$\text{tst}_2 \rightarrow \text{trn}_1$	0.43s	5.76s	4.84s	1.32s	0.928	1571	0	454
	Corel64	0.08s	29.60s	6.99s	1.70s	0.998	152	7	5
	CIFAR10	0.04s	0.45s	2.90s	0.59s	0.998	76	1	1
IS	$\text{tst}_2 \rightarrow \text{trn}_1$	0.43s	29.29s	8.84s	2.31s	0.883	3292	0	387
	Corel64	0.08s	76.03s	17.13s	4.05s	0.900	4608	0	1
	CIFAR10	0.04s	2.05s	2.90s	0.85s	0.991	287	0	17
BL	All data sets			N/A	N/A	N/A	N/A	N/A	N/A
Int	All data sets			N/A	N/A	N/A	N/A	N/A	N/A

In any case, the benchmarks reported in Table 4 reveal an interesting trade-off. Compared to our implementation, the SWG offers faster query time (typically one order of magnitude faster), at the cost of slower build time (typically two orders of magnitude slower). We reiterate that these methods do not provide performance guarantees – while our method does. Overall, SWG is useful for performing numerous imprecise searches, while Kd-trees are useful for fewer searches or when guarantees are required.

9 Summary

We proved several results on Bregman divergences, demonstrating that the geometries they induce are well-behaved. In particular, we show that the lack of symmetry and triangle inequality does not preclude them from being used as a measurement for Kd-trees. This is perhaps unexpected, since the triangle inequality is typically used to prove the correctness of

Kd-trees. Furthermore, we show that certain additional properties of decomposable Bregman divergences enable an efficient query algorithm. These theoretical results provide the basis for an efficient implementation, whose properties are outlined below.

- Computational complexity: a crucial operation is optimized to work in $O(1)$ time. In comparison, several popular Euclidean Kd-trees implementations use a naive $O(d)$ algorithm.
- Speed: it is up to 100× faster than linear search and between 3 and 20× faster than competing methods on practical data in dimension 100.
- Simplicity: the algorithm is simple which makes it more likely to be adopted in practice.
- Ease of use: works for any decomposable Bregman divergence out of the box (competing approaches requires custom, nontrivial implementation for each divergence).
- Flexibility: handles exact and guaranteed ϵ-approximate Bregman queries with divergence computed in either direction.

From an applied perspective, one can now perform efficient queries for practical data measured with the KL divergence, in particular on medium-dimensional data coming from machine learning. This opens up new ways of using computational geometry algorithms within machine learning.

On the theoretical side, this work opens up new questions. Of primary importance is the expected computational complexity of a Kd-tree query. This problem is significantly more involved than in the Euclidean case, and will require developing novel proof techniques and deepening our understanding of the geometries induced by Bregman divergences.

References

1 Ahmed Abdelkader, Sunil Arya, Guilherme D da Fonseca, and David M Mount. Approximate nearest neighbor searching with non-euclidean and weighted distances. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 355–372. SIAM, 2019.

2 Amirali Abdullah, John Moeller, and Suresh Venkatasubramanian. Approximate bregman near neighbors in sublinear time: beyond the triangle inequality. In *Proceedings of the Twenty-Eighth Annual Symposium on Computational Geometry*, SoCG '12, pages 31–40, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2261250.2261255.

3 Marcel R. Ackermann and Johannes Blömer. Coresets and approximate clustering for bregman divergences. In *Proceedings of the 2009 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1088–1097, 2009. doi:10.1137/1.9781611973068.118.

4 Martin Adamčík. The information geometry of bregman divergences and some applications in multi-expert reasoning. *Entropy*, 16(12):6338–6381, 2014. doi:10.3390/e16126338.

5 Sunil Arya and David M. Mount. Algorithms for fast vector quantization. In *[Proceedings] DCC '93: Data Compression Conference*, pages 381–390, 1993. doi:10.1109/DCC.1993.253111.

6 Sunil Arya, David M. Mount, and Onuttom Narayan. Accounting for Boundary Effects in Nearest Neighbor Searching. In *Proceedings of the Eleventh Annual Symposium on Computational Geometry*, SCG '95, pages 336–344, New York, NY, USA, 1995. Association for Computing Machinery. doi:10.1145/220279.220315.

7 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, November 1998. doi:10.1145/293347.293348.

8 M. Aumüller, E. Bernhardsson, and A. Faithfull. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101–374, 2020.

9 Arindam Banerjee, Srujana Merugu, Inderjit S. Dhillon, and Joydeep Ghosh. Clustering with Bregman divergences. *Journal of Machine Learning Research*, 6(58):1705–1749, 2005. URL: https://jmlr.org/papers/v6/banerjee05b.html.

10 Heinz H Bauschke, Jonathan M Borwein, et al. Legendre functions and the method of random bregman projections. *Journal of convex analysis*, 4(1):27–67, 1997.

11 Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, 1975. doi:10.1145/361002.361007.

12 Jon Louis Bentley. K-d trees for semidynamic point sets. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, SCG '90, pages 187–197, New York, NY, USA, 1990. Association for Computing Machinery. doi:10.1145/98524.98564.

13 Jean-Daniel Boissonnat, Frank Nielsen, and Richard Nock. Bregman Voronoi diagrams. *Discrete and Computational Geometry*, 44:281–307, 2010. doi:10.1007/S00454-010-9256-1.

14 Leonid Boytsov and Bilegsaikhan Naidan. Engineering efficient and effective non-metric space library. In Nieves R. Brisaboa, Oscar Pedreira, and Pavel Zezula, editors, *Similarity Search and Applications - 6th International Conference*, volume 8199 of *Lecture Notes in Computer Science*, pages 280–293. Springer, 2013. doi:10.1007/978-3-642-41062-8_28.

15 Leonid Boytsov and Bilegsaikhan Naidan. Learning to prune in metric and non-metric spaces. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

16 Lev M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200–217, 1967.

17 L. Cayton. *Bregman Proximity Search.* Doctoral dissertation, UC San Diego, 2009.

18 Lawrence Cayton. Bbtrees. URL: https://github.com/lcayton/bbtree.

19 Lawrence Cayton. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 112–119, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1390156.1390171.

20 Minh N Do and Martin Vetterli. Wavelet-based texture retrieval using generalized Gaussian density and Kullback-Leibler distance. *IEEE Transactions on Image Processing*, 11(2):146–158, 2002. doi:10.1109/83.982822.

21 H. Edelsbrunner and H. Wagner. Topological data analysis with bregman divergences. In *"Proc. 33rd Ann. Symp. Comput. Geom., 2017", 1–16*, pages 1–16, 2017.

22 Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, September 1977. doi:10.1145/355744.355745.

23 Yikun Han, Chunjiang Liu, and Pengfei Wang. A comprehensive survey on vector database: Storage and retrieval technique, challenge. *arXiv preprint arXiv:2310.11703*, 2023. doi:10.48550/arXiv.2310.11703.

24 Piotr Indyk and Haike Xu. Worst-case performance of popular approximate nearest neighbor search implementations: guarantees and limitations. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2024. Curran Associates Inc.

25 Fumitada Itakura. Analysis synthesis telephony based on the maximum likelihood method. *Reports of the 6^{th} Int. Cong. Acoust.*, 1968.

26 Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. arXiv:1312.6114.

27 A. C. Koivunen and A. B. Kostinski. The feasibility of data whitening to improve performance of weather radar. *Journal of Applied Meteorology*, 38(6):741–749, 1999. doi:10.1175/1520-0450(1999)038<0741:TFODWT>2.0.CO;2.

28 Solomon Kullback and Richard A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

29 Prasanta Chandra Mahalanobis. On the generalised distance in statistics. *Proceedings of the National Institute of Sciences of India*, 2(1):49–55, 1936. Retrieved 2016-09-27.

30 Yu A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4):824–836, April 2020. doi:10.1109/TPAMI.2018.2889473.

31 Yu A Malkov and Dmitry A Yashunin. Hnswlib - fast approximate nearest neighbor search. https://github.com/nmslib/hnswlib, 2023.

32 Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014. doi:10.1016/J.IS.2013.10.006.

33 MaridDB. Mariadb vector. URL: https://mariadb.org/projects/mariadb-vector/.

34 Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. UMAP: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018. doi:10.21105/JOSS.00861.

35 MongoDB. Atlas Vector Search. URL: https://www.mongodb.com/products/platform/atlas-vector-search.

36 David Mount. *The ANN Programming Manual*.

37 Kevin P. Murphy. *Machine learning: A Probabilistic Perspective*. MIT press, 2012.

38 Frank Nielsen and Richard Nock. The dual voronoi diagrams with respect to representational bregman divergences. In *2009 Sixth International Symposium on Voronoi Diagrams*, pages 71–78, 2009. doi:10.1109/ISVD.2009.15.

39 Frank Nielsen and Richard Nock. The Bregman chord divergence. In *4th International Conference on Geometric Science of Information (GSI)*, pages 299–308. Springer, 2019. doi:10.1007/978-3-030-26980-7_31.

40 Frank Nielsen, Paolo Piro, and Michel Barlaud. Vptrees and bbtreepp. URL: https://github.com/FrankNielsen/FrankNielsen.github.io/tree/master/BregmanProximity.

41 Frank Nielsen, Paolo Piro, and Michel Barlaud. Bregman vantage point trees for efficient nearest neighbor queries. In *Proceedings - 2009 IEEE International Conference on Multimedia and Expo, (ICME)*, pages 878–881, 2009. doi:10.1109/ICME.2009.5202635.

42 Frank Nielsen, Paolo Piro, and Michel Barlaud. Tailored Bregman ball trees for effective nearest neighbors. In *Proceedings of the 25th European Workshop on Computational Geometry (EuroCG)*, pages 29–32, 2009.

43 R. Nock and F. Nielsen. Fitting the Smallest Enclosing Bregman Ball. In *Machine Learning: ECML 2005*, pages 649–656, October 2005.

44 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. doi:10.5555/1953048.2078195.

45 Alexander Ponomarenko, Nikita Avrelin, Bilegsaikhan Naidan, and Leonid Boytsov. Comparative analysis of data structures for approximate nearest neighbor search. In *Proceedings of the Third International Conference on Data Analytics*, January 2014.

46 Ralph T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

47 Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948. doi:10.1002/J.1538-7305.1948.TB01338.X.

48 Yang Song, Yu Gu, Rui Zhang, and Ge Yu. BrePartition: Optimized High-Dimensional kNN Search with Bregman Distances, 2020. arXiv:2006.00227.

49 Robert F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6:579–589, 1991. doi:10.1007/BF01759061.

50 Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11), 2008.

51 Ruben Winastwan. Vector Indexing in Milvus. URL: https://zilliz.com/learn/how-to-pick-a-vector-index-in-milvus-visual-guide.

52 Jun Zhang. Divergence Function, Duality, and Convex Analysis. *Neural Computation*, 16(1):159–195, January 2004. doi:10.1162/08997660460734047.

53 Zhenjie Zhang, Beng Chin Ooi, Srinivasan Parthasarathy, and Anthony K. H. Tung. Similarity Search on Bregman Divergence: Towards Non-Metric Indexing. *Proc. VLDB Endow.*, 2(1):13–24, 2009. doi:10.14778/1687627.1687630.

A Additional tests

A.1 Higher dimensional experiments

Baselines in higher dimensions. For 50,000 data points and 10,000 query points sampled sampled from the simplex, we can compare query times between Kd-trees and Cayton's BBT. In Table 5, we compare query times in higher dimensions. Although Kd-trees are often said to be slower in higher dimensions, we see a speed up in our method even at 500 dimensions.

Table 5 Kd-tree and BBT 10 nearest KL–neighbours search times for increasing dimensions.

Dimension	100	150	200	250	500	1000
Kd-tree query	208.39s	321.54s	469.21s	588.67s	1,229.72	2,689.70s
BBT query	247.80s	402.93s	517.07s	625.38s	1,237.84s	2,410.28s
Speed-up	$\approx 1.19\times$	$\approx 1.25\times$	$\approx 1.10\times$	$\approx 1.06\times$	$\approx 1.00\times$	$\approx 0.90\times$

SW-graph comparison. In comparison to Kd-trees, SWG has slow build times. We compare the SWG build time on 500,000 points in $\triangle^{999} \subset \mathbb{R}^{1000}$, with parameters reduced for speed while maintaining >0.9 average recall for 10 queries. SWG build time was 1464.07s, while Kd-tree build time and query time were 19.44s and 25.76s respectively. The build time for SWG is >30× longer than the sum of build and query time for Kd-trees.

A.2 Other exact query experiments

For these additional tests, $\square^{100}$ is a uniform sample of the unit cube with the same data sizes as above. In Table 6, we record total query time of other possible pairs of prediction data and $\square^{100}$.

Table 6 Additional total query time comparisons.

		$\text{tst}_1 \to \text{trn}_2$	$\square^{100}$	$\text{trn}_2 \to \text{trn}_1$	$\text{trn}_1 \to \text{trn}_2$
KL	Kd-Tree	4.46	17.22	12.73	264.84s
	Linear Search	285.16	1435.50	1431.93	407.80s
	Speed up	63.94×	83.36×	112.49×	1.54×
IS	Kd-Tree	26.76	121.49	180.55	170.58s
	Linear Search	277.77	1377.00	1365.40	397.75s
	Speed up	10.38×	11.33×	7.56×	2.33×
SE	Kd-Tree	0.41	1.89	1.87	89.95s
	Linear Search	23.32	116.44	116.466	24.36s
	Speed up	56.88×	61.61×	62.28×	0.27×

Skipping Ropes: An Efficient Gray Code Algorithm for Generating Wiggly Permutations

Vincent Pilaud
Departament de Matemàtiques i Informàtica, Universitat de Barcelona, Spain

Aaron Williams
Department of Computer Science, Williams College, Williamstown, MA, USA

Abstract

Wiggly permutations were introduced by Bapat and Pilaud (*Wigglyhedron* Mathematische Zeitschrift 2025). We positively answer one of their conjectures by finding a Hamilton path in the wiggly flip graph that is isomorphic to the wigglyhedron. Our path provides a Gray code in which successive wiggly permutations are obtained by a single jump or hop, meaning that one or two consecutive symbols move past some number of smaller symbols. The Gray code has a simple greedy description that produces a recursive zig-zag pattern reminiscent of *plain changes* for permutations. More broadly, our results extend Algorithm J and the series of papers on zig-zag languages initiated by Hartung, Hoang, Mütze and Williams (*Combinatorial Generation via Permutation Languages* SODA 2020). Finally, we use *wiggly changes* as the basis for an $\mathcal{O}(n)$-time delay generation algorithm.

2012 ACM Subject Classification Mathematics of computing → Permutations and combinations; Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Paths and connectivity problems

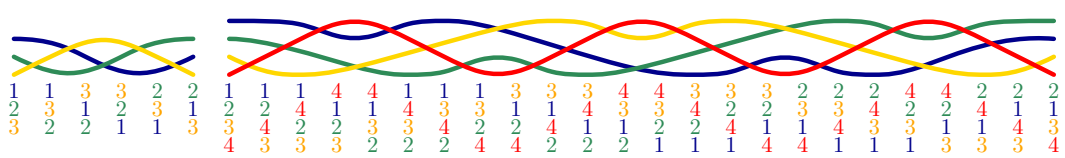
Keywords and phrases permutations, wiggly permutations, pattern avoidance, permutahedron, wigglyhedron, Hamilton path, flip graph, Gray code, combinatorial generation, generation algorithm

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.46

1 Introduction

We use a classic order of permutations called *plain changes* as inspiration for ordering an interesting new subset of permutations called *wiggly permutations*. Plain changes can be described as a *swap Gray code of permutations*, meaning that it lists all $n!$ permutations so that successive permutations differ by a swap (i.e., the transposition of two adjacent symbols)[1]. It is arguably the most prominent non-lexicographic order of permutations, and it is often used when generating permutations efficiently. Likewise, we use our *wiggly changes* as the basis for the first efficient generation algorithm of wiggly permutations. This introductory section also discusses how our results can be seen as the next step in the broad generalization of plain changes from the recent *Permutation Languages* series of papers.

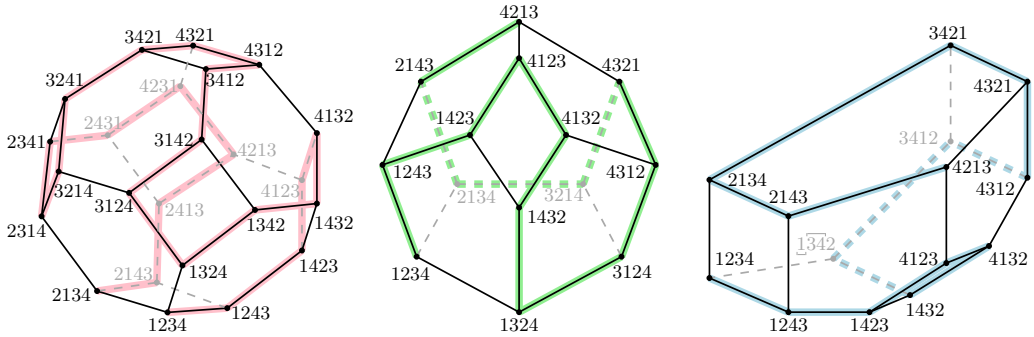
[1] For excellent background information on Gray codes see Mütze's dynamic survey [34].



(a) $n = 3$. **(b)** $n = 4$.

Figure 1 Plain changes is a swap Gray code for the permutations of $\{1, 2, \ldots, n\}$. It is visualized here with one rope per number, where each swap moves a larger rope over an adjacent smaller rope.

© Vincent Pilaud and Aaron Williams;
licensed under Creative Commons License CC-BY 4.0



(a) Permutahedron for $n = 4$. **(b)** Associahedron for $n = 4$. **(c)** Wigglyhedron for $n = 4$.

Figure 2 Polyhedra with Hamilton paths on their 1-skeletons. The vertices are (a) permutations, (b) 231-avoiding permutations, and (c) wiggly permutations. The edges are cover relations in Hasse diagrams (see Figure 6) and are (a) swaps, (b) minimal jumps, and (c) wiggly flips. The paths follow (a) plain changes, (b) Algorithm J [16], and (c) wiggly changes. The paths are all created greedily by Algorithm F via the edge labels in Definition 6. We also generate wiggly changes efficiently.

1.1 Plain Changes for Permutations

In the 17th century bell-ringers faced a combinatorial problem. They wished to list all $n!$ permutations of $[n] = \{1, 2, \ldots, n\}$ so that successive permutations differ by a swap (i.e., the transposition of two adjacent symbols). In this context, a permutation of $[n]$ represents the order in which n large church bells can be rung by n individuals, the ringing of all $n!$ permutations is a *peal*, and the closeness condition on successive permutations represents a physical limitation in altering the cadence of each bell. Over the course of several decades the community devised a pattern for $n = 4$ bells and then generalized it to arbitrary n [21]. The resulting *plain changes* is often illustrated using ropes, where each rope illustrates the movement of an individual bell. Figure 1 provides the order for $n = 3$ and $n = 4$ along with a subtle but important graphical feature: larger ropes go over smaller ropes. For example, we view the transition from 1243 to 1423 not just as a swap, but as 4 moving over 2. To emphasize this perspective, we will often write such a transition as $1\underline{2}\overline{4}3$ to $1\overline{4}\underline{2}3$.

In the 20th century computer scientists faced the same combinatorial problem. This time the goal was to generate all $n!$ permutations as quickly as possible within a computer program. Plain changes was discovered independently multiple times, and the pattern became known in this community as the *Steinhaus-Johnson-Trotter algorithm* [51, 19, 54].

We can visualize this type of problem using a *flip graph*. Its vertices are the objects to create and its edges represent the allowable changes between objects. A solution is a Hamilton path. We refer to the order of successive objects on the path as a *Gray code* for the associated objects using the associated operations or changes. The flip graph for the bell-ringing problem is shown in Figure 2a for $n = 4$ and it can be described in a number of other ways. For example, it is the Cayley graph of the symmetric group under swaps [12]. It is also the Hasse diagram of the permutations under the weak order [55] as seen in Figure 6a. The flip graph is also isomorphic to the permutahedron [38] as seen in Figure 2a.

Plain changes is typically defined using *local recursion* meaning that each permutation of $[n-1]$ is expanded into n permutations of $[n]$. The first n permutations in the order for $[n]$ is obtained by sweeping the value n from right-to-left through the first permutation in the order for $[n-1]$. Then n is swept from left-to-right through the second permutation in the order for $[n-1]$, and so on. This zig-zag pattern is illustrated by the red rope in Figure 1.

Alternatively, plain changes for $[n]$ can be defined using a simple greedy algorithm without any reference to the order for $[n-1]$. The algorithm starts a list from $12\cdots n$, then repeatedly extends the list to a new permutation as follows: from the most recently added permutation swap the largest value that gives a new permutation. For example, consider the algorithm once it has generated the partial list $1234, 1243, 1423, 4123$. At this point the most recently added permutation is 4123. The algorithm cannot swap the largest value 4 to the left (since it is in the leftmost position) or to the right (since $\overline{4}\underline{1}23 = 1423$ is already in the list). So it considers swapping the next largest value 3, which produces the next permutation in the list $41\underline{2}\overline{3} = 4132$. Note that the algorithm description is potentially ambiguous since it doesn't specify whether a particular value should prefer to swap to the left or right, however, it turns out that such a choice never arises. This greedy interpretation was introduced at WADS 2013 as part of a larger investigation that showed how simple greedy algorithms can recreate the recursive definitions used in classic Gray code constructions [58].

1.2 Jump Gray Codes for Zig-Zag Languages

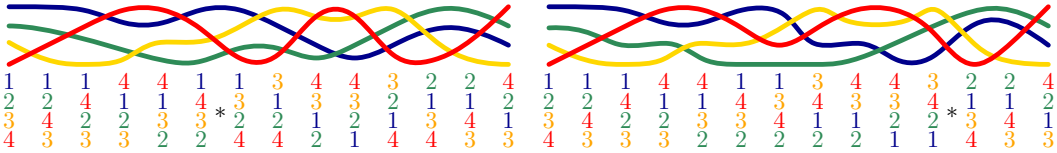
Permutations can be used to represent other combinatorial objects. For example, a permutation $p_1p_2\cdots p_n$ *avoids* the pattern 123 if there is no increasing subsequence of length three. That is, $\nexists i < j < k$ with $p_i < p_j < p_k$. These permutations were first studied over a hundred years ago by MacMahon [29] and were shown to be counted by the Catalan numbers. Knuth then showed that the same numbers count the 231-avoiding permutations. Both sets of permutations have natural bijections with other objects counted by the Catalan numbers, including binary trees with n nodes [50]. For basic concepts in pattern avoidance see [2].

Using these types of bijections we can create lists of various combinatorial objects by creating lists of corresponding permutations. In this context it is sometimes the case that a small change in the permutation corresponds to a small change in the combinatorial object that it represents. In other words, a Gray code for the permutations may provide a Gray code for the combinatorial objects. This connection has been explored in great detail by the recent *Permutation Languages* series of papers starting from [16]. In these papers, it is shown that every "zig-zag language" has a *jump Gray code*, meaning that consecutive permutations differ by moving one larger symbol over some number of smaller symbols. Furthermore, these Gray codes can be seen as natural generalizations of plain changes. They are generated by a simple greedy algorithm called Algorithm J: jump the largest possible value the shortest possible distance that creates a new permutation in the language. When applied to 231-avoiding permutations the algorithm generates a Hamilton path in a flip graph that is isomorphic to the associahedron [38] as shown in Figure 2b. And just like in plain changes, this algorithm creates a locally recursive zig-zag pattern, except this time a value may skip over some positions when it sweeps from side to side as shown in Figure 3a.

1.3 Not-So Plain Changes: Wiggly Changes

We consider *wiggly permutations* which were recently introduced by Babat and Pilaud [1]. Informally, a wiggly permutation restricts the values between pairs of the form $(2j-1, 2j)$. More specifically, if the smaller value $2j-1$ appears to the left/right of the larger value $2j$, then the values between them must be larger/smaller. To make this idea more concrete, consider a wiggly permutation π for $n = 8$ and the restrictions placed on it by the pair $(3, 4)$.

- If $\pi =$ _3_4_, then the middle piece is restricted to values selected from $\{5, 6, 7, 8\}$.
- If $\pi =$ _4_3_, then the middle piece is restricted to values selected from $\{1, 2\}$.



(a) 231-avoiding permutations. The order uses jumps that are not swaps (e.g., the 4 jumps in $*$).

(b) Wiggly permutations. The order uses hops that are not jumps (e.g., the 34 hop together in $*$).

Figure 3 Gray codes for two permutations languages visualized using $n = 4$ ropes. To avoid invalid permutations, the red rope changes speed in (a), and also reverses direction earlier in (b). These two "skipping rope" patterns (and plain changes) are generated by Algorithm F.

Just as permutations are the vertices of the permutahedron, and 231-avoiding permutations are the vertices of the associahedron, the wiggly permutations are the vertices of the wigglyhedron (see Figure 2). When viewed as a graph, the edges of the wigglyhedron represent wiggly flips, which are formally defined in Section 3 along with other wiggly concepts.

▶ **Conjecture 1** ([1]). *The wigglyhedron has a Hamilton path. In other words, there is a Gray code for wiggly permutations using wiggly flips.*

We affirm a stronger version of Conjecture 1 that only allows jumps and hops. A *hop* is like a jump except that a pair of consecutive and adjacent symbols moves over some adjacent smaller values. For example, $6578\underline{413}2 = 65413782$ is a hop as the consecutive and adjacent pair 78 moves over the adjacent smaller values 413. It is also *minimal* since the larger symbols move the shortest possible distance to create another valid wiggly permutation.

▶ **Theorem 2.** *There is a minimal jump and minimal hop Gray code of wiggly permutations.*

1.4 Generation Algorithms

We proved Theorem 2 using a simple greedy algorithm that generalizes Algorithm J. Furthermore, the Gray code creates a locally recursive zig-zag pattern. Unlike the jump Gray codes for zig-zag languages, more than one symbol moves between the end of one sweep and the start of the next sweep (see Figure 3b). Nevertheless, we use our new Gray code order in an efficient generation algorithm. It runs with $\mathcal{O}(n)$-*time delay* meaning each wiggly permutation of $[n]$ is generated exactly once, and worst-case $\mathcal{O}(n)$-time is used between successive wiggly permutations; further analysis may reduce this to amortized $\mathcal{O}(1)$-time delay.

Our algorithm is the first to generate wiggly permutations, and it adds to a new trend in the area of combinatorial generation [42, 24]. When developing a generation algorithm, one must first decide on which order to follow. The most common choices are lexicographic orders [43, 60] followed by Gray code orders. While Gray code orders have certain benefits they are typically assumed to be more challenging. This assumption has long been false for permutations [49] and other basic combinatorial objects [11, 21] including combinations [46]. Gray code algorithms for more general objects have also been simplified over the years – compare the pioneering Gray code algorithms for multiset permutations [22, 23] to more modern examples [57]. However, Gray code algorithms have almost always been developed after lexicographic algorithms. But this practice has begun to change with the rise of greedily constructed Gray codes (see Section 2) which often produce orders that can be used in generation algorithms that are simpler, faster, and easier to analyze. Other recent examples of this trend include the generation of various permutation languages representing rectangulations [30] and s-Stirling permutations representing s-increasing trees [3].

1.5 Outline

Section 2 formulates the greedy Gray code algorithm using edge-labeled graphs. Section 3 provides background on wiggly concepts. Section 4 describes *wiggly changes* both greedily and using local recursion. Finally, Section 5 generates our Gray code efficiently. Appendix A has a full Python implementation. Wiggly changes for $n = 4, 5, 6$ appears in Table 2 and can serve as a handy reference.

2 Greedy Gray Code Algorithm

The *greedy Gray code algorithm (GGA)* was originally presented using an initial object $x \in \mathbf{X}$ and a prioritized list of operations $\mathcal{O} = o_1, o_2, \ldots, o_k$ where $o_i : \mathbf{X} \to \mathbf{X}$ for all $1 \leq i \leq k$. We now reformulate the approach with edge-labeled flip graphs[2]. For example, plain changes is created in the permutahedron by labeling the edges with the larger value that is swapped.

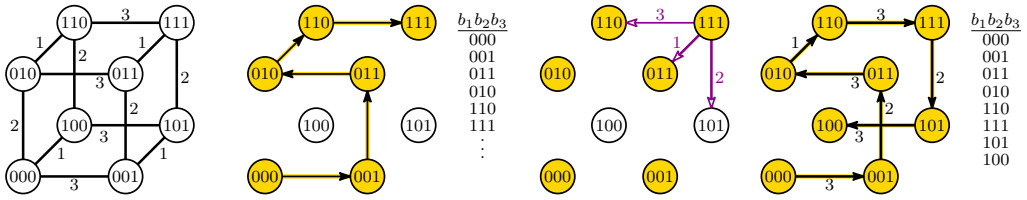
Algorithm F (Greedy max-flips).

This algorithm attempts to greedily traverse a Hamilton path in an edge-labeled flip graph starting from an initial node.

1. [Initialize] Start a path at the initial node.
2. [Greedy] Extend the path to a new node using an edge with a label that is as large as possible. If there is no such edge (i.e., every neighbour is in the path) or ambiguity (i.e., multiple suitable edges with the same label), then halt. Otherwise repeat 2.

Figure 4 shows how Algorithm F creates the *binary reflected Gray code* (BRGC) [14]. The edges of the hypercube are labeled by the index of the bit that differs between adjacent vertices. The BRGC was previously described in [58] as "greedily flip the rightmost bit". Algorithm F is sensitive to the choice of edge labels. For example, a Hamilton path is not created in the permutahedron when labeling its edges by the index of the swap (see Figure 5).

One of the most famous Gray codes is the middle levels theorem by Mütze [33], which is a Hamilton cycle in the middle two levels of the hypercube for odd n. While the proof has been simplified over the past decade [35], it seems very unlikely that the theorem could be proven with any simple greedy construction. Nevertheless, the GGA has been used to reinterpret many previous Gray code results that were originally described recursively. For example, see the Lucas–Roelants van Baronaigien-Ruskey tree rotation Gray code [28] described in [15].



(a) Edge-labeled flip graph. **(b)** Partial path from 000. **(c)** Choose next edge. **(d)** Completed path.

Figure 4 Algorithm F is run on the hypercube in (a) starting at node 000. The partial path after five steps is shown in (b). The next step is shown in (c) where the edges incident to the most recently added node 111 are considered; the largest label that leads to a new node is 2, so that edge is selected. It halts in (d) with a Hamilton path that follows the binary reflected Gray code.

[2] This reformulation is too strong as it can generate any Hamilton path (i.e., label edges on the path with 1 and others with 0). The algorithm is intended to be used with simple locally-defined labels.

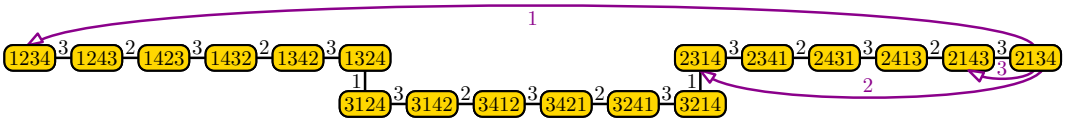


Figure 5 Algorithm F does not work on the flip graph of S_n with swaps (i.e., the permutahedron) with the (smaller) index of the swap as edge labels. When $n = 4$ the above path of length 18 is created from the initial node 1234. The algorithm terminates at 2134 because each of its edges is incident to a previously visited node. Note that none of the vertices starting with 4 were visited.

The GGA has also been used to create new Gray codes. One of the most striking new results is a simple greedy algorithm for generating an exchange Gray code for the bases of any matroid [31]. In particular, the spanning trees of any graph are generated by giving its edges distinct labels and then by greedily performing any edge exchange that minimizes the larger of the two labels. When approaching a new problem like Conjecture 1 we can try labeling the flip graph's edges (i.e., prioritizing the flips) in various ways. But the "correct" prioritization may not be obvious, so it is important to be creative and exercise patience. For example, the second author tried hundreds of greedy approaches for generating spanning trees before the simple solution described above was found by Merino [31] (also see [32]).

The GGA is inefficient as it remembers visited objects. However, it tends to produce simple orders that can be re-expressed to support efficient generation. We aim for *history-free* (i.e., previous objects are not remembered) and *iterative* (i.e., non-recursive) algorithms. Prior examples of this involve pancake flipping for signed and colored permutations [47, 5].

3 Wiggly Concepts

This section formally defines the objects and flip operations in our Gray code. The latter requires a discussion of the weak order. Additional motivation for our results is then provided.

3.1 Wiggly Permutations

Wiggly permutations restrict the values that are between *partners* which are $(2j-1, 2j)$ pairs. When the smaller odd-value is to the left of its larger even-value partner, then the values between them are larger. In contrast, if the larger even-value is to the left of the smaller odd-value, then the values between them are smaller. This is formalized in Definition 3.

▶ **Definition 3.** *A permutation π over $[n]$ is* wiggly *if the following two points hold for all pairs of values the form $(2j-1, 2j)$.*

- *Upward order. If $2j-1$ is before $2j$ in π, then symbols between them in π are larger.*
- *Downward order. If $2j$ is before $2j-1$ in π, then symbols between them in π are smaller.*

In other words, partners in their *upward order* can only have larger values between them, while those in their *downward order* can only have smaller values between them. Note that we allow n to be odd in Definition 3 (cf. [1]), and in this case n doesn't have a partner.

The relatively simple conditions in Definition 3 cannot be expressed using standard concepts from pattern avoidance [2]. However, they forbid pairs of partners from interacting with each other in particular manner. More specifically, a pair of partners in upward order can be beside each other, or nested with the larger partnership inside. For example, the partners $(1, 2)$ and $(3, 4)$ can appear in upward order as $-1-2-3-4-$, or $-3-4-1-2-$, or $-1-3-4-2-$. However, they cannot be interlaced as in $-1-3-2-4-$ or $-3-1-4-2-$. This is formalized for upward and downward orders in the following lemma.

▶ **Lemma 4.** *Let π be a wiggly permutation with distinct partnerships $\{u, u'\} = \{2x-1, 2x\}$ and $\{v, v'\} = \{2y-1, 2y\}$. If $\pi = -u-v-u'-v'-$, then the partners have opposite order.*

Proof. If $u < v$ then both partners cannot be upward by $v{-}u'{-}v'$, nor downward by $u{-}v{-}u'$. If $u > v$ then both partners cannot be downward by $v{-}u'{-}v'$, nor upward by $u{-}v{-}u'$. ◀

Let $W_n \subseteq S_n$ be the set of wiggly permutations over $[n]$. For example, $W_2 = \{12, 21\}$ and the sets for $n = 3, 4$ appear below; Table 2 contains W_5 and W_6 in our wiggly changes order.

$$W_3 = \{123, 132, 213, 312, 321\}$$
$$W_4 = \{1234, 1243, 1342, 1423, 1432, 2134, 2143, 3412, 3421, 4123, 4132, 4213, 4312, 4321\}$$

The number of wiggly permutations is not yet known in general. However, the counts for small n have been computed via recurrences on generating functions [1] and the sequence does not appear in the Online Encyclopedia of Integer Sequences (OEIS) [18].

Table 1 The number of wiggly permutations $|W_n|$ for small n.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	5	14	51	176	807	3232	17449	78384	479897	2366248	16041147	85534176	631596455	3602770400

3.2 Weak Order

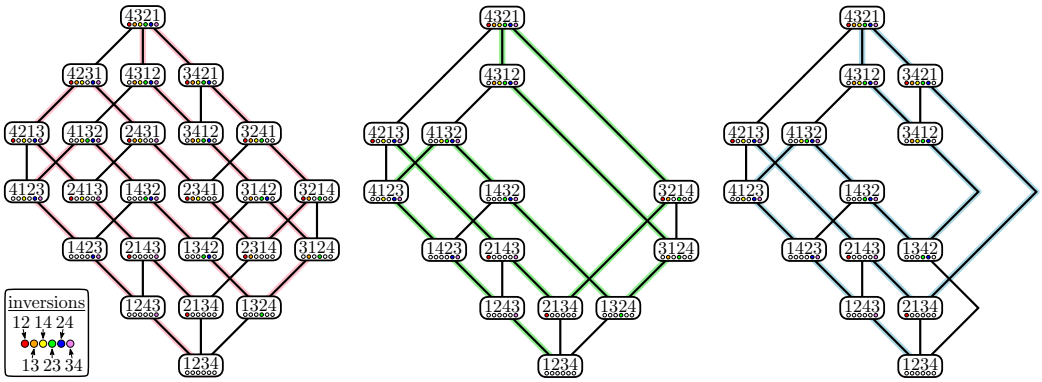
An *inversion* in a permutation $\pi = \pi_1\pi_2\cdots\pi_n$ is a pair of values that is out of order relative to the identity $12\cdots n$. In other words, it is a smaller value that appears to the right of a larger value. The *inversion set* of a permutation is its set of inversions.

$$\mathsf{inv}(\pi_1\pi_2\cdots\pi_n) = \{(\pi_i, \pi_j) \mid \pi_i < \pi_j \text{ and } i > j\} \tag{1}$$

Each permutation of S_n is uniquely determined by its inversion set.

A swap changes the inversion set by adding or removing one inversion. More specifically, if the larger value swaps to the left, then an inversion is added, and if it swaps to the right, then an inversion is removed. For example, $\mathsf{inv}(52\underline{\overline{4}1}36) \smallsetminus \{(1,4)\} = \mathsf{inv}(521436)$. In this way, plain changes is a Gray code that changes the inversion set by one inversion per operation. More broadly, the number of inversions added or removed by a jump is equal to the length of the jump. For example, $\mathsf{inv}(52\underline{\overline{4}1}\underline{36}) \smallsetminus \{(1,4),(3,4)\} = \mathsf{inv}(521346)$. Furthermore, Algorithm J only makes *minimal jumps* meaning that the jump is as short as possible to create another permutation in the underlying language. Thus, Algorithm J also produces Gray codes that change the inversion set in a minimal way. The same will be true for the wiggly flip operations that we apply to wiggly permutation. More specifically, it will add or remove a minimal subset of inversions to create another wiggly permutation. To formalize and contextualize this idea we will introduce some additional background material. Then we will provide a direct definition of a wiggly flip in Definition 5. A similar presentation is provided in [1].

The *weak order* of permutations is a partial order based on setwise inclusion of inversion sets. In other words, if $\pi, \sigma \in S_n$ then we consider $\pi < \sigma$ if $\mathsf{inv}(\pi) \subsetneq \mathsf{inv}(\sigma)$. Similarly, distinct permutations are incomparable if neither inversion set is a subset of the other. Given a subset $S \subseteq S_n$, the *Hasse diagram* is a graph whose vertices are S and whose edges are the cover relations, where π is *covered* by σ if $\mathsf{inv}(\pi) \subsetneq \mathsf{inv}(\sigma)$ and $\nexists \tau$ where $\mathsf{inv}(\pi) \subsetneq \mathsf{inv}(\tau) \subsetneq \mathsf{inv}(\sigma)$. In other words, $\pi < \sigma$ and there is no other permutation τ with between them $\pi < \tau < \sigma$. When $S = S_n$ (i.e., all permutations) the cover relations are swaps. Similarly, when S is the set of 231-avoiding permutations $\mathsf{Av}_n(231)$, the cover relations are minimal jumps. Likewise, when $S = W_n$ (i.e., wiggly permutations) the cover relations are *wiggly flips*. In other words, our Gray code operations involve adding or removing minimal subsets of inversions. Figure 6 illustrates Hasse diagrams for the three aforementioned subsets of permutations of S_4.



(a) All permutations S_4. **(b)** Pattern avoiding $\mathsf{Av}_4(231)$. **(c)** Wiggly permutations W_4.

Figure 6 Hasse diagrams for sets of permutations under the weak order. The cover relations are (a) swaps, (b) minimal jumps, and (c) wiggly flips. The graphs are isomorphic to those in Figures 2 and 9. The dots indicate which pairs of symbols are inverted in each permutation as per the legend in (a). For example, the hop between $\underline{21\overline{34}}$ and $\overline{34\underline{21}}$ in (c) changes the number of inversions by ± 4.

These concepts have lovely generalizations to s-permutations [6], and this paper inspired a new Hamilton path in the s-permutahedron [3]. Readers may enjoy other classic and new works on related concepts including the weak order and associahedron [53, 40, 41, 25, 36, 37].

3.3 Wiggly Flips

Wiggly flips are defined in terms of cover relations, but this is somewhat of an indirect description. We now develop some intuition on our way to a direct definition. As a first step, note that the graphs in Figure 6 are 3-regular (i.e., each vertex has degree 3). More generally, all three are $(n-1)$-regular graphs as vertices have one neighbor associated with each *ascent* (i.e., $\pi_{j-1} < \pi_j$) and *descent* (i.e., $\pi_{j-1} > \pi_j$). We discuss this below; see [1] for proofs.

Note that $12\cdots n \in W_n$ and $n\cdots 21 \in W_n$. In other words, the permutations with no inversions and all inversions are wiggly. Therefore, if we add one inversion (i.e., swap an ascent), then we can continue "up" to at least one vertex in the Hasse diagram. Similarly, if we remove one inversion (i.e., swap a descent), then we can continue "down" to at least one vertex. Thus, there is at least one cover relation associated with each ascent and descent.

To illustrate this point, the neighborhood of $\pi = 6\,4\,1\,3\,5\,7\,9\,10\,8\,2 \in W_{10}$ is in Figure 7. Consider its ascent $\mathbf{5}\,\mathbf{7}$. By our discussion, π is covered by at least one wiggly permutation whose inversion set includes $\mathsf{inv}(\pi) \cup \{(5,7)\}$. Now consider the partners of $\mathbf{5}$ and $\mathbf{7}$. Note that the smaller partners $(\mathbf{5}, 6)$ are downward and the larger partners $(\mathbf{7}, 8)$ are upward in π. Thus, the symbols in $6\,4\,1\,3\,\mathbf{5}$ cannot be between $\mathbf{7}$ and 8, and similarly, the symbols in $\mathbf{7}\,9\,10\,8$ cannot be between 6 and $\mathbf{5}$. So adding inversion $(\mathbf{5}, \mathbf{7})$ forces the run $6\,4\,1\,3\,\mathbf{5}$ to move left of the run $\mathbf{7}\,9\,10\,8$. The transposition of these two abutting runs is shown below.

$$\pi = 6\,4\,1\,3\,\mathbf{5}\,\mathbf{7}\,9\,10\,8\,2 \qquad \underline{6\,4\,1\,3\,\mathbf{5}}\,\overline{\mathbf{7}\,9\,10\,8}\,2 = \mathbf{7}\,9\,10\,8\,6\,4\,1\,3\,\mathbf{5}\,2 = \sigma. \tag{2}$$

Note that this change adds a total of $5 \cdot 4 = 20$ inversions (i.e., every value in $7\,9\,10\,8$ is inverted with every value in $6\,4\,1\,3\,5$) without removing any, so $\mathsf{inv}(\pi) \subsetneq \mathsf{inv}(\sigma)$ and $\pi < \sigma$. With some additional reasoning, we can conclude that this is the unique minimal addition in terms of inversions. In other words, there is no other wiggly τ with $\mathsf{inv}(\pi) \subsetneq \mathsf{inv}(\tau) \subsetneq \mathsf{inv}(\sigma)$. To briefly justify this, note that the internal values in $\overline{7\,9\,10\,8}$ must move left with 8 since existing inversions $(8,9), (8,10) \in \mathsf{inv}(\pi)$ cannot be removed. Similarly, the internal values in

$\underline{4\,1\,3}\,\mathbf{5}$ must move right with 4 lest additional inversions $(1, 4)$ or $(3, 4)$ be added. Finally, neither run needs to move past additional values. In particular, the runs cannot improperly interlace with another partnership otherwise the initial permutation violated Lemma 4.

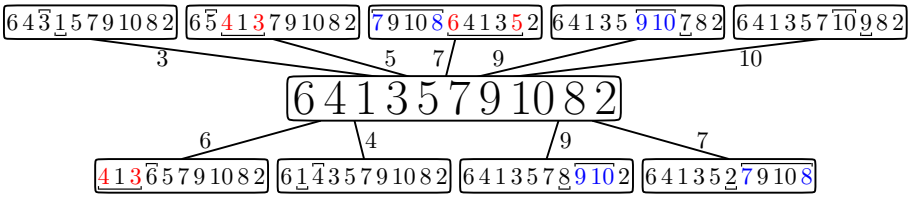


Figure 7 Neighbors of wiggly permutation $\pi = 6\,4\,1\,3\,5\,7\,9\,10\,8\,2 \in W_{10}$ in the wiggly flip graph. The neighbors are drawn above or below to match the Hasse diagram. There are five jump edges (including one swap edge) where there is one larger value, and two hop edges where there are two larger values; the other two edges are guaranteed not be used in our Hamilton path (see Theorem 2). The edges are drawn into π to highlight their associated ascent or descent. However, they are labeled by the minimum larger value (see Definition 6) for use with Algorithm F. The downward and upward partners are colored when there is more than one smaller and larger value, respectively.

More generally, if $\pi_{j-1}\pi_j$ is an ascent and π_j begins an upward order partnership, then the associated edge will move that entire run to the left; otherwise, only π_j moves left. Similarly, if π_{j-1} ends a downward order partnership, then the associated edge moves the entire run to the right; otherwise, only π_{j-1} moves right. This is formalized for the edges associated with ascents in Definition 5; edges associated with descents are defined in reverse.

▶ **Definition 5.** *Let $\pi = \pi_1\pi_2\cdots\pi_n \in W_n$ be a wiggly permutation with $u = \pi_{j-1} < \pi_j = v$ being one of its ascents*[3] *and let $\pi^{-1} = q_1q_2\cdots q_n$ be π's inverse (i.e., $q_x = y \iff \pi_y = x$). Define two runs $\alpha = \pi_i\pi_{i+1}\cdots\pi_{j-1}$ and $\beta = \pi_j\pi_{j+1}\cdots\pi_k$ where*

- *$i = q_{u+1}$ if u is odd and $q_{u+1} < q_u$; otherwise, $i = j - 1$.*
- *$k = q_{v+1}$ if v is odd and $q_{v+1} > q_v$; otherwise, $k = j$.*

The wiggly (left) flip *associated with this ascent moves β to the left over α. That is, $\pi_1\pi_2\cdots\pi_{i-1}\,\underline{\alpha}\,\overline{\beta}\,\pi_{k+1}\pi_{k+2}\cdots\pi_n = \pi_1\pi_2\cdots\pi_{i-1}\beta\alpha\pi_{k+1}\pi_{k+2}\cdots\pi_n$. If $|\beta| = 1$ (i.e., $j = k$), then the flip is a* wiggly jump. *If $|\beta| = 2$ (i.e., $j = k - 1$), then the flip is a* wiggly hop.

Note that wiggly flips are always minimal in the sense that β moves the shortest possible distance to create a new wiggly permutation. In particular, wiggly jumps are identical to the notion of minimal jumps in the *Permutation Languages* series [16].

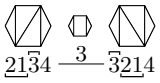
3.4 Motivation: Why Wiggly Permutations and Wiggly Flips?

Wiggly permutations are a new combinatorial object. However, Lemma 4 is reminiscent of pairings in certain Catalan objects like well-formed parentheses [50]. This similarity arises in the context of triangulations. Triangulations of a convex polygon are counted by the Catalan numbers [50]. Similarly, wiggly numbers $|W_{2n}|$ count wiggly triangulations [1]. See Figure 8. Many other connections involving wiggly permutations are discussed in [1].

Historically, Gray code research has focused on array operations like swaps and transpositions [34]. This is in large part due to the use of arrays in efficient generation algorithms [42]. For example, the first investigation into efficiently generating pattern avoiding permutation Gray codes allowed a small number of permutation entries to change (i.e., four for $\mathsf{Av}_n(231)$) [9].

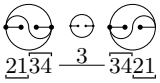
[3] Note that Lemma 31 in [1] instead indexes ascents as $\pi_j < \pi_{j+1}$.

(a) 231-avoiding permutations and their corresponding triangulations.



(b) A jump edge.

(c) Wiggly permutations and their corresponding wiggly triangulations.



(d) A hop edge.

Figure 8 Permutation languages representing other combinatorial objects. The flips in (b) and (d) correspond to removing then adding a (wiggly) diagonal in the corresponding (wiggly) triangulation.

Our paper is the first such investigation into wiggly permutations, so it is natural to ask why we consider wiggly flips (which can change *every* entry). One response is that wiggly flips arise in two natural graphs (see Figure 2c and 6c). Another is that wiggly flips flip diagonals in wiggly triangulations (see Figure 8d). However, the true answer is much deeper.

The *Permutation Languages* series [16] broadened interest in Gray code operations that are computationally motivated to those that are mathematically motivated. One pleasant consequence has been natural Gray codes for related combinatorial objects. Given two combinatorial objects and a bijection between them, a *simultaneous Gray code* is a single order that is a Gray code for both. To make this concrete, consider balanced parenthesis strings of length $2n$ and binary trees with n nodes [50]. Making a constant-size change in the string (e.g., transposing a pair of symbols) can cause a non-constant change (e.g., $\omega(1)$ pointer changes) in the corresponding trees. Thus, Gray codes for the former [44, 4, 56] and the latter [28] were not simultaneous. A Gray code known as cool-lex order [46] was shown to be simultaneous [45, 10, 26], however, it doesn't apply as naturally to other Catalan objects [8]. In contrast, the orders from the *Permutation Languages* series appear almost universal in their application. For example, Figure 5 in [16] shows that Algorithm J's Gray code for $\mathsf{Av}_n(231)$ is a simultaneous Gray code for binary trees, triangulations, and Dyck paths.

Wiggly permutations are also an ideal challenge for extending the *Permutation Languages* series. It was observed that wiggly permutations do not form a *zig-zag language* [1]. This means that wiggly permutations are not closed under inserting a largest value on the left and the right. In other words, if $\pi \in W_{n-1}$, then it is not always true that (i) $n\pi \in W_n$ and (ii) $\pi n \in W_n$. Thus, the results of [16] do not apply to W_n. But (i) is true for even n, and both (i) and (ii) are true for odd n. Indeed, the $n = 5$ column of Table 2 uses the same local recursion as Algorithm J for zig-zag languages. Furthermore, the $n = 6$ columns illustrate a similar local recursion except that the value of 5 acts as a right delimiter or sign-post for 6. We hope that our results will spur a new generalization of the *Permutations Languages* series.

Finally, our Gray code might be very efficient in terms of array-based generation. It only uses wiggly jumps and wiggly hops, so its efficiency depends only on the expected lengths of these operations. Theorem 9 provides a worst-case analysis of our efficient algorithm; further analysis may reveal that it runs in *constant amortized time* (i.e., amortized $\mathcal{O}(1)$-time delay). In this way, wiggly flips may mirror prefix-shifts [59]: the operation is inefficient in general, but Gray codes may use them efficiently by restricting the average length [57], biasing or restricting the lengths [7, 52, 48], or using alternate data structures [27].

4 Wiggly Changes

Now we describe *wiggly changes*. The Gray code was discovered greedily, and we start by discussing our experiments with Algorithm F. Then we define the local recursion that arose from a successful experiment. Finally, we prove that the order only uses jumps and hops.

4.1 Choosing Edge Labels

The wiggly flip graph consists of wiggly permutations under wiggly flips. To apply Algorithm F from Section 2 we must choose an initial node (e.g., $12\cdots n \in W_n$) and provide an edge labeling. Wiggly flips correspond to indices (i.e., one edge per ascent or descent $\pi_{j-1}\pi_j$) and the index j would provide natural edge labels. Indeed, this choice works for $n \leq 4$

$$12\underline{3}\overline{4}, 1\underline{2}\overline{4}3, 14\underline{2}\overline{3}, 1\overline{4}\underline{3}2, 1\overline{3}\overline{4}\underline{2}, 34\underline{1}\overline{2}, \overline{3}\overline{4}21, 21\underline{3}\overline{4}, \underline{2}1\overline{4}3, 42\underline{1}\overline{3}, 43\overline{2}\underline{1}, 4\overline{3}\underline{1}2, 41\overline{3}\underline{2}, 4123 \tag{3}$$

and (3) differs from Table 2. However, it stops working for larger n with only $\frac{32}{51}$ and $\frac{115}{176}$ objects generated for $n = 5$ and $n = 6$. This edge labeling also fails on the permutahedron.

Plain changes was previously described greedily as "swap the largest value" while Algorithm J "jumps the largest value". This is equivalent to using Algorithm F with the larger relocated value as the edge label. We cannot use this exact labeling as wiggly flips can move more than one larger value. However, a simple generalization is the *minimum larger value*.

▶ **Definition 6.** *If $\pi, \sigma \in S_n$ and $\mathsf{inv}(\pi) \subsetneq \mathsf{inv}(\sigma)$, then the* minimum larger label *of the edge (π, σ) is the minimum value $v \in [n]$ such that there exists $u \in [n]$ with $(u, v) \in \mathsf{inv}(\sigma) \smallsetminus \mathsf{inv}(\pi)$.*

Definition 6 is illustrated in Figure 7. We ran Algorithm F with these labels and observed that it worked! That is, every wiggly permutation is generated starting from $12\cdots n$. Equivalently, it creates a Hamilton path in the wigglyhedron starting from $12\cdots n$. By the above discussion, the same edge labeling allows Algorithm F to generate the previously discovered Hamilton paths in the permutahedron and associahedron. See Figure 9.

4.2 Zig-Zag Local Recursion

Given a permutation π of $[n]$, we let the parent $\mathsf{p}(\pi)$ be the result of removing n from π to create a permutation of $[n-1]$. In the opposite direction, given a permutation π of $[n-1]$ and an index $i \in [n]$, we let the child $\mathsf{c}_i(\pi)$ be the result of inserting n at position i to create a permutation of $[n-1]$. For example, $\mathsf{c}_3(615243) = 6175243$ and $\mathsf{p}(6175243) = 615243$.

Now we consider the family or hierarchy of wiggly permutation languages $W_0, W_1, W_2, \ldots$. It is easy to see that this family has the following properties.

1. If $\pi \in W_n$ and $n \geq 1$, then $\mathsf{p}(\pi) \in W_{n-1}$.
2. If $\pi \in W_{n-1}$, then $\mathsf{c}_1(\pi) = n \cdot \pi \in W_n$.

In other words, a new wiggly permutation is created by removing the largest value or by inserting a new largest value on the left. Given a family with property 1, we can construct an ordering of each language using a *zig-zag local recursion*. Towards this goal we recall $\overleftarrow{c}(\pi)$ ('zig') and $\overrightarrow{c}(\pi)$ ('zag') from [16]. If π is a permutation of $[n-1]$, then $\overleftarrow{c}(\pi)$ is the list $\mathsf{c}_1(\pi), \mathsf{c}_2(\pi), \ldots, \mathsf{c}_n(\pi)$ except that every permutation that is not in the associated language is skipped over. Similarly, $\overrightarrow{c}(\pi)$ is $\mathsf{c}_n(\pi), \mathsf{c}_{n-1}(\pi), \ldots, \mathsf{c}_1(\pi)$ with invalid entries skipped. Zig-zag local recursion expands each permutation of $[n-1]$ alternately using zigs and zags.

▶ **Definition 7.** *The* wiggly changes *order of W_n uses zig-zag local recursion:* $\mathsf{wiggly}(1) = 1$ *and if* $\mathsf{wiggly}(n-1) = \pi_1, \pi_2, \pi_3, \pi_4, \ldots$ *then* $\mathsf{wiggly}(n) = \overleftarrow{c}(\pi_1), \overrightarrow{c}(\pi_2), \overleftarrow{c}(\pi_3), \overrightarrow{c}(\pi_4), \ldots$.

Table 2 Wiggly changes for $n = 4, 5, 6$ and the edge labels that are greedily selected by Algorithm F to create the next wiggly permutation. Note that the local recursion from $n = 4$ to $n = 5$ is identical to Algorithm J. That is, the largest value 5 jumps into every valid position from the rightmost to leftmost, then vice versa, through successive permutations of $n = 4$. The local recursion from $n = 5$ to $n = 6$ is similar, except that the rightmost allowable position of the largest value 6 is constrained by the position of its partner 5. (This does not interfere with local recursion because if 5 has a smaller symbol to its left and to its right, then the next transition in the $n = 5$ order is a jump of 5. In other words, the larger symbols don't get in the way of transitions made in the $n = 4$ order.) Finally, notice that some jumps of 5 in the $n = 5$ order become hops of 56 in the $n = 6$ order.

1234 4	12345 5	123456 6	641325 5	346152 6	213564 6
1243 4	12354 5	123465 6	641352 6	341652 6	213654 6
1423 4	12534 5	123645 6	413652 6	341562 5	216354 6
4123 3	15234 5	126345 6	413562 5	341256 6	621354 5
4132 4	51234 4	162345 6	564132 6	341265 6	621345 6
1432 4	51243 5	612345 5	654132 4	341625 6	216345 6
1342 3	15243 5	612354 6	651432 6	346125 6	213645 6
3412 4	12543 5	162354 6	561432 5	364125 6	213465 6
4312 2	12435 4	126354 6	156432 6	634125 4	213456 4
4321 4	14235 5	123654 6	165432 6	643125 6	214356 6
3421 3	15423 5	123564 5	615432 5	436125 6	214365 6
2134 4	51423 4	125634 6	614352 6	431625 6	216435 6
2143 4	54123 5	126534 6	164352 6	431265 6	621435 5
4213	41235 3	162534 6	143652 6	431256 5	621543 6
	41325 5	612534 5	143562 5	431562 6	216543 6
	41352 5	615234 6	143256 6	431652 6	215643 5
	54132 4	165234 6	143265 6	436152 6	562143 6
	51432 5	156234 5	143625 6	643152 5	652143 4
	15432 5	561234 6	164325 6	643512 6	654213 6
	14352 5	651234 4	614325 4	436512 6	564213 5
	14325 4	651243 6	613425 6	435612 5	421356 6
	13425 5	561243 5	163425 6	564312 6	421365 6
	13452 5	156243 6	136425 6	654312 2	642135
	13542 5	165243 6	134625 6	654321 6	
	15342 5	615243 5	134265 6	564321 5	
	51342 3	612543 6	134256 5	435621 6	
	53412 5	162543 6	134562 6	436521 6	
	35412 5	126543 6	134652 6	643521 5	
	34512 5	125643 5	136452 6	643215 6	
	34152 5	124356 6	163452 6	436215 6	
	34125 4	124365 6	613452 5	432165 6	
	43125 5	126435 6	613542 6	432156 4	
	43152 5	162435 6	163542 6	342156 6	
	43512 5	612435 4	136542 6	342165 6	
	54312 2	614235 6	135642 5	346215 6	
	54321 5	164235 6	156342 6	364215 6	
	43521 5	142365 6	165342 6	634215 5	
	43215 4	142356 5	615342 5	634521 6	
	34215 5	156423 6	651342 6	364521 6	
	34521 5	165423 6	561342 3	346521 6	
	35421 5	615423 5	563412 6	345621 5	
	53421 3	651423 6	653412 5	356421 6	
	52134 5	561423 4	635412 6	365421 6	
	21534 5	564123 6	365412 6	635421 5	
	21354 5	654123 5	356412 5	653421 6	
	21345 4	641235 6	345612 6	563421 3	
	21435 5	412365 6	346512 6	562134 6	
	21543 5	412356 3	364512 6	652134 5	
	52143 4	413256 6	634512 5	621534 6	
	54213 5	413265 6	634152 6	216534 6	
	42135	413625 6	364152 6	215634 5	

For example, the order of wiggly permutations is extended from $n = 3$ to $n = 4$ as follows.

$$\begin{aligned}\mathsf{wiggly}(3) &= 123, 132, 312, 321, 213\\ \mathsf{wiggly}(4) &= \overleftarrow{c}\,(123), \overrightarrow{c}\,(132), \overleftarrow{c}\,(312), \overrightarrow{c}\,(321), \overleftarrow{c}\,(213)\\ &= \underbrace{1234, 1243, 1423, 4123}_{\overleftarrow{c}\,(123)}, \underbrace{4132, 1432, 1342}_{\overrightarrow{c}\,(132)}, \underbrace{3412, 4312}_{\overleftarrow{c}\,(312)}, \underbrace{4321, 3421}_{\overrightarrow{c}\,(321)}, \underbrace{2134, 2143, 4213}_{\overleftarrow{c}\,(213)}\end{aligned}$$

Note that $\overrightarrow{c}\,(132)$ ends with 1342 (and not 1324 as in a zig-zag language) while $\overleftarrow{c}\,(312)$ begins with 3412. So the transition between these sublists is the hop $\underline{1}\overline{34}2$ to $\overline{34}\underline{1}2$. The next lemma describes the beginning and end of each zig and zag and follows from Definition 3.

▶ **Lemma 8.** *Let $\pi = \pi_1\pi_2\cdots\pi_{n-1} \in \mathsf{wiggly}(n-1)$ with $\pi_m = n-1$. If n is odd, then*

$$\overrightarrow{c}\,(\pi) = \mathsf{c}_1(\pi), \ldots, \mathsf{c}_n(\pi) \text{ and } \overleftarrow{c}\,(\pi) = \mathsf{c}_n(\pi), \ldots, \mathsf{c}_1(\pi). \tag{4}$$

In other words, these zigs and zags result in the largest symbol moving from the first position to the last position (or vice versa) as in a zig-zag language. However, if n is even, then

$$\overrightarrow{c}\,(\pi) = \mathsf{c}_1(\pi), \ldots, \mathsf{c}_{m+1}(\pi) \text{ and } \overleftarrow{c}\,(\pi) = \mathsf{c}_{m+1}(\pi), \ldots, \mathsf{c}_1(\pi). \tag{5}$$

In other words, these zigs and zigs result in the largest symbol moving from the first position to the position immediately to the right of the next largest symbol (or vice versa).

The flip graph of wiggly permutations and wiggly flips is $\mathbb{W}_n$. We now prove Theorem 2.

Proof. We prove the result by induction on $n \geq 1$ with $\mathsf{wiggly}(1) = 1$ as a base case.

Assume that $\mathsf{wiggly}(n-1) = \pi_1, \pi_2, \ldots$ is a Hamilton path in $\mathbb{W}_{n-1}$ and consider

$$\mathsf{wiggly}(n) = \overleftarrow{c}\,(\pi_1), \overrightarrow{c}\,(\pi_2), \overleftarrow{c}\,(\pi_3), \overrightarrow{c}\,(\pi_4), \ldots.$$

It is clear that $\mathsf{wiggly}(n)$ is an ordering of W_n. The transitions within each $\overleftarrow{c}$ and $\overrightarrow{c}$ sublist are jumps. So we need only consider the transitions between consecutive sublists. Let $\sigma = \pi_i$ and $\tau = \pi_{i+1}$ so that we can index their entries using subscripts.

We first consider transitions between sublists of the form $\overleftarrow{c}\,(\sigma)$ and $\overrightarrow{c}\,(\tau)$. By Lemma 8,

$$\overleftarrow{c}\,(\sigma),\ \overrightarrow{c}\,(\tau) = \ldots, \mathsf{c}_1(\sigma),\ \mathsf{c}_1(\tau), \ldots = \ldots, n\sigma,\ n\tau, \ldots$$

By induction, (σ, τ) is a jump/hop edge of $\mathbb{W}_{n-1}$. Thus, $(n\sigma, n\tau)$ is a jump/hop edge of $\mathbb{W}_n$.

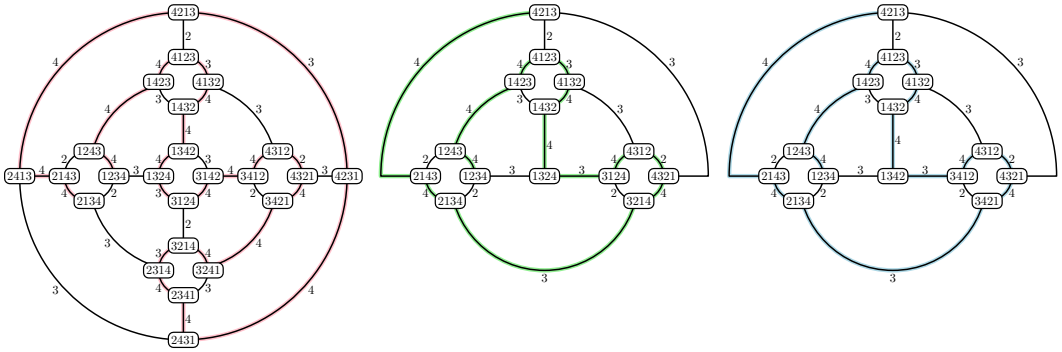
Next consider transitions between sublists of the form $\overrightarrow{c}\,(\sigma)$ and $\overleftarrow{c}\,(\tau)$. We consider two cases depending on the parity of n. If n is odd, then by Lemma 8,

$$\overrightarrow{c}\,(\sigma),\ \overleftarrow{c}\,(\tau) = \ldots, \mathsf{c}_n(\sigma),\ \mathsf{c}_n(\tau), \ldots = \ldots, \sigma n,\ \tau n, \ldots$$

By induction, (σ, τ) is a jump/hop edge of $\mathbb{W}_{n-1}$. Thus, $(\sigma n, \tau n)$ is a jump/hop edge of $\mathbb{W}_n$. If n is even, then let $\sigma_m = n-1$ and $\tau_{m'} = n-1$. In other words, the largest symbol $n-1$ is at index m in $\sigma \in W_{n-1}$, and at index m' in $\tau \in W_{n-1}$. Thus, by Lemma 8,

$$\overrightarrow{c}\,(\sigma),\ \overleftarrow{c}\,(\tau) = \ldots, \mathsf{c}_{m+1}(\sigma),\ \mathsf{c}_{m'+1}(\tau), \ldots.$$

By induction, (σ, τ) is a jump/hop edge of $\mathbb{W}_{n-1}$. Since n is even, we know that $n-1$ is odd. Thus, if $n-1$ moves between σ and τ, then the change between them is a jump of $n-1$. Therefore, the change between $\mathsf{c}_m(\sigma)$ and $\mathsf{c}_{m'}(\tau)$ is a hop of $n-1$. If $n-1$ does not move between σ and τ, then it must be that $m = n-1$ and $m' = n-1$ since the unpartnered odd value $n-1$ can always be inserted into the rightmost position. Therefore, the change between $\mathsf{c}_n(\sigma)$ and $\mathsf{c}_n(\tau)$ in $\mathbb{W}_n$ is the same as the change between σ and τ in $\mathbb{W}_{n-1}$. ◀



(a) S_4 with swaps. **(b)** $\mathsf{Av}_4(231)$ with minimal jumps. **(c)** W_4 with wiggly flips.

Figure 9 Flip graphs including minimum larger value edge labels (Definition 6). The graphs with unlabeled edges and the highlighted Hamilton paths are isomorphic to those in Figures 2 and 6. The paths are generated greedily by Algorithm F. These embeddings show that (b) and (c) are isomorphic (and the edge labels on the Hamilton paths are identical) but this is not true for $n > 4$.

5 Efficient Generation

Now we efficiently generate wiggly changes. Our algorithm differs from the greedy construction as it is *history-free* (i.e., it doesn't remember previous objects). It also iterative, so it differs from a direct translation of the zig-zag local recursion. As is customary in generation algorithms, we store only one object (i.e., one wiggly permutation) and `visit` (i.e., *yield*) it every time it is modified to be the next object in the order. Each successive wiggly permutation is generated in worst-case $\mathcal{O}(n)$-time. Appendix A has a complete Python implementation.

We begin by reviewing a well-known *loopless algorithm* for generating plain changes. This means that each successive permutation is visited with worst-case $\mathcal{O}(1)$-time delay. The approach was devised by Ehrlich [11] and a modern treatment appears in Knuth's Algorithm P (*Plain changes*) and Algorithm H (*Loopless reflected mixed-radix Gray generation*) [20].

5.1 Generating Plain Changes

The main idea is to generate the *inversion word* that encodes each permutation. Inversion words are mixed-radix words with bases $1, 2, \ldots, n$ of which there are $n!$. The ith digit of such a word encodes the number of values that are smaller than i and which are inverted with it in its corresponding permutation. The words are generated in standard *reflected Gray code order* (i.e., each digit increments from 0 to its maximum, then decrements from its maximum to 0). Each swap in a permutation changes its inversion word by ± 1 in one digit and vice versa. Pleasingly, this Gray code of inversion words corresponds to plain changes.

To generate the inversion words looplessly we store a direction for each digit, or equivalently, a direction in which each value is moving in the permutation. *Focus pointers* determine which digit changes (i.e., which value is swapped) in constant time; their name is somewhat misleading as they are an array of integers. The inverse of the current permutation is maintained to determine the location of the value to move in worst-case $\mathcal{O}(1)$-time. Our presentation of this classic algorithm appears in Algorithm 1 and it is organized to highlight the similarities with our wiggly changes algorithm. In particular, we use `swapLeft` and `swapRight` functions to update the permutation, its inverse, and its inversion word. The basic approach has previously been modified to generate other permutation Gray codes [17, 13] and Gray codes for other types of permutations [39].

Algorithm 1 Plain changes generated in worst-case $\mathcal{O}(1)$-time delay. Each array has length n (with 1-based indexing) and is traced at the **while** (line 8) for $\mathtt{n} = 4$ in the table below. The operations are swaps $\cdots \underline{\pi_{i-1}\pi_i} \cdots$ (left) or $\cdots \underline{\pi_i\pi_{i+1}} \cdots$ (right) with edge label $\mathtt{v}$ as per Definition 6. The table highlights edge labels $\mathtt{v} = \mathtt{fs}[4]$; see Algorithm 2's table for more information on each array.

```
function swapLeft(perm, v, inv, word)
1: i ← inv[v]                          ▷ index of edge label v
2: u ← perm[i − 1]                     ▷ smaller value to swap
3: word[v] ← word[v]+1                 ▷ new inversion (u, v)
4: perm[i] ← u                         ▷ swap u right
5: perm[i − 1] ← v                     ▷ swap v left
6: inv[u] ← i                          ▷ update u's index
7: inv[v] ← i − 1                      ▷ update v's index
```

```
function swapRight(perm, v, inv, word)
1: i ← inv[v]                          ▷ index of edge label v
2: u ← perm[i + 1]                     ▷ smaller value
3: word[v] ← word[v]−1                 ▷ old (u, v)
4: perm[i] ← u                         ▷ swap u left
5: perm[i + 1] ← v                     ▷ swap v right
6: inv[u] ← i                          ▷ update u's index
7: inv[v] ← i + 1                      ▷ update v's index
```

```
function plain(n)
1: perm ← [1, 2, ..., n]               ▷ current permutation
2: inv ← [1, 2, ..., n]                ▷ inverse permutation
3: word ← [0, 0, ..., 0]               ▷ inversion word
4: fs ← [1, 2, ..., n]                 ▷ focus pointers
5: dirs ← [−1, −1, ..., −1]            ▷ value directions
6: visit perm                          ▷ first permutation
7: v ← fs[n]                           ▷ next edge label (swap value)
8: while v > 1 do                      ▷ no edges labeled 1
9:    if dirs[v] = −1                  ▷ leftward swap?
10:      swapLeft(perm, v, inv, word)
11:   else                             ▷ rightward swap
12:      swapRight(perm, v, inv, word)
13:   if word[v] = 0 or word[v] = v − 1   ▷ limit?
14:      dirs[v] ← −dirs[v]            ▷ change direction
15:      fs[v] ← fs[v − 1]             ▷ inherit focus pointer
16:      fs[v − 1] ← v − 1             ▷ reset focus pointer
17:   visit perm                       ▷ current permutation
18:   v ← fs[n]                        ▷ next edge label (swap value)
19:   fs[n] ← n                        ▷ reset focus pointer
```

perm	inv	word	fs	dirs
1234	1234	0000	1234	−−−−
1243	1243	0001	1234	−−−−
1423	1342	0002	1234	−−−−
4123	2341	0003	1233	−−−+
4132	2431	0013	1234	−−−+
1432	1432	0012	1234	−−−+
1342	1423	0011	1234	−−−+
1324	1324	0010	1233	−−−−
3124	2314	0020	1224	−−+−
3142	2413	0021	1224	−−+−
3412	3412	0022	1224	−−+−
4313	3421	0023	1232	−−++
4321	4321	0123	1134	−+++
3421	4312	0122	1134	−+++
3241	4213	0121	1134	−+++
3214	3214	0120	1133	−+−−
2314	3124	0110	1134	−++−
2341	4123	0111	1134	−++−
2431	4132	0112	1134	−++−
4231	4231	0113	1133	−+++
4213	3241	0103	1214	−+−+
2413	3142	0102	1214	−+−+
2143	2143	0101	1214	−+−+
2134	2134	0100	1231	−+−−

5.2 Generating Wiggly Changes

Generating wiggly changes is similar to generating plain changes. The same concepts allow us to determine the value of the operation to apply (i.e., the next edge label to follow) in worst-case $\mathcal{O}(1)$-time. Moreover, we can update the resulting permutation's inverse and inversion word in worst-case $\mathcal{O}(1)$-time. Pseudocode appears in Algorithm 2 with a full Python implementation in Appendix A. Two specific challenges are highlighted below.

- The framework gives us the value and direction of the next operation. But we must determine if the operation is a jump or a hop. Similarly, we must determine how many smaller values are jumped or hopped over. Fortunately, the necessary information comes directly from whether certain partners are in their upward or downward order as per Definition 3 and Definitions 5–6.
- Each even value must change directions whenever it is swapped to be immediately to the right of its odd partner. This leads to an extra condition in line 17.

Algorithm 2 Wiggly permutations generated in worst-case $\mathcal{O}(n)$-time delay. Each array has length n (with 1-based indexing) and is traced for $\texttt{n} = 4$ below. The operations are wiggly jumps and hops $\cdots \underline{\pi_i \cdots \pi_{j-1}} \overline{\pi_j \cdots \pi_k} \cdots$ (left) or $\cdots \overline{\pi_i \cdots \pi_{j-1}} \underline{\pi_j \cdots \pi_k} \cdots$ (right) with edge label v as per Definitions 5–6.

```
function wiggleLeft(perm, v, inv, word, hop)
 1: h ← [[hop]]                          ▷ [[True/False]] is 1/0
 2: j ← inv[v]                           ▷ index of edge label v
 3:
 4: u ← perm[j−1]                        ▷ last smaller value
 5: i ← j − 1                            ▷ first smaller index?
 6: if u is odd and inv[u] ≥ inv[u+1]
 7:   i ← inv[u + 1]                     ▷ correct i to partner
 8: word[v] ← word[v]+j−i                ▷ +inversions
 9: if hop = True                        ▷ partner inversions?
10:   word[v+1] ← word[v+1] + j−i
11: while j > i do                       ▷ shift smaller values
12:   perm[j + h] ← perm[j − 1]          ▷ one or two
13:   inv[perm[j − 1]] ← j + h           ▷ positions
14:   j ← j − 1                          ▷ to the right
15: perm[i] ← v                          ▷ move larger value
16: inv[v] ← i                           ▷ update its inverse
17: if hop = True                        ▷ another larger value?
18:   perm[i+1] ← v+1                    ▷ move partner
19:   inv[v+1] ← i+1                     ▷ update its inverse
```

```
function wiggleRight(perm, v, inv, word, hop)
 1: h ← [[hop]]                          ▷ [[True/False]] is 1/0
 2: i ← inv[v]                           ▷ index of edge label v
 3: j ← i + 1 + h                        ▷ first smaller index
 4: u ← perm[j]                          ▷ first smaller value
 5: k ← j                                ▷ last smaller index?
 6: if u is even and inv[u] ≤ inv[u−1]
 7:   k ← inv[u − 1]                     ▷ correct k to partner
 8: word[v] ← word[v]−k+j−1              ▷ −inversions
 9: if hop = True                        ▷ partner inversions?
10:   word[v+1] ← word[v+1]−k+j−1
11: while i < k−h do                     ▷ shift smaller values
12:   perm[i] ← perm[i + 1 + h]          ▷ one or two
13:   inv[perm[i + 1 + h]] ← i           ▷ positions
14:   i ← i + 1                          ▷ to the left
15: perm[k − h] ← v                      ▷ move larger value
16: inv[v] ← k − h                       ▷ update its inverse
17: if hop = True                        ▷ another larger value?
18:   perm[k] ← v+1                      ▷ move partner
19:   inv[v+1] ← k                       ▷ update its inverse
```

```
function wiggly(n)
 1: perm ← [1, 2, . . . , n]             ▷ current permutation
 2: inv ← [1, 2, . . . , n]              ▷ inverse permutation
 3: word ← [0, 0, ..., 0]                ▷ inversion word
 4: fs ← [1, 2, . . . , n]               ▷ focus pointers
 5: dirs ← [−1, −1, . . . , −1]          ▷ value directions
 6: visit perm                           ▷ first wiggly permutation
 7: v ← fs[n]                            ▷ next edge label (wiggle value)
 8: while v > 1 do                       ▷ no edges labeled 1
 9:   i ← inv[v]                         ▷ index of v
10:   hop ← v is even and i < n and perm[i+1] = v+1
11:   if dirs[v] = −1                    ▷ leftward jump/hop?
12:     wiggleLeft(perm, v, inv, word, hop)
13:   else                               ▷ rightward jump/hop
14:     wiggleRight(perm, v, inv, word, hop)
15:   i ← inv[v]                         ▷ new index of v
16:   pair ← v is even and perm[i−1] = v−1)
17:   if word[v] = 0 or word[v] = v−1 or pair
18:     dirs[v] ← −dirs[v]               ▷ change direction
19:     fs[v] ← fs[v−1]                  ▷ inherit focus pointer
20:     fs[v−1] ← v−1                    ▷ reset focus pointer
21:   visit perm                         ▷ current wiggly permutation
22:   v ← fs[n]                          ▷ next edge label (wiggle value)
23:   fs[n] ← n                          ▷ reset focus pointer
```

perm	inv	word	fs	dirs
123$\overline{4}$	1234	0000	1234	−−−−
12$\overline{4}$3	1243	0001	1234	−−−−
1$\overline{4}$23	1342	0002	1234	−−−−
41$\overline{2}$3	2341	0003	1233	−−−+
$\overline{4}$132	2431	0013	1234	−−−+
1$\overline{4}$32	1432	0012	1234	−−−+
1$\overline{3}\overline{4}$2	1423	0011	1233	−−−−
3$\overline{4}$12	3412	0022	1224	−−+−
431$\overline{2}$	3421	0023	1232	−−++
$\overline{4}\overline{3}$21	4321	0123	1134	−+++
$\overline{3}\overline{4}$21	4312	0122	1133	−++−
21$\overline{3}\overline{4}$	2134	0100	1214	−+−−
2$\overline{1}\overline{4}$3	2143	0101	1214	−+−−
4213	3241	0103	1231	−+−+

Variable trace at **while** (line 8) in wiggly changes wiggly(4). The edge label of next jump/hop is v = fs[4] in direction dirs[v]. Indexing and direction changes in $\mathcal{O}(1)$-time via inv and word.

The algorithm is loopless except for the time taken to apply each operation. More specifically, each jump and hop can take up to $\Theta(n)$-time because a linear number of symbols may need to be relocated in the array. This leads to the following theorem. In future work we wish to further analyze our Gray code to provide a tighter average or amortized run-time.

▶ **Theorem 9.** *Algorithm 2 generates the wiggly changes Gray code of wiggly permutations of length n with worst-case $\mathcal{O}(n)$-time delay using $\mathcal{O}(n)$ memory.*

References

1 Asilata Bapat and Vincent Pilaud. Wigglyhedra. *Mathematische Zeitschrift*, 310(3):1–41, 2025.

2 David Bevan. Permutation patterns: basic definitions and notation. *arXiv preprint*, 2015. arXiv:1506.06673.

3 Samuel Buick, Goertz Madeleine, Amos Lastmann, Kunal Pal, Helen Qian, Sam Tacheny, Aaron Williams, Leah Williams, and Yulin Zhai. Exhaustive generation of pattern-avoiding s-words. In *Proceedings of the 23rd International Conference on Permutation Patterns*, 2025.

4 Bette Bultena and Frank Ruskey. An Eades-McKay algorithm for well-formed parentheses strings. *Information Processing Letters*, 68(5):255–259, 1998. doi:10.1016/S0020-0190(98)00171-9.

5 Ben Cameron, Joe Sawada, Wei Therese, and Aaron Williams. Hamiltonicity of k-sided pancake networks with fixed-spin: Efficient generation, ranking, and optimality. *Algorithmica*, 85(3):717–744, 2023. doi:10.1007/S00453-022-01022-X.

6 Cesar Ceballos and Viviane Pons. The s-weak order and s-permutahedra I: Combinatorics and lattice structure. *SIAM Journal on Discrete Mathematics*, 38(4):2855–2895, 2024.

7 Peter F Corbett. Rotator graphs: An efficient topology for point-to-point multiprocessor networks. *IEEE Transactions on Parallel & Distributed Systems*, 3(05):622–626, 1992. doi:10.1109/71.159045.

8 Emily Downing, Stephanie Einstein, Elizabeth Hartung, and Aaron Williams. Catalan squares and staircases: Relayering and repositioning Gray codes. In *Proceedings of the 35th Canadian Conference on Computational Geometry, CCCG*, pages 273–281, 2023.

9 WMB Dukes, Mark F Flanagan, Toufik Mansour, and Vincent Vajnovszki. Combinatorial Gray codes for classes of pattern avoiding permutations. *Theoretical Computer Science*, 396(1-3):35–49, 2008. doi:10.1016/J.TCS.2007.12.002.

10 Stephane Durocher, Pak Ching Li, Debajyoti Mondal, Frank Ruskey, and Aaron Williams. Cool-lex order and k-ary Catalan structures. *Journal of Discrete Algorithms*, 16:287–307, 2012. doi:10.1016/J.JDA.2012.04.015.

11 Gideon Ehrlich. Loopless algorithms for generating permutations, combinations, and other combinatorial configurations. *Journal of the ACM (JACM)*, 20(3):500–513, 1973. doi:10.1145/321765.321781.

12 Joseph Gallian. *Contemporary abstract algebra*. Chapman and Hall/CRC, 2021.

13 Pramod Ganapathi and Rezaul Chowdhury. A unified framework to discover permutation generation algorithms. *The Computer Journal*, 66(3):603–614, 2023. doi:10.1093/COMJNL/BXAB181.

14 Frank Gray. Pulse code communication. *United States Patent Number 2632058*, 1953.

15 Petr Gregor, Torsten Mütze, and Namrata. Combinatorial generation via permutation languages. VI. Binary trees. *European Journal of Combinatorics*, 122:104020, 2024. doi:10.1016/J.EJC.2024.104020.

16 Elizabeth Hartung, Hung P Hoang, Torsten Mütze, and Aaron Williams. Combinatorial generation via permutation languages. I. Fundamentals. *Transactions of the American Mathematical Society*, 375:2255–2291, 2022.

17 Alexander E Holroyd, Frank Ruskey, and Aaron Williams. Shorthand universal cycles for permutations. *Algorithmica*, 64:215–245, 2012. doi:10.1007/S00453-011-9544-Z.

18 OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences. https://oeis.org, 2025.

19 Selmer M Johnson. Generation of permutations by adjacent transposition. *Mathematics of computation*, 17(83):282–285, 1963.

20 Donald E Knuth. *The Art of Computer Programming, Volume 4A, Fascicle 2: Generating All Tuples and Permutations*. Addison-Wesley, 2005.

21 Donald E Knuth. *Art of Computer Programming, Volume 4, Fascicle 4, The: Generating All Trees–History of Combinatorial Generation*. Addison-Wesley Professional, 2013.

22 James Korsh and Seymour Lipschutz. Generating multiset permutations in constant time. *Journal of Algorithms*, 25(2):321–335, 1997. doi:10.1006/JAGM.1997.0889.

23 James F Korsh and Paul S LaFollette. Loopless array generation of multiset permutations. *The Computer Journal*, 47(5):612–621, 2004. doi:10.1093/COMJNL/47.5.612.

24 Donald L Kreher and Douglas R Stinson. Combinatorial algorithms: generation, enumeration, and search. *ACM SIGACT News*, 30(1):33–35, 1999. doi:10.1145/309739.309744.

25 Carsten Lange and Vincent Pilaud. Associahedra via spines. *Combinatorica*, 38(2):443–486, 2018. doi:10.1007/S00493-015-3248-Y.

26 Paul Lapey and Aaron Williams. Pop & push: Ordered tree iteration in O(1)-time. In *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*, pages 53–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ISAAC.2022.53.

27 Zsuzsanna Lipták, Francesco Masillo, Gonzalo Navarro, and Aaron Williams. Constant time and space updates for the sigma-tau problem. In *International Symposium on String Processing and Information Retrieval*, pages 323–330. Springer, 2023. doi:10.1007/978-3-031-43980-3_26.

28 Joan M Lucas, D. Roelants van Baronaigien, and Frank Ruskey. On rotations and the generation of binary trees. *Journal of Algorithms*, 15(3):343–366, 1993. doi:10.1006/JAGM.1993.1045.

29 Percy A MacMahon. *Combinatory analysis, volumes I and II*, volume 137. American Mathematical Society, 2001.

30 Arturo Merino and Torsten Mütze. Combinatorial generation via permutation languages. III. Rectangulations. *Discrete & Computational Geometry*, 70(1):51–122, 2023. doi:10.1007/S00454-022-00393-W.

31 Arturo Merino, Torsten Mütze, and Aaron Williams. All your bases are belong to us: Listing all bases of a matroid by greedy exchanges. In *11th International Conference on Fun with Algorithms (FUN 2022)*, pages 22–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.FUN.2022.22.

32 Arturo Ignacio Merino Figueroa. *Combinatorial generation: greedy approaches and symmetry*. PhD thesis, Technische Universität Berlin, 2023.

33 Torsten Mütze. Proof of the middle levels conjecture. *Proceedings of the London Mathematical Society*, 112(4):677–713, 2016.

34 Torsten Mütze. Combinatorial Gray codes – An updated survey. *The Electronic Journal of Combinatorics*, page 93 pages, 2023.

35 Torsten Mütze. A book proof of the middle levels theorem. *Combinatorica*, 44(1):205–208, 2024. doi:10.1007/S00493-023-00070-3.

36 Arnau Padrol, Yann Palu, Vincent Pilaud, and Pierre-Guy Plamondon. Associahedra for finite-type cluster algebras and minimal relations between g-vectors. *Proceedings of the London Mathematical Society*, 127(3):513–588, 2023.

37 Vincent Pilaud, Francisco Santos, and Günter M Ziegler. Celebrating loday's associahedron. *Archiv der Mathematik*, 121(5):559–601, 2023.

38 Alexander Postnikov. Permutohedra, associahedra, and beyond. *International Mathematics Research Notices*, 2009(6):1026–1106, 2009.

39 Yuan Qiu and Aaron Williams. Generating signed permutations by twisting two-sided ribbons. In *Latin American Symposium on Theoretical Informatics*, pages 114–129. Springer, 2024. doi:10.1007/978-3-031-55598-5_8.

40 Nathan Reading. Lattice congruences of the weak order. *Order*, 21(4):315–344, 2004. doi:10.1007/S11083-005-4803-8.

41 Nathan Reading. Finite Coxeter groups and the weak order. In *Lattice Theory: Special Topics and Applications: Volume 2*, pages 489–561. Springer, 2016.

42 Frank Ruskey. Combinatorial generation. *Working version (1j-CSC 425/520)*, 2003.

43 Frank Ruskey and TC Hu. Generating binary trees lexicographically. *SIAM Journal on Computing*, 6(4):745–758, 1977. doi:10.1137/0206055.

44 Frank Ruskey and Andrzej Proskurowski. Generating binary trees by transpositions. *Journal of algorithms*, 11(1):68–84, 1990. doi:10.1016/0196-6774(90)90030-I.

45 Frank Ruskey and Aaron Williams. Generating balanced parentheses and binary trees by prefix shifts. In *CATS*, volume 8, page 140, 2008.

46 Frank Ruskey and Aaron Williams. The coolest way to generate combinations. *Discrete Mathematics*, 309(17):5305–5320, 2009. doi:10.1016/J.DISC.2007.11.048.

47 Joe Sawada and Aaron Williams. Successor rules for flipping pancakes and burnt pancakes. *Theoretical Computer Science*, 609:60–75, 2016. doi:10.1016/J.TCS.2015.09.007.

48 Joe Sawada and Aaron Williams. Solving the sigma-tau problem. *ACM Transactions on Algorithms (TALG)*, 16(1):1–17, 2019. doi:10.1145/3359589.

49 Robert Sedgewick. Permutation generation methods. *ACM Computing Surveys (CSUR)*, 9(2):137–164, 1977. doi:10.1145/356689.356692.

50 Richard P Stanley. *Catalan numbers*. Cambridge University Press, 2015.

51 Hugo Steinhaus. *One hundred problems in elementary mathematics*. Courier Corp., 1979.

52 Brett Stevens and Aaron Williams. Hamilton cycles in restricted and incomplete rotator graphs. *Journal of Graph Algorithms and Applications*, 16(4):785–810, 2012. doi:10.7155/JGAA.00278.

53 Dov Tamari. Monoïdes préordonnés et chaînes de malcev. *Bulletin de la Société mathématique de France*, 82:53–96, 1954.

54 Hale F Trotter. Algorithm 115: perm. *Communications of the ACM*, 5(8):434–435, 1962. doi:10.1145/368637.368660.

55 William T Trotter. *Combinatorics and partially ordered sets*. Johns Hopkins Press, 1992.

56 Vincent Vajnovszki and Timothy Walsh. A loop-free two-close Gray-code algorithm for listing k-ary Dyck words. *Journal of Discrete Algorithms*, 4(4):633–648, 2006. doi:10.1016/J.JDA.2005.07.003.

57 Aaron Williams. Loopless generation of multiset permutations using a constant number of variables by prefix shifts. In *Proceedings of the twentieth annual ACM-SIAM symposium on discrete algorithms*, pages 987–996. SIAM, 2009. doi:10.1137/1.9781611973068.107.

58 Aaron Williams. The greedy Gray code algorithm. In *Workshop on Algorithms and Data Structures*, pages 525–536. Springer, 2013. doi:10.1007/978-3-642-40104-6_46.

59 Aaron Michael Williams. *Shift Gray codes*. PhD thesis, University of Victoria, 2009.

60 Shmuel Zaks and Dana Richards. Generating trees and other combinatorial objects lexicographically. *SIAM Journal on Computing*, 8(1):73–81, 1979. doi:10.1137/0208006.

A Python Implementation

A full Python implementation appears below. To run it in the terminal copy the top two columns into `wiggly.py`, then use `python3` with a value of `n` as shown. (Note that `pdf` files omit leading spaces, so indents must be restored to `wiggly.py` after copy and pasting.)

```
def wiggleLeft(perm,v,inv,word,hop):
  j = inv[v]
  u = perm[j-1]
  i = j-1
  if u%2 == 1 and inv[u] >= inv[u+1]:
    i = inv[u+1]
  word[v] += j-i
  if hop:
    word[v+1] += j-i
  while j > i:
    perm[j+hop] = perm[j-1]
    inv[perm[j-1]] = j+hop
    j -= 1
  perm[i] = v
  inv[v] = i
  if hop:
    perm[i+1] = v+1
    inv[v+1] = i+1

def wiggleRight(perm,v,inv,word,hop):
  i = inv[v]
  j = i + 1 + hop
  u = perm[j]
  k = j
  if u%2 == 0 and inv[u] <= inv[u-1]:
    k = inv[u-1]
  word[v] -= k-j+1
  if hop:
    word[v+1] -= k-j+1
  while i < k-hop:
    perm[i] = perm[i+1+hop]
    inv[perm[i+1+hop]] = i
    i += 1
  perm[k-hop] = v
  inv[v] = k-hop
  if hop:
    perm[k] = v+1
    inv[v+1] = k
```

```
def wiggly(n):
  perm = list(range(n+1))
  inv = list(range(n+1))
  word = [0] * (n+1)
  fs = list(range(n+1))
  dirs = [-1]*(n+1)
  yield perm
  v = fs[n]
  while v > 1:
    i = inv[v]
    hop = v%2==1 and i<n and perm[i+1]==v+1
    if dirs[v] == -1:
      wiggleLeft(perm, v, inv, word, hop)
    else:
      wiggleRight(perm, v, inv, word, hop)
    i = inv[v]
    pair = (v%2 == 0 and perm[i-1] == v-1)
    if word[v]==0 or word[v]==v-1 or pair:
      dirs[v] *= -1
      fs[v] = fs[v-1]
      fs[v-1] = v-1
    yield perm
    v = fs[n]
    fs[n] = n

if __name__ == "__main__":
  import sys  # Handle command-line arguments
  ok = len(sys.argv) == 2
  ok = sys.argv[1].isdigit() if ok else False
  ok = int(sys.argv[1]) > 0 if ok else False
  if not ok:
    print("usage: %s n" % sys.argv[0])
    exit(0)
  n = int(sys.argv[1])
  total = 0
  for perm in wiggly(n):
    print(*perm[1:], sep="")
    total += 1
  print("total: %d" % total)
```

```
~$ python3 wiggly.py 6
123456
123465
123645
...
642135
total: 176
```

B-Treaps Revised: Write Efficient Randomized Block Search Trees with High Load

Roodabeh Safavi
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Martin P. Seybold
Faculty of Computer Science, Theory and Applications of Algorithms, University of Vienna, Austria

Abstract

Uniquely represented (UR) data structures represent each logical state with a unique storage state. We study the problem of maintaining a dynamic set of n keys from a totally ordered universe in this context. UR structures are also called "strongly history independent" structures in the literature.

We introduce a two-layer data structure called (α, ε)-Randomized Block Search Tree (RBST) that is uniquely represented and suitable for external memory (EM). Though RBSTs naturally generalize the well-known binary Treaps, several new ideas are needed to analyze the *expected* search, update, and storage efficiency in terms of block-reads, block-writes, and blocks stored. We prove that searches have $O(\varepsilon^{-1} + \log_\alpha n)$ block-reads, that dynamic updates perform $O(\varepsilon^{-1} + \log_\alpha(n)/\alpha)$ block-writes and $O(\varepsilon^{-2} + (1 + \frac{\varepsilon^{-1}+\log n}{\alpha}) \log_\alpha n)$ block-reads, and that (α, ε)-RBSTs have an asymptotic load-factor of at least $(1 - \varepsilon)$ for every $\varepsilon \in (0, 1/2]$.

Thus (α, ε)-RBSTs improve on the known, uniquely represented B-Treap [Golovin; ICALP'09]. Compared with non-UR structures, the RBST is also, to the best of our knowledge, the first external memory structure that is storage-efficient and has a non-amortized, write-efficient update bound.

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

Keywords and phrases Unique Representation, Randomization, Top-Down Analysis, Block Search Tree, Write-Efficiency, Storage-Efficiency

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.47

Related Version *Full Version*: https://arxiv.org/abs/2303.04722

Funding This work was supported under the Australian Research Council Discovery Projects funding scheme (project number DP180102870). This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101019564) "The Design of Modern Fully Dynamic Data Structures (MoDynStruct)" and from the Austrian Science Fund (FWF) project Z 422-N and project "Fast Algorithms for a Reactive Network Layer (ReactNet)" P 33775-N, with additional funding from the *netidee SCIENCE Stiftung*, 2020–2024.

1 Introduction

Organizing a set of keys such that insertions, deletions, and searches are supported is one of the most basic problems in computer science that drives the development and analysis of structures with various guarantees and performance trade-offs. Binary search trees for example have several known height balancing methods (e.g. AVL and Red-Black trees) and weight balancing methods (e.g. BB[a] trees), some providing $\mathcal{O}(1)$ writes for *each* update [17].

Block Search Trees are generalizations of binary search trees that store in every tree node up to α keys and $\alpha + 1$ child pointers, where the array size α of the blocks is a (typically) fixed parameter. Since such layouts enforce a certain data locality, block structures are central for read- and write-efficient access in machine models with a memory hierarchy.

© Roodabeh Safavi and Martin P. Seybold;
licensed under Creative Commons License CC-BY 4.0

External Memory and deterministic B-Trees variants. In the classic External Memory (EM) model, the n data items of a problem instance must be transferred in blocks of a fixed size B between a block-addressible external memory and an internal Main Memory (MM), which can store up to M data items and perform computations. Typically, $n \gg M > B$ and the MM can only store a small number of blocks throughout processing, say $M/B = \mathcal{O}(1)$. Though Vitter's Parallel Disk Model can formalize more general settings [30, Section 2.1], we are interested in the most basic case with one, merely block-addressible, EM device and one computing device with MM. The cost of a computation is typically measured in the number of *block-reads* from EM (Is), the number of *block-writes* to EM (Os), or the total number of IOs performed on EM. For basic algorithmic tasks like the range search, block search trees with $\alpha = \Theta(B)$ support algorithms with asymptotically optimal total IOs (e.g. [30, Section 10]). Classic B-Trees [4] guarantee that every block, beside the root, has a load-factor of at least $1/2$ and that all leaves are at equal depth, using a deterministic balance criteria. The B*-Tree variant [23, Section 6.2.4] guarantees a load-factor of at least $2/3$ based on a (deterministic) overflow strategy that shares keys among the two neighboring nodes on equal levels. Yao's Fringe Analysis [32] showed that inserting keys under a random order in (deterministic balanced) B-Trees yields an expected asymptotic load-factor of $\ln(2) \approx 69\%$, and the expected asymptotic load-factor is $2\ln(3/2) \approx 81\%$ for B*-Trees. For general workloads however, maintaining higher load-factors (with even wider overflow strategies) further increases the update write-bounds in block search trees [3, 24]. The B$^+$-Tree variant for key-value pairs, which stores the values separate from the keys to increase the load-factor and thus *fan-outs*, is heavily used in practical file systems, see e.g. [14].

The IO-optimized[1] B$^\varepsilon$-Tree [13, 7] provides smooth trade-offs between $\mathcal{O}(\log_{1+\alpha^\varepsilon}(n)/\alpha^{1-\varepsilon})$ amortized IOs per update and $\mathcal{O}(\log_{1+\alpha^\varepsilon} n)$ block-reads (Is) for searches. E.g. tuning parameter $\varepsilon = 1$ retains the bounds of B-Trees and $\varepsilon = 1/2$ provides improved bounds. In the fully update IO-optimized case ($\varepsilon = 0$) however, searches have merely a $\mathcal{O}(\log n)$ bound for the number of block-reads. The main design idea to achieve this is to augment the non-leaf nodes of a B-Tree with additional "message buffers" so that a key insertion can be stored close to the root. Messages then only need propagation, further down the tree, once a message buffer is full (cf. [13, Section 3.4] and [7]). Clearly, the state of either of those (deterministic) structures depends *heavily* on the actual sequence in which the keys were inserted in them.

Uniquely Represented Data Structures. For security or privacy reasons, some applications, see e.g. [19], require data structures that do not reveal any information about historical states to an observer of the memory representation, i.e. evidence of keys that were deleted or evidence of the keys' insertion sequence. Data structures are called *uniquely represented* (UR) if they have this strong history independence property. For example, sorted arrays are UR, but the aforementioned search trees are not UR due to deterministic rebalancing. Early results [28, 29, 1] show lower and upper bounds on comparison-based search in UR data structures on the pointer machine. Using UR hash tables [9, 26] in the RAM model, pointer structures can also be mapped uniquely into memory. (See [9, Thm. 4.1], [10, Sec. 2].)

The Randomized Search Trees are defined by inserting keys, one at a time, in order of a random permutation into an initially empty *binary* search tree. The well-known Treap [27, 31] maintains the tree shape, of the permutation order, and supports efficient insertions, deletions, and searches (see also [25, Chapter 1.3]). Searches read with high probability $\mathcal{O}(\log n)$ tree

[1] B^ε-trees are usually called "write-optimized", though they aim to minimize IOs and not writes (Os).

Table 1 Bounds of bottom-up B-Treaps and the proposed top-down RBSTs.

UR Tree	Model		[# Blocks]	
B-Treap [20]	BEM	Size	$\mathcal{O}(n/\alpha)$	
		Update **(Os)** Search **(Is)**	$\mathcal{O}\left(\frac{1}{\delta}\log_\alpha n\right)$	, if $\alpha = \Omega\left(\ln^{\frac{1}{1-\delta}} n\right)$.
(α,ε)-RBST	EM	Size	$(1+\mathcal{O}(\varepsilon))\ n/\alpha$	
		Update **(Os)**	$\mathcal{O}(\varepsilon^{-1}+\log_\alpha(n)/\alpha)$	
		Update **(Is)**	$\mathcal{O}(\varepsilon^{-2}+(1+\frac{\varepsilon^{-1}+\log n}{\alpha})\log_\alpha n)$	
		Search **(Is)**	$\mathcal{O}(\varepsilon^{-1}+\log_\alpha n)$	

nodes, any insertion (or deletion) performs expected $\mathcal{O}(1)$ rotations (writes), and the expected subtree weight, beneath the updated key, is $\mathcal{O}(\log n)$. Beside UR, these remarkably strong bounds are central in the design of simple MM algorithms for dynamic problems. The *constant writes bound* of Treaps, and some Red-Black Tree variants, is particularly valuable for persistent data structures [16, 22], that is $\mathcal{O}(1)$ amortized writes are *not sufficient* to design fully-persistent data structures with linear space. The *logarithmic weight bound* for example is particularly valuable for obtaining a $\mathcal{O}(1)$ query time for order-queries in dynamic sets with $\mathcal{O}(\log n)$ expected update time [12, Section 5]. This is a central sub-problem in designing fully-persistent data structures, see [18] for example. Further important examples include asymmetric read/write cost settings [8, 5] and parallel computation [11], or simply storage media that can only endure few write-cycles.

Golovin [20] introduced the B-Treap as the first UR data structure for Byte-addressable External Memory (BEM). The B-Treap originates from a certain block storage layout of an associated (binary) Treap, i.e. the block tree is obtained, bottom-up, by iteratively placing subtrees, of a certain size, in one block node[2]. The author shows bounds of B-Treaps, under certain parameter conditions: If α is at least polylogarithmic, specifically $\alpha = \Omega((\ln n)^{\frac{1}{1-\delta}})$ for some $\delta \in (0,1)$, then updates have expected $\mathcal{O}(\frac{1}{\delta}\log_\alpha n)$ block-writes and range-queries have expected $\mathcal{O}(\frac{1}{\delta}\log_\alpha n + k/\alpha)$ block-reads, where k is the output size (see Theorem 1 in [20]). The paper also discusses experimental data, from a non-dynamic implementation, that shows an expected load-factor close to 1/3 [20, Sec.6]. Though this storage layout approach allows leveraging known bounds of Treaps to analyze B-Treaps, the block size conditions are limiting for applications of B-Treaps in algorithms. Since the update algorithm is rather involved, Golovin [21] proposed a simpler, Skip-List based EM approach for a UR solution.

The known Skip-List based approaches [21] and [6] avoid the restrictions on α, however both require an involved form of history independent hash-table for strings of key values (see [21, Section 2.2.2]). Moreover, the B-Skip-List [21] provides only a logarithmic $\mathcal{O}(\log n)$ depth bound (see [6, Lemma 15]) and the EM Skip-List in [6] offers only a weaker notion of History Independence than UR and has *logarithmic* block-writes (cf. [6, proof of Lem.19]).

Contribution and Paper Outline. Our work provides the first UR EM search tree that is generally applicable for all block sizes $\alpha \geq 2$, whilst providing bounds that improve on the only known UR EM search tree solution, B-Treap [20].

[2] E.g. byte-addressable memory is required for intra-block child pointers and each key maintains a size field that stores its subtree size (in a binary Treap), resulting in logarithmic writes (cf. Os in Table 1).

We introduce a *new design technique*, and provide a self-contained analysis, that leads to a two-layer randomized data structure called (α, ε)-Randomized Block Search Trees (RBSTs), where α is the block size and $\varepsilon \in (0, 1/2]$ a tuning parameter for the space-efficiency of our secondary layer (see Section 2). Without the secondary structures, the RBSTs are precisely the distribution of trees that are obtained by inserting the keys, one at a time, under a random order into an initially empty block search tree (e.g. $\alpha = 1$ yields the Treap distribution). RBSTs are UR and the algorithms for searching are simple. In Section 2.1, we give a partial-rebuild algorithm for dynamic updates that occupies $\mathcal{O}(1)$ blocks of temporary MM storage and performs a number of writes (Os) to EM that is proportional to the structural change in the RBST. Its number of reads (Is) is bounded within a small factor of the writes. We prove three performance bounds for (α, ε)-RBSTs in terms of block-reads, block-writes, and blocks stored.

Section 3 shows that searching reads expected $\mathcal{O}(\varepsilon^{-1} + \log_\alpha n)$ blocks, and that RBST depth is with high probability $\mathcal{O}(\varepsilon^{-1} + \sqrt{\varepsilon^{-1} \ln n} + \log_\alpha n)$ for all $\alpha \geq 2$. If $\alpha = \text{polylog}(n)$, RBST depth is with high probability $\mathcal{O}(\varepsilon^{-1} + \log_\alpha n)$. Section 4 shows that (α, ε)-RBSTs have an expected asymptotic load-factor of at least $(1 - \varepsilon)$ for every $\varepsilon \in (0, 1/2]$. Combining both analysis techniques, we show in Section 5 that dynamic updates perform expected $\mathcal{O}(\varepsilon^{-1} + \log_\alpha(n)/\alpha)$ block-writes and expected $\mathcal{O}(\varepsilon^{-2} + (1 + \frac{\varepsilon^{-1} + \log n}{\alpha}) \log_\alpha n)$ block-reads.

See Table 1 for an overview of the write bounds (Os), and read bounds (Is), of the known bottom-up B-Treap and our proposed top-down RBST. To compare the update ***write-bounds*** (Os), note that RBSTs have *constant* $\mathcal{O}(\varepsilon^{-1})$ writes for all α that are admissible in B-Treaps, whereas B-Treaps write at least *logarithmic* $\Omega(\log_\alpha n)$ blocks. This is a significant asymptotic improvement. To compare the update ***read-bounds*** (Is), note that the factor $(1 + \frac{\log n}{\alpha}) = \mathcal{O}(1)$ for *all* $\alpha = \Omega(\ln^{\frac{1}{1-\delta}} n)$ with $\delta \in (0, 1)$ that are admissible in B-Treaps, whereas the factor $\frac{1}{\delta} \to \infty$ for α close to $\ln(n)$. That is, the RBST bounds improve on *all* B-Treap bounds. Thus, RBSTs are simple UR search trees, for *all* block sizes $\alpha \geq 2$, that simultaneously provide improved search, storage utilization, and write-efficiency, compared to the bottom-up B-Treap [20].

Comparing also with non-UR structures, (α, ε)-RBSTs are, to the best of our knowledge, the first EM structure that provides storage-efficiency and write-efficiency for *every* update. The decisive novelty of our tree structure is that all block-nodes of the primary layer use the *maximum* fan-out and that the block-nodes of the secondary layer use a *weight-proportional* fan-out, i.e. proportional to the number of keys that are stored in the subtree. The design of our secondary layer structures has thus *nothing* in common with the "message-buffers" in B^ε-trees. Our result is incomparable to IO-optimized B^ε-trees: Though RBSTs are UR and provide better bounds for space, searching (Is), and writes (Os), the block-reads (Is) for updates are $\Omega(\log_\alpha n)$ and thus worse than the IO-bound of the (non-UR) B^ε-Tree.

2 Randomized Block Search Trees

Unlike the well-known B-Trees that store between α and $\alpha/2$ keys in every block and have all leaves at equal depth, our definition of (α, ε)-RBSTs does not strictly enforce a minimum load for every single node. Like binary Treaps, the shape of an RBST $T_X(\pi)$, over a set X of n keys, is defined by an incremental process that inserts the keys $x \in X$ in a random order π into an initially empty block search tree structure, i.e. with ascending priority values $\pi(x)$. The actual algorithms for dynamic updates are discussed in Section 2.1.

This section defines the basic tree structure that supports both, reporting queries (i.e. range search) and order-statistic queries (i.e. range counting and searching the k-th smallest value). Section 2.1 then shows how to avoid explicitly maintaining the subtree weights

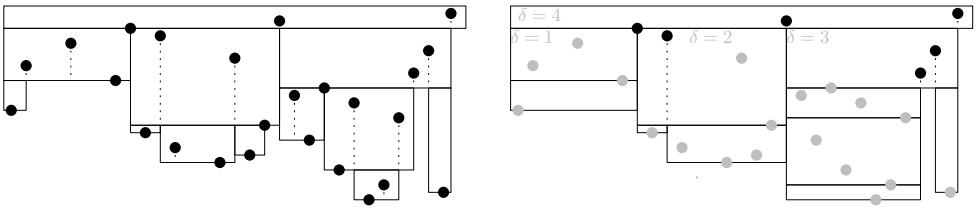


Figure 1 Cartesian representations of two RBSTs that shows the i-th smallest key $x_i \in X$ with priority value $\pi(i)$ as point $(i, -\pi(i))$ in the plane ($n = 25$ and $\alpha = 3$). (Cf. [31] for Cartesian Trees.) Active separators of a block v are shown with dotted vertical lines and the horizontal lines are $p^\dagger(v)$. The left tree has $\beta = 0$ and the right tree has buffering subtrees. Note that all, except the last block, on a root-to-leaf path are full.

of nodes close to the root, which yields improved update write-bounds for the case that order-statistic queries are not required in applications. Next, we define the structure of RBSTs and state their invariants.

Let array size $\alpha \geq 1$ and buffer threshold $\beta \geq 0$ be fixed integer parameters. Every RBST block node stores a fixed-size array for α keys, $\alpha + 1$ child pointers, and one parent pointer. We use the key-value x of minimum priority $\pi(x)$ from those keys contained in the node's array as the UR label of the block-node. Every child pointer is organized as a *pair*, storing the pointer to the child block and the total number of keys $s \geq 1$ contained in that subtree[3]. All internal blocks of an RBST are full, i.e. they store exactly α keys. Any block has one of two possible states, either ***non-buffering*** ($s \geq \alpha + \beta$) or ***buffering*** ($s < \alpha + \beta$). Though (α, ε)-RBSTs will use $\beta = \Theta(\alpha^2/\varepsilon)$, we first give the definition of the primary tree without the use of buffers ($\beta = 0$), i.e. every internal block remains non-buffering.

For $\beta = 0$, the tree shape is defined by the following incremental process that inserts the keys in ascending order of their priority $\pi(\cdot)$. Starting at the root block, the insertion of key x first locates the leaf block of the tree using the search tree property. If the leaf is non-full, x is emplaced in the sorted key array of the node. Otherwise, a new child block is allocated (according to the search tree property) and x is stored therein. This concludes the definition of the tree structure for $\beta = 0$.

(Invariant 1:) Thus, any internal block stores those α keys that have the smallest priority values in its subtree[4].

See Figure 1 (left) for an example on 25 keys with $\beta = 0$ that consists of 12 blocks. It demonstrates that leaves may merely store one key in their block. We address this storage inefficiency issue next.

The "high-level idea" for $\beta > 0$ is to delay generating new leaves until the subtree contains a sufficiently large number of keys. To this end, we use secondary structures, which we call ***buffers***[5], that replace the storage layout of all those subtrees that contain $< \alpha + \beta$ keys.

(Invariant 2:) Thus, all remaining block nodes of the primary structure are full and their children are either a primary block or a buffer.

There are several possible ways for UR organization of buffers, for example using a list of at most $\lceil (\alpha + \beta)/\alpha \rceil = \mathcal{O}(\alpha/\varepsilon)$ blocks that are sorted by key values. We propose the following structure that leads to stronger bounds.

[3] E.g. subtree weights are immediately known to the parent, without reading the respective child blocks.
[4] For example, $\alpha = 1$ and $\beta = 0$ yields the well-known Treap and $\alpha > 1$ search trees with fan-out $\alpha + 1$.
[5] Though we use the word "buffer" as it describes this purpose best, our definition has nothing to do with the "message buffers" in [13, 2].

Secondary UR Trees: Buffers with weight proportional fan-out. Our buffers are search trees that consist of nodes similar to the primary tree. They also store the keys with the α smallest priority values, from the subtree, in their block. However, the fan-out of internal nodes varies based on the number of keys $n \leq \alpha + \beta$ in the subtree. Define $\delta(n) = \min\{\alpha + 1, \max\{1, \lceil \frac{n-\alpha}{\rho} \rceil\}\}$ as the ***fan-out bound*** for a subtree of weight $n < \alpha + \beta =: \alpha + (\alpha + 1)\rho$. We defer the calibration of the parameter $\rho = \Theta(\alpha/\varepsilon)$ to the start of the proof of Theorem 9. The subtree weight of buffering nodes is maintained explicitly, with the aforementioned in-pointers. To obtain UR, the buffer layout is again defined recursively solely based on the set of keys in the buffer and the random permutation π.

For fan-out bound $\delta = 1$, the subtree root has at most one child, which yields a list of blocks. The keys inside each block are stored in ascending key order, and the blocks in the list have ascending priority values. That is, the first block contains those keys with the α smallest priorities, the second block contains, from the remaining $n - \alpha$ keys, those with the α smallest priorities, and so forth.

For fan-out bound $\delta \in [2, \alpha + 1]$, we also store the keys with the α smallest priorities in the root. We call those keys with the $\delta - 1$ smallest priority values the ***active separators*** of this block. The remaining $n - \alpha$ keys in the subtree are partitioned in δ sets of key values, using the active separators only. Each of these key sets is organized recursively, in a buffer, and the resulting trees are referenced as the children of the active separators.

This concludes the definition of our tree structure.

See the right part of Figure 1 for an example of the (α, ε)-RBST on the same permutation (y-coordinates) and 25 keys (x-coordinates) that consists of 11 blocks. (See [31] for Cartesian Tree representations.) The active separators are shown in black, non-active separators in gray, and the block nodes as rectangles.

Some remarks on the definition of primary and secondary RBST nodes are in order. The buffer is UR since the storage layout only depends on the priority order π of the keys, the number of keys in the buffer, and the order of the key values in it. Note that summing the state values of the child pointers yields $s_1 + \ldots + s_\delta + \alpha = n$ the weight of the subtree, without additional block reads. Moreover, whenever an update in the secondary structure brings the buffer's root above the threshold, i.e. $s_1 + \ldots + s_{\alpha+1} \geq \beta$, we have that its root contains those keys with the α smallest priorities form the set and all keys are active separators, i.e. the buffer root immediately provides both invariants of blocks from the primary tree.

Note that this definition of RBSTs not only naturally contains the randomized search trees of block size α, but also allows a *smooth trade-off* towards a simple list of blocks (respectively $\varepsilon = \infty$ and $\varepsilon = 0$). Though leaves are non-empty, it is possible that the tree shape defined by the random permutation π yields leaves with load as low as $1/\alpha$ in RBSTs.

As stated at the beginning of this section, the definition above does not yield an efficient algorithm to actually maintain RBSTs. The remainder of this section specifies our algorithms for searching, dynamic insertion, and dynamic deletion in RBSTs. Our analysis is presented in Sections 3, 4, and 5.

Successor and Range Search in (α, ε)-RBSTs. As with ordinary B-Trees, we determine with binary search on the array the successor of search value q, i.e. the smallest active separator key in the block that has a value of at least q, and follow the respective child pointer. If the current search node is buffering, then we additionally check the set of non-active separators in the block (i.e. all keys) during the search to find the actual successor of q from X. The search descends until the list, i.e. block nodes with fan-out ≤ 1, in the respective RBST buffer is found. Since the lists are sorted by priority, we check all keys, by scanning

the $\mathcal{O}(1/\varepsilon)$ blocks of the list, to determine the successor of q. This point-search extends to range-queries in the ordinary way by considering the two search paths of the range's boundary $[q, q']$.

To summarize, successor and range search occupies $\mathcal{O}(1)$ blocks in main memory, does not write blocks, and the number of block-reads is $\mathcal{O}(D)$, where D is the depth of the RBST.

2.1 Insertion and Deletion via Partial-Rebuilds

As with the Treaps, the tree shape of (α, ε)-RBSTs is solely determined by the permutation π of the keys. For $X \cup \{x\} = X'$, any update method that transforms $T_X(\pi)$ into $T_{X'}(\pi)$ and vice-versa are suitable algorithms for insertions and deletions. Unlike Treaps, that use rotations to always perform insertions and deletions at the leaf level, rotations in block search trees seem challenging to realize and analyze. We propose an update algorithm that rebuilds subtrees in a top-down fashion, which has vague similarities to rebalancing Scapegoat Trees.

The main difficulty for RBST update algorithms is to achieve IO-bounds for EM-access that are (near) proportional to the structural change in the trees, *while* having an $\mathcal{O}(1)$ worst-case bound on the temporary blocks needed in main memory. The *partial*-rebuild algorithm we introduce in this section has bounds on the number of block-writes (Os) in terms of the structural change of the update, the expected Os and Is are analyzed in Section 5.

▶ **Observation 1.** *Let π be a permutation on the key universe U and $X' = X \cup \{x\}$ the keys in the tree, after and before the insertion of x. Let m be the number of blocks in $T_X(\pi)$ that are different from the blocks in $T_{X'}(\pi)$ and m' the number of blocks in $T_{X'}(\pi)$ that are different from the blocks in $T_X(\pi)$. Let D and D' be the depth of the subtrees in $T_X(\pi)$ and $T_{X'}(\pi)$ that contain those blocks.*

The rebuild algorithm in this section performs $\mathcal{O}(m + m')$ block-writes to EM, reads $\mathcal{O}(m' + D' \cdot m)$ blocks, and uses $\mathcal{O}(1)$ blocks of temporary MM storage during the update.

Our rebuild-algorithm aims at minimizing the number of block-writes (Os) in an update, using a greedy top-down approach that builds the subtree of $T_{X'}$ in auxiliary EM storage while reading the keys in T_X from UR EM. On completion, we delete the m obsolete blocks from UR memory and write the m' result blocks back to UR EM to obtain the final tree $T_{X'}$. This way, our update algorithm will only require $\mathcal{O}(1)$ blocks of temporary storage in MM. The basic idea is as follows. Given a designated separator-interval $(\ell(u), r(u))$ when assembling block u for writing to auxiliary EM, we determine those keys with the α-smallest priority values by searching in T_X, place them in the array of u, determine the fan-out δ_u and the active separators of block node u, and recursively construct the remaining subtrees for those keys within each section between active separators of u. Eventually, we free the obsolete blocks of T_X form UR and move the resulting subtree from the auxiliary to UR memory.

Next we discuss all possible rebuild cases when ***inserting*** a new key x to the tree, their realization is discussed thereafter. For a block node v, let $X(v)$ denote the set of keys stored in the subtree of v, $B(v) \subseteq X(v)$ the keys stored in the array of v itself, $p^\dagger(v) = \max\{\pi(y) : y \in B(v)\}$, and $p^*(v) = \min\{\pi(y) : y \in B(v)\}$[6].

Consider the search path for x in T_X. Starting at the root, we stop at the first node v that i) must store x in its array (i.e. $p^\dagger(\text{parent}(v)) < \pi(x) \leq p^\dagger(v)$) or ii) that requires rebuilding (i.e. must increase its fan-out). To check for case ii), we use the augmented child

[6] For example, the key y with $\pi(y) = p^*(v)$ is the UR label of the block v. See also Figure 1 for $p^\dagger(v)$.

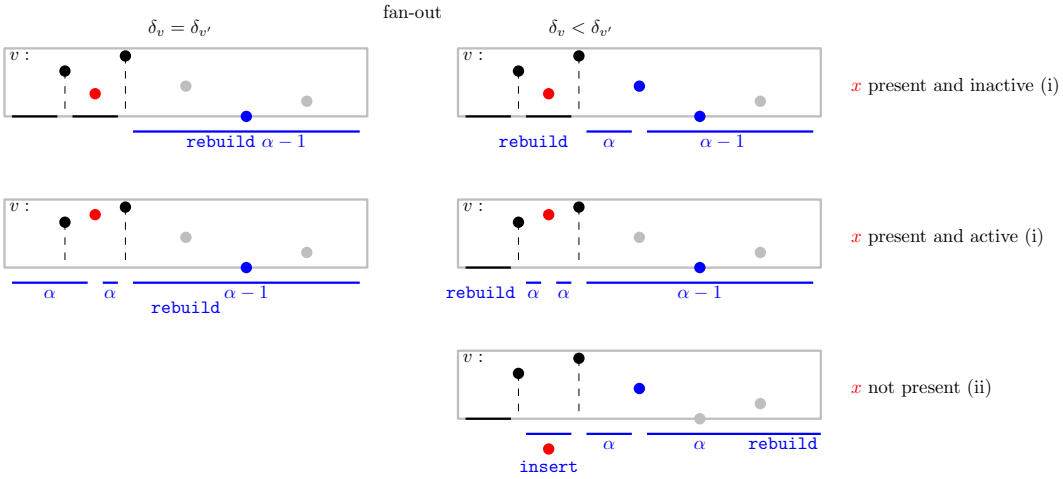


Figure 2 Rebuild cases (blue) for insertion of key x (red) in block v of an RBST.

references of buffering node v to determine the number of keys in the resulting subtree v'. Thus, both fan-outs, δ_v pre-insertion and $\delta_{v'}$ post-insertion, are known without additional block reads from the stored subtree weights.

If neither i) nor ii) occurs, the search halts in a buffer list ($\delta_v = \delta_{v'} \leq 1$) and we simply insert the key and rewrite all subsequent blocks of the list in one linear pass. Note that, if the list is omitted (as there are no keys in the active separator interval), the insertion must simply allocate a new block that solely stores x to generate the output RBST. It remains to specify the insertion algorithm for case i) and ii). See Figure 2 for an overview of all cases.

In ii), v is a buffer node that must increase its fan-out. It suffices to build two trees from one subtree, each containing the top α keys in their root. If the new key x is contained in this interval, we pass x as additional "carry" parameter to the procedure. If x is not contained in this interval, the insertion continues on the respective child of v that must contain x. (See Figure 2 bottom.) Note that ≤ 3 subtrees of v' are built in auxiliary EM, regardless of α.

In i), v is an internal node that stores α keys and there are four cases; depending on if x is an active separator in v' and if the fan-out stays or increases (see Figure 2). In all cases, it suffices to rebuild at most three subtrees. Two trees for the two separator-intervals that are incident to x and one tree that contains the two subtrees of the separator-intervals incident to the key y that is displaced from block $B(v)$ by the insertion of x, i.e. $\pi(y) = p^{\dagger}(v)$.

To complete the insertion algorithm, it remains to specify our top-down greedy rebuild procedure for a given interval between two active separators. First, we isolate the task of finding the relevant keys, needed for writing the next block to auxiliary EM within the following range-search function $\texttt{top}(k, v, (\ell, r))$. Given a subtree root v in T_X, a positive integer $k \leq \alpha$, and query range (ℓ, r), we compute those keys from $Y = X(v) \cap (\ell, r)$ that have the k smallest priority values, together with their respective new subtree weights $w_0 + \ldots + w_k = |Y| - k$, based on a naïve search that visits (in ascending key order) all descendants of v that overlap with the query range.

For our $k = \alpha$ and $k = \alpha - 1$ rebuilds, we run the respective `top`-query on the designated interval range and check if x is stored in the output block or beneath it. Then we determine the fan-out from the range count results $\{w_i\}$, which determines the active separators in the output block. We allocate one block in auxiliary EM for each non-empty child-reference and set their parent pointers. Finally, we recursively rebuild the subtrees for the intervals of the active sections, passing them the reference to the subtree root in T_X that holds all necessary

keys for rebuilding its subtree. Note that for this form of recursion, using parent pointers allows to implement the rebuild procedure such that only a constant number of temporary blocks of MM storage suffice. After the subtrees are built in the auxiliary EM, we delete the m old blocks from UR and move the m' blocks to UR to obtain the tree T'.

Note that ***deleting*** has the same cases for rebuilding sections of keys between active separators. Moreover, if a deletion leads to a buffering block ($\delta_{v'} < \alpha + 1$), then all its children are buffering and thus store their subtree weight together with the reference to it. Thus, determining the fan-out for deletions requires no additional block reads.

Clearly, the number of block writes is $\mathcal{O}(m + m')$ and the total number of block reads of all `top`-calls is $\mathcal{O}(D' \cdot m)$, from a charging argument: That is, any one block u of the m blocks in old tree is only read by a `top` call if the key range $(\ell(u), r(u))$ of u intersects the query range (ℓ, r) of the `top` call, i.e. $(\ell, r) \cap (\ell(u), r(u)) \neq \emptyset$. Consequently (ℓ, r) contains the smallest key, the largest key, or all keys of $B(u)$. The number of either such reads that is charged to a block u (from the m blocks) is bounded by D', since those blocks from the output tree that issue the calls to `top` have nested intervals $(\ell_1, r_1) \supset (\ell_2, r_2) \supset \dots$ that are contained in each other, by the search tree property (of the m' result blocks).

Our bounds for Observation 1 in terms of writes (Os) and reads (Is) will follow from our analysis of the depth and size of RBSTs in the next sections.

3 Bounds for Depth

Since successful tree searches terminate earlier, it suffices to analyze the query cost for keys $q \notin X$, i.e. unsuccessful searches. In this section, we bound the expected block reads for searching for some fixed q in the block trees T_X, where the randomness is over the permutations $\pi \in \mathrm{Perm}(n)$, i.e. the set of bijections $\pi : X \to \{1, \dots, n\}$.

Consider the sequence $v_1, v_2, \dots$ of blocks of an RBST on the search path from the root to some fixed q. Since the subtree weight is a (strictly) decreasing function, we have that the fan-out δ of the nodes is a non-increasing function that eventually assumes a value ≤ 1. From the definition of δ, we have that there are (in the worst case) at most $O(1/\varepsilon)$ blocks in the search path with $\delta = 1$. Thus, it suffices to bound the expected number of internal blocks with $\delta \geq 2$, which all have the search tree property $[\ell(v_i), r(v_i)] \supset (\ell(v_{i+1}), r(v_{i+1})) \ni q$.

▶ **Lemma 2.** *Let X be a set of n keys, $q \in \mathbb{R} \setminus X$, and $\alpha \geq 2$ an integer. The expected number of primary tree nodes ($\delta = \alpha + 1$) in an (α, ε)-RBST on X that contain q in their interval is $\mathcal{O}(\log_\alpha n)$. The expected number of secondary tree nodes ($\delta < \alpha + 1$) that have q in their interval is $\mathcal{O}(1/\varepsilon)$.*

In particular, the expected number of blocks in the search path of q is $\mathcal{O}(\varepsilon^{-1} + \log_\alpha n)$.

Further, the depth of an (α, ε)-RBST on X is with high probability less than $\mathcal{O}(\varepsilon^{-1} + \log n)$.

The block structure of RBSTs do not allow a clear extension of the Treap analysis based on "Ancestor-Events" [27] or the "Backward-Analysis" of point-location cost in geometric search structures [15, Chapter 6.4] to obtain bounds on block-reads (Is) that are stronger than the high-probability $O(\log n)$-depth bound of ordinary Treaps. Our proof technique is vaguely inspired by a top-down analysis technique of randomized algorithms for *static* problems that consider "average conflict list" size. However, several new ideas are needed to analyze a dynamic setting where the location q is user-specified and not randomly selected by the algorithm. Our proof uses a partition in layers whose sizes increase exponentially with factor α and a bidirectional counting process of keys in a layer of that partition.

Proof. Recall that there are in the worst case $\mathcal{O}(1/\varepsilon)$ nodes with fan-out $\delta \leq 1$ on the search path of q. Consider the active separators $\{x_1, \ldots, x_d\}$ of the RBST blocks that contain q in their search-interval. This set consists of two sequences $\{x_i^+\}$ and $\{x_i^-\}$ of strictly decreasing and increasing key values, respectively. Since their priority values are strictly increasing, i.e. $\pi(x_1) < \pi(x_2) < \ldots < \pi(x_d)$, we have from the known bounds of ordinary Treaps that $d = \mathcal{O}(\log n)$ with high probability [27, Lem. 4.8]. Thus, RBST depth is with high probability less than $\mathcal{O}(\varepsilon^{-1} + \log n)$.

To show our sharper search bound, let random variable D_q be the number of primary tree nodes v that contain q in their interval $[\ell(v), r(v)]$. Partition $\{1, \ldots, n\}$ with intervals of the form $[\alpha^i, \alpha^{i+1})$ for indices $0 \leq i \leq \lfloor \log_\alpha n \rfloor$. E.g. permutation π induces an assignment of keys $x \in X$ to an unique layer index, i.e. i with $\pi(x) \in [\alpha^i, \alpha^{i+1})$. For internal node v let $p^*(v) = \min\{\pi(x) : x \text{ is stored in } v\}$ and $p^\dagger(v)$ is the maximum over the set. Note that $p^*(v) + \alpha - 1 \leq p^\dagger(v)$, since every internal node stores exactly α keys. Thus, D_q counts nodes v that either have both $p^*(v), p^\dagger(v) \in [\alpha^i, \alpha^{i+1})$ or have that $p^\dagger(v)$ has a larger layer index than i. We bound the expected number of nodes of the first kind, since there are in the worst case at most $\lfloor \log_\alpha n \rfloor$ from the second kind. Defining for every layer index i a random variable V_i that counts the tree nodes in that layer

$$V_i(\pi) = \left| \left\{ v \in T_X(\pi) \ : \ \ell(v) < q < r(v) \text{ and } p^*(v), p^\dagger(v) \in [\alpha^i, \alpha^{i+1}) \right\} \right| \quad ,$$

we have that $D_q \leq \lfloor \log_\alpha n \rfloor + V$, where $V = \sum_{i \geq 0} V_i$ is the total number of blocks of the first kind. The remainder of the proof shows bounds for V_i.

Let $K_i = \{x \in X : \pi(x) < \alpha^{i+1}\}$ for each index i, $K_i^- = \{x \in K_i : x < q\}$, and $K_i^+ = K_i \setminus K_i^-$. Thus, $K_i \subset K_{i+1}$, and $|K_i| = \alpha^{i+1} - 1$. Define Y_i^- to be the number of consecutive keys form K_i^- that are less than q but not contained in K_{i-1}. Analogously, random variable Y_i^+ counts those larger than q and $Y_i := Y_i^- + Y_i^+$ is the number of consecutive keys of K_i, whose range contain q, but do not contain elements from K_{i-1}. Since all separators of primary nodes are active and internal nodes store exactly α keys, we have $V_i(\pi) \leq Y_i(\pi)/\alpha$ for every $\pi \in \mathrm{Perm}(n)$.

Next we bound $\mathbb{E}[Y_i^-]$ based on a sequence of binary events: Starting at q, we consider the elements $x \in K_i^-$ in descending key-order and count as "successes" if $x \in K_i \setminus K_{i-1}$ until one "failure" ($x \in K_{i-1}$) is observed. If K_i^- has no more elements, the experiment continues on K_i^+ in ascending key order. Defining Z_i as the number of successes on termination, we have $Y_i^-(\pi) \leq Z_i(\pi)$. The probability to obtain failure, after observing j successes, is $\frac{|K_{i-1}|}{|K_i|-j}$, which is at least $p_i := \frac{|K_{i-1}|}{|K_i|}$ for all $j \geq 0$ and $i > 0$.

Hence $\Pr[Z_i = j] \leq \Pr[Z_i' = j+1]$ where random variable $Z_i' \sim NB(1, p_i)$, i.e. the number of trials to obtain one failure in a sequence of independent Bernoulli trials with failure probability p_i. Since $\mathbb{E}[Z_i'] = 1/p_i$, we have $\mathbb{E}[Y_i^-] \leq \mathbb{E}[Z_i] < \mathbb{E}[Z_i'] = \frac{1}{p_i} = \frac{\alpha^{i+1}-1}{\alpha^i - 1} < \frac{\alpha}{1-1/\alpha^i}$.

We thus have $\mathbb{E}[V_i] \leq \mathbb{E}[Y_i]/\alpha < \frac{2}{\alpha} \frac{\alpha}{1-1/\alpha^i} \leq 4$ for all $\alpha \geq 2$, which shows the lemma's statement for the primary tree nodes.

Since there are in the worst case $\mathcal{O}(1/\varepsilon)$ nodes with fan-out $\delta \leq 1$ on a search path, it remains to bound the expected number of secondary nodes with a fan-out in $[2, \alpha]$ on a search path. If there are any such nodes to count, consider the top-most buffering node v and let $n' := |X(v)|$ be its subtree weight. For any fixed integer n', random variables regarding the subtree only depend on the relative order of the n' keys, which are uniform from $\mathrm{Perm}(n')$. Since the expected number of keys in either of the $\delta(n')$ sections is $(n'-\alpha)/\delta(n')$, the expected number of keys in the section of q has an upper bound of the form $(n' - \alpha)/\delta(n') = \mathcal{O}(\alpha/\varepsilon)$. Since the bound holds for all n', this bound holds unconditionally. The lemma's $\mathcal{O}(1/\varepsilon)$ expectation bound on the nodes follows from the fact that all secondary nodes (with $\delta \geq 2$) of a search path store exactly α keys. Thus, the expected number of such nodes is $\mathcal{O}(1/\varepsilon)$. ◀

Next, we show our depth bound that improves on the simpler $\mathcal{O}(\varepsilon^{-1} + \ln n)$ bound above. We first give a tail-estimate for the number of primary nodes and then an estimate for the number of secondary nodes on a search-path to the fixed query point q.

▶ **Lemma 3.** *Let $\alpha \geq 2$. The random variable D_q is with high probability $\mathcal{O}(\log_\alpha n)$.*

We will tighten above's analysis by considering the relative priority orders of the Y_i keys that are counted in layer i, where $i \leq r = \mathcal{O}(\log_\alpha n)$. Indeed, even if random variable Y_i exceeds its mean $\Theta(\alpha)$ by a large factor, the blocks of a search path to q can only contain all Y_i keys if also their relative priority-order have a, very specific, deep block structure. This observation allows to obtain a much stronger tail bound.

Proof. Let $\hat{Y}_i$ be the number of keys from $K_i \setminus K_{i-1}$ that are stored in the block of a node on the search path to q, i.e. $\hat{Y}_i \leq Y_i$. We will show for the random variables $\hat{Y}_i$ that $\Pr[\hat{Y}_i = c\alpha] < (e^2/c)^{c\alpha}$ to obtain a tailbound that is strong enough for a union bound of all $n+1$ possible queries $q \notin X$. Recall from Stirling's formula, that $1 < \frac{n!}{\sqrt{2\pi n}(n/e)^n} < e^{1/12n}$ for all $n \geq 1$.

To observe $c\alpha$ keys from layer i stored in c blocks of a search path, the α keys stored in the last block must all have priorities larger than the first $(c-1)\alpha$ keys, the α keys in the second to last block must all have priorities larger than the first $(c-2)\alpha$ keys, and so forth. Further, each of the $\alpha!$ relative orders of any one such batch of α keys does not affect them being stored in the block of a search path. Thus the probability that $c\alpha$ keys have relative priorities to form c blocks of a search path is at most

$$\frac{(\alpha!)^c}{(c\alpha)!} < \frac{(e^{\frac{1}{12\alpha}}\sqrt{2\pi\alpha}(\alpha/e)^\alpha)^c}{(c\alpha/e)^{c\alpha}} = e^{\frac{c}{12\alpha}}(2\pi\alpha)^{\frac{c}{2}}\left(\frac{\alpha/e}{c\alpha/e}\right)^{c\alpha} = e^{\frac{c}{12\alpha}}\left(\frac{\sqrt{2\pi\alpha}}{c^\alpha}\right)^c < e^{\frac{c}{12\alpha}}\left(\frac{e}{c}\right)^{\alpha c},$$

where the last inequality is due to $\sqrt{2\pi\alpha} < e\sqrt{\alpha}$ and $\frac{e\sqrt{\alpha}}{c^\alpha} \leq (e/c)^\alpha \Leftrightarrow \sqrt{\alpha} \leq e^{\alpha-1}$ being true for all $\alpha \geq 2$. Thus $\Pr[\hat{Y}_i = c\alpha] < (e^2/c)^{c\alpha}$.

For $c \geq e^3$, $\Pr[\hat{Y}_i = c\alpha] < (\frac{1}{e^\alpha})^c =: z^c$ and we have

$$\Pr[\hat{Y}_i \geq c\alpha] \leq \sum_{j \geq c} \sum_{0 \leq k < \alpha} \Pr[\hat{Y}_i = j\alpha + k] \leq \alpha \sum_{j \geq c} \Pr[\hat{Y}_i = j\alpha] \leq \alpha \sum_{j \geq c} z^j = \alpha \frac{z^c}{1-z} < 2\alpha/e^{\alpha c}.$$

From the layer partition, we have that the outcome of Y_i is independent of Y_{i+1}, i.e. bounds for Y_i are with respect to $|K_i|$ and regardless of the actual set K_i, and all relative priority-orders of the keys in $K_i \setminus K_{i-1}$ are equally likely. Since across all r layers the probability bound for the sum is maximized when all r factors have equal thresholds, we have

$$\Pr[\hat{Y}_1 + \ldots + \hat{Y}_r \geq c\alpha r] \leq \prod_{i=1}^{r} \Pr[\hat{Y}_i \geq c\alpha] < \left(\frac{2\alpha}{e^{\alpha c}}\right)^r \leq \frac{2\alpha \cdot n^{1+1/\log_2 \alpha}}{n^{c\alpha/\ln \alpha}} \leq \frac{n^3}{n^{c\alpha/\ln \alpha}} < n^{3-c},$$

where the third last inequality uses that $\log_\alpha(n+1) \leq r \leq \log_\alpha(n) + 1$, the second last inequality uses that $\alpha \geq 2$ and $2\alpha \leq n$, and and the last inequality uses that $\alpha/\ln \alpha > 1$. ◀

▶ **Lemma 4.** *The depth of a buffering subtree is with high probability $\mathcal{O}(\varepsilon^{-1} + \sqrt{\varepsilon^{-1} \log n})$. If $\alpha = \text{polylog}(n)$, then the depth of a buffering subtree is with high probability $\mathcal{O}(\varepsilon^{-1} + \log_\alpha n)$.*

We believe that Lemma 4 cannot be improved substantially without redesigning the secondary structures, update algorithms, and proofs. Specifically, parameter constellations with small $\delta(n) = \omega(1)$ provide a major obstacle in obtaining better bounds.

Proof. For the number of secondary tree nodes, which have $\delta_v \in [2, \alpha]$ and store exactly α keys, we observe that having d such nodes on a search path requires that the topmost node has one child with weight $N_1 \geq (d-1)\alpha + \rho$, and the second topmost node must have one child with weight $N_2 \geq (d-2)\alpha + \rho$, and so forth, where the RBST parameter $\rho = \Theta(\alpha/\varepsilon)$. Thus, at least $d/4$ nodes must have weight $N_i \geq (d/2)\alpha + \rho$, and consequently fan-out value at least $k \geq \delta((d/2)\alpha + \rho) = \Omega(\varepsilon d)$. Using our $\Pr[\delta(N_i) \geq k] \leq e^{2-k}$ bound of Corollary 17 on the fan-out of a child, we obtain the probability of having d such nodes on a search path is at most $\prod_{i=d/4}^{d/2} \exp(2 - \delta(N_i)) \leq e^{-\Omega(\varepsilon d^2)}$, i.e. a high-probability bound for all $d = \omega(1)$.

Thus, the number secondary nodes ($\delta_v \geq 2$) is at most $d = O(\log_\alpha n)$ with probability at least $1 - n^{-\Omega(\varepsilon \ln(n)/(\ln \alpha)^2)}$, and $d = O(\sqrt{\varepsilon^{-1} \ln n})$ with probability at least $1 - n^{-\Omega(1)}$. ◀

4 Bounds for Size using a Top-Down Analysis

Our analysis will frequently use the following characterization of the partitions of a set X of n keys that are induced by the α elements of the smallest priority values from the set.

▶ **Observation 5.** *There is a bijective mapping between the partitions on n keys, induced by the first α keys from $\pi \in \mathrm{Perm}(n)$, and the solutions to the equation $X_1 + X_2 + \cdots + X_{\alpha+1} = n - \alpha$, where the variables X_i are non-negative integers. This bijection implies that the solutions of the equation happen with the same probability.*

In other words, X_i is the number of keys in the i-th section beneath the root of T_X, where $1 \leq i \leq \alpha + 1$. Thus, X_i can be considered as the number of consecutive keys from X that are between the i-th and $(i+1)$-th key stored in the root. For example, in an RBST on the keys $\{1, \ldots, 10\}$ and block size $\alpha = 3$, a root block consisting of the key values 4, 7, 8 is characterized by the assignment $X_1 = 3,\ X_2 = 2,\ X_3 = 0,\ X_4 = 2$. Next we analyze the effect of our secondary structures, since the size of RBSTs without buffer ($\beta = 0$) can be dramatically large. E.g. the expected number of blocks $\mathbb{E}[S]$ for subtrees of size $n = 2\alpha$ is

$$\begin{aligned}\mathbb{E}[S] &= 1 + \Pr[X_1 > 0] + \cdots + \Pr[X_{\alpha+1} > 0] \\ &= 1 + (\alpha+1)\left(1 - \binom{n-1}{\alpha-1} / \binom{n}{\alpha}\right) = 1 + (\alpha+1)(1 - \alpha/n) = 1 + (\alpha+1)/2 \quad .\end{aligned}$$

In this example, buffering stores the whole subtree using only one additional block.

4.1 Buffering for UR trees with load-factor $1 - \varepsilon$

Our top-down analysis of the expected size, and thus load-factors, of (α, ε)-RBSTs frequently uses the following counting and index exchange arguments.

▶ **Lemma 6** (Exchange). *We have $\Pr[X_i < c] = \Pr[X_j < c]$ for all $i, j \in \{1, \ldots, \alpha + 1\}$.*

Proof. Due to Observation 5, each solution to $X_1 + \cdots + X_{\alpha+1} = n - \alpha$ occurs with the same probability. Hence, we can calculate $\Pr[X_i \geq c]$ by counting the number of solutions where $X_i \geq c$ and dividing it by the total number of solutions.

Thus, $\Pr[X_i < c] = 1 - \Pr[X_i \geq c] = 1 - \binom{n-c}{\alpha} / \binom{n}{\alpha} = 1 - \Pr[X_j \geq c] = \Pr[X_j < c]$. ◀

Next, we show our size bound for an (α, ε)-RBST with n keys, where the priorities are from a uniform distribution over the permutations in $\mathrm{Perm}(n)$. Our analysis crucially relies on the basic fact that restricting $\mathrm{Perm}(n)$ on an arbitrary key subset of cardinality $n' < n$ yields the uniform distribution on $\mathrm{Perm}(n')$. Random variable S_n denotes the space, i.e. the

number of blocks used by the RBST on a set of n keys, F_n denotes the number of full, and $E_n := S_n - F_n$ the number of non-full blocks. Next we show a probability bound for the event that a given section, say the k-th, contains a number of keys X_k of a certain range.

▶ **Lemma 7.** *For any $1 \le k \le \alpha + 1$ and $i < j$, we have $\sum_{x=i}^{j} \Pr[X_k = x] = \frac{\binom{n-i}{\alpha} - \binom{n-1-j}{\alpha}}{\binom{n}{\alpha}}$.*

Proof. We have $\Pr[X_k \ge i] = \binom{n-i}{\alpha}/\binom{n}{\alpha}$ and $\Pr[X_k \ge j+1] = \binom{n-(j+1)}{\alpha}/\binom{n}{\alpha}$. ◀

These basic facts allow us to compute the following expressions.

▶ **Lemma 8.** *We have $\sum_{x=1}^{m} \Pr[X_k = x]x = \left(\binom{n}{\alpha+1} - \binom{n-m}{\alpha+1} - m\binom{n-1-m}{\alpha}\right)/\binom{n}{\alpha}$.*

Proof. By elementary computation, we have

$$\begin{aligned}\sum_{x=1}^{m} \Pr[X_k = x]x &= \sum_{x=1}^{m} \Pr[X_k = x] + \sum_{x=2}^{m} \Pr[X_k = x] + \ldots + \sum_{x=m}^{m} \Pr[X_k = x] \\ &= \frac{\binom{n-1}{\alpha} - \binom{n-1-m}{\alpha}}{\binom{n}{\alpha}} + \ldots + \frac{\binom{n-m}{\alpha} - \binom{n-1-m}{\alpha}}{\binom{n}{\alpha}} \\ &= \sum_{i=1}^{m} \frac{\binom{n-i}{\alpha}}{\binom{n}{\alpha}} - m\frac{\binom{n-1-m}{\alpha}}{\binom{n}{\alpha}} \ .\end{aligned} \tag{1}$$

Note that an RBST on $n' = n+1$ keys with block size $\alpha' = \alpha + 1$ has $\Pr[X'_1 = i] = \frac{\binom{n-i}{\alpha}}{\binom{n+1}{\alpha+1}}$. Thus, we have from Lemma 7 that

$$\begin{aligned}\sum_{i=1}^{m} \Pr[X'_1 = i] = \sum_{i=1}^{m} \frac{\binom{n-i}{\alpha}}{\binom{n+1}{\alpha+1}} &= \frac{\binom{(n+1)-1}{\alpha+1} - \binom{(n+1)-1-m}{\alpha+1}}{\binom{n+1}{\alpha+1}} \\ \Rightarrow \sum_{i=1}^{m} \binom{n-i}{\alpha} &= \binom{n}{\alpha+1} - \binom{n-m}{\alpha+1}\end{aligned} \tag{2}$$

and the lemma follows by using (2) for the last summation in (1). ◀

We are now ready to state our size bound.

▶ **Theorem 9** (Size). *The expected number of non-full blocks in an n-key (α, ε)-RBST is at most $\mathbb{E}[E_n] \le \max\{\varepsilon n/\alpha,\ 1\}$.*

Our proof is by induction on n, where the base cases are due to the number of non-full blocks that are occupied by the secondary buffer structures (Appendix A).

Proof. Observation 18 states that $E_n \le 1$ for all n with $\delta(n) \le 1$. Moreover, Theorem 21 shows that $\mathbb{E}[E_n] \le 27\delta(n)$ for $n \le \beta + \alpha = (\alpha+1)\rho + \alpha$. (Recall that $\beta = (\alpha+1)\rho$.) Next, we set ρ sufficiently large for the following proof. For simplicity, we will use t and γ instead of $\rho + \alpha$ and $(\alpha+1)\rho + \alpha$ respectively. We can conclude from Theorem 21 that for $t < n \le \gamma$,

$$\mathbb{E}[E_n] \le 27\left\lceil \frac{n-\alpha}{\rho} \right\rceil = 27\left\lceil \frac{n-\alpha}{108\alpha/\varepsilon} \right\rceil \le 27\left(\frac{n-\alpha}{108\alpha/\varepsilon} + 1\right) \le 27 \cdot 2\frac{n-\alpha}{108\alpha/\varepsilon} = \frac{n-\alpha}{2\alpha/\varepsilon} \ .$$

The last inequality holds because $n > \rho + \alpha = 108\alpha/\varepsilon + \alpha$ and $\frac{n-\alpha}{108\alpha/\varepsilon} > 1$.

For $n > \gamma$, we prove the theorem by induction. For each index $i \in \{1, \ldots, \alpha+1\}$, define Y_i as the number of non-full blocks in the i-th section beneath the root (of an RBST with n keys). Note that Y_i only depends on X_i, i.e. the number of keys in the i-th section, and their relative priorities. We thus have

$$\begin{aligned}
\mathbb{E}[Y_i] &= \sum_{x_i=1}^{n-\alpha} \Pr[X_i = x_i] \cdot \mathbb{E}[E_{x_i}] \\
&\leq \sum_{x_i=1}^{t} \Pr[X_i = x_i] \cdot 1 + \sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \frac{x_i}{2\alpha/\varepsilon} + \sum_{x_i=\gamma+1}^{n-\alpha} \Pr[X_i = x_i] \frac{x_i}{\alpha/\varepsilon} \\
&= \left(\sum_{x_i=1}^{t} \Pr[X_i = x_i] + \sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \right) - \sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \\
&+ \left(\sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \frac{x_i}{2\alpha/\varepsilon} - \sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \frac{x_i}{\alpha/\varepsilon} \right) - 2 \sum_{x_i=1}^{t} \Pr[X_i = x_i] \frac{x_i}{2\alpha/\varepsilon} \\
&+ \left(\sum_{x_i=1}^{t} \Pr[X_i = x_i] \frac{x_i}{\alpha/\varepsilon} + \sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \frac{x_i}{\alpha/\varepsilon} + \sum_{x_i=\gamma+1}^{n-\alpha} \Pr[X_i = x_i] \frac{x_i}{\alpha/\varepsilon} \right) \\
&= \left(\sum_{x_i=1}^{\gamma} \Pr[X_i = x_i] - \sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \frac{x_i}{2\alpha/\varepsilon} - \sum_{x_i=1}^{t} \Pr[X_i = x_i] \frac{x_i}{2\alpha/\varepsilon} \right) \\
&+ \sum_{x_i=1}^{n-\alpha} \Pr[X_i = x_i] \frac{x_i}{\alpha/\varepsilon} - \sum_{x_i=1}^{t} \Pr[X_i = x_i] \frac{x_i}{2\alpha/\varepsilon} - \sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \\
&\leq \sum_{x_i=1}^{\gamma} \Pr[X_i = x_i] \left(1 - \frac{x_i}{2\alpha/\varepsilon} \right) + \frac{\mathbb{E}[X_i]}{\alpha/\varepsilon}
\end{aligned}$$

Using Lemma 6, we have that the summation term has equal value for each section $i \in \{1, \ldots, \alpha+1\}$. Since, for $n \geq \alpha$, the root is full and not counted in E_n, we have

$$\mathbb{E}[E_n] = \mathbb{E}[Y_1 + \ldots + Y_{\alpha+1}] \leq (1+\alpha) \sum_{x_1=1}^{\gamma} \Pr[X_1 = x_1] \left(1 - \frac{x_1}{2\alpha/\varepsilon} \right) + \frac{n-\alpha}{\alpha/\varepsilon} \ . \tag{3}$$

To proof the inequality it suffices to show that the right-hand side of (3) is at most $\frac{\varepsilon n}{\alpha}$. This is true if and only if $(1+\alpha) \sum_{x_1=1}^{\gamma} \Pr[X_1 = x_1] \left(1 - \varepsilon \frac{x_1}{2\alpha}\right) \leq \varepsilon$. Since $\varepsilon > 0$ and $\alpha + 1 > 0$, it suffices to show that the value of the sum is not positive. This is if and only if

$$\sum_{x_1=1}^{\gamma} \Pr[X_1 = x_1] \leq \frac{\varepsilon}{2\alpha} \sum_{x_1=1}^{\gamma} \Pr[X_1 = x_1] x_1 \ . \tag{4}$$

Using Lemma 7 on the left-hand and Lemma 8 on the right-hand side, it remains to show

$$\frac{2\alpha}{\varepsilon} \binom{n-1}{\alpha} - \frac{2\alpha}{\varepsilon} \binom{n-1-\gamma}{\alpha} \leq \binom{n}{\alpha+1} - \binom{n-\gamma}{\alpha+1} - \gamma \binom{n-1-\gamma}{\alpha} \ . \tag{5}$$

By elementary computation, this inequality holds for all $n > \gamma$. (See Appendix B.) ◀

Since each full block contains α distinct keys, i.e. $F_n \leq n/\alpha$, we showed that $S_n \leq (1+\varepsilon)n/\alpha$. Thus, the load-factor, i.e. the relative utilization of the allocated blocks, has:

▶ **Corollary 10** (Load-Factor). *Let $\varepsilon \in (0, 1/2]$ and $n \geq \alpha + \beta$. The expected number of blocks in an n-key (α, ε)-RBST is at most $(1+\varepsilon)n/\alpha$, i.e. the expected load-factor is at least $1 - \varepsilon$.*

Proof. The load-factor is $L = n/\alpha S_n$. Since $\phi(x) = 1/x$ is a convex function, Jensen's inequality gives $1/\mathbb{E}[X] \leq \mathbb{E}[1/X]$. Using Thm. 9 yields $\mathbb{E}[L] \geq \frac{n}{\alpha}\frac{1}{(1+\varepsilon)n/\alpha} = \frac{1}{1+\varepsilon} \geq 1-\varepsilon$. ◀

Combining Lemma 2 and Theorem 9, we obtained the following for the range-search.

▶ **Corollary 11** (Range-Search). *Let $\varepsilon \in (0, 1/2]$. The expected number of block-reads for range reporting in (α, ε)-RBSTs is $\mathcal{O}(\varepsilon^{-1} + k/\alpha + \log_\alpha n)$, where k is the number of results.*

5 Bounds for Dynamic Updates via Partial-Rebuilding

The weight-proportional fan-out in the design of our secondary UR tree structure has the advantage that it avoids the need of additional restructuring work whenever a subtree must change its state between buffering and non-buffering, due to updates (cf. Section 2.1). Together with the space bound from last section and the observation that the partial-rebuild algorithm for updating RBSTs only rebuilds at most *three* subtrees beneath the affected node, regardless of its fan-out, allows us to show our bound on the expected structural change.

Using the counting technique from Lemma 2, it is easy to show that $\mathbb{E}|X(v)|$ is at most $\frac{2}{n}\sum_{i=1}^{n} \alpha\frac{n}{i} = \mathcal{O}(\alpha \ln n)$. The proof of the theorem will use the following, stronger bound.

▶ **Lemma 12** (Subtree Weight). *Let x be a key and random variable $|X(v)|$ denote the subtree weight of node v that stores $x \in B(v)$. Then $\sum_{N\geq\alpha+\beta}^{n} N \Pr[|X(v)| = N] = \mathcal{O}(\alpha \log_\alpha n)$.*

Proof. Let random variable $T \subseteq X$ be the largest set of consecutive keys that have $\pi(x) = \min_{x'\in T}\{\pi(x')\}$ and $x \in T$. Note that $T \subseteq X(v) \subseteq X$, and $T = X(v)$ iff $\pi(x) = p^*(v)$. Recall from binary Treaps that $\mathbb{E}|T| = \Theta(\ln n)$ for any $x \in X$, i.e. $H(n) \leq \mathbb{E}|T| \leq 2H(n/2)$ where $H(m) = \frac{1}{1} + \ldots + \frac{1}{m}$. We will show an upper bound of $\mathcal{O}(\frac{\alpha}{\log\alpha})\mathbb{E}|T|$ by arguing for the case $|X(v)| = N$. Indeed, if $\mathbb{E}\Big[|T|\Big||X(v)| = N\Big] = N\Omega(\frac{\log\alpha}{\alpha})$ holds for given $N \geq \alpha + \beta$, then

$$\sum_{N=\alpha+\beta}^{n} \Pr[|X(v)| = N]N = \sum_{N=\alpha+\beta}^{n} \Pr[|X(v)| = N]\mathbb{E}\Big[|T|\Big||X(v)| = N\Big]/\Omega(\log(\alpha)/\alpha)$$
$$= \mathcal{O}(\frac{\alpha}{\log\alpha}) \sum_{N=\alpha+\beta}^{n} \Pr[|X(v)| = N]\mathbb{E}\Big[|T|\Big||X(v)| = N\Big] \leq \mathcal{O}(\frac{\alpha}{\log\alpha})\mathbb{E}[T] \ .$$

Observe that x partitions the N keys in $X(v)$ into N^- keys smaller than x and N^+ keys larger than x, where $N^- + 1 + N^+ = N$. Though the priority orders of a set $X(v)$ of N keys are restricted to those that have $x \in B(v)$, we have that all relative orders of the keys in $B(v)$ remain equally likely. Consider the case that $B(v) = \{x_i^-, \ldots, x_1^-, x, x_1^+, \ldots, x_{\alpha-1-i}^+\}$ contains exactly i many separators smaller than x for some given integer $i \in [0, \alpha)$, i.e. there are $\alpha - 1 - i$ separators larger than x. Let the random variables $X_i^-, \ldots, X_0^- \subseteq X(v)$ denote the key sets of the respective sections smaller than x and $X_0^+, X_1^+, \ldots$ for the sections larger than x. That is, we have $\sum_j 1 + |X_j^-| = N^- + 1$ and $\sum_j 1 + |X_j^+| = N^+ + 1$ regardless of the priority orders. We consider the relative π-orders of those $i+1$ separators $\{x_i^-, \ldots, x_1^-\} \cup \{x\}$ in $B(v)$ and use the fact (regarding ancestors in binary Treaps) that the entire keys of the j-th section are in $X_j^- \subseteq T$ if $\pi(x) = \min\{\pi(x), \pi(x_j^-), \ldots, \pi(x_1^-)\}$, which has probability $\frac{1}{j+1}$. Now, using that $\mathbb{E}\Big[|X_j^-|\Big|N, N^-, i\Big] = \mathbb{E}\Big[|X_0^-|\Big|N, N^-, i\Big] = \frac{N^- - i}{i+1} \geq 0$ for all j, we have

$$\mathbb{E}\Big[|T|\Big|N,N^-,i\Big]-1=\mathbb{E}[T^-|N,N^-,i]+\mathbb{E}[T^+|N,N^-,i]$$
$$=\sum_{j=0}^{i}\frac{1}{i+1}\left(1+\mathbb{E}\Big[|X_j^-|\Big|N,N^-,i\Big]\right)+\sum_{j=0}^{\alpha-i}\frac{1}{\alpha-i}\left(1+\mathbb{E}\Big[|X_j^+|\Big|N,N^-,i\Big]\right)$$
$$=\frac{N^-+1}{i+1}\sum_{j=0}^{i}\frac{1}{j+1}+\frac{N^++1}{\alpha-i}\sum_{j=0}^{\alpha-i-1}\frac{1}{j+1}=(N^-+1)\frac{H(i+1)}{i+1}+(N^++1)\frac{H(\alpha-i)}{\alpha-i}.$$

Since $\frac{H(i+1)}{i+1}\geq\frac{H(\alpha)}{\alpha}$, we have that $\mathbb{E}\Big[|T|\Big|N,\ i\Big]=N\Omega(\frac{\ln\alpha}{\alpha})$, thus $\mathbb{E}\Big[|T|\Big|N\Big]=N\Omega(\frac{\ln\alpha}{\alpha})$. ◀

▶ **Theorem 13.** *The expected number of writes (Os) of an update in an n-key (α,ε)-RBST is $\mathcal{O}(\varepsilon^{-1}+\log_\alpha(n)/\alpha)$. Updates have $\mathcal{O}(\varepsilon^{-2}+(1+\frac{\varepsilon^{-1}+\log n}{\alpha})\log_\alpha n)$ expected reads (Is).*

Recall that our algorithm reads blocks only from one search path and during the `top`-queries. To bound $\mathbb{E}[m+m']$ and $\mathbb{E}[D'\cdot m]$ from Obs. 1, we first analyze the writes (Os). This extends to the reads (Is), using a high-probability bound for depth (e.g. Lemma 2 or Lemma 3 and 4), since both random variables are in the worst-case $\mathcal{O}(n)$.

Proof. We show our bound on the expected total structural change for deleting a key x from the RBST. (Insertion of the key would count the same number of blocks.) Let $X'=X\cup\{x\}$ be the set of keys before and after the deletion of x respectively, where $|X'|=n$. Let v be the subtree root that is subject to the partial rebuild algorithm from Section 2.1. From Lemma 2, determining the search path of v takes expected $\mathcal{O}(\varepsilon^{-1}+\log_\alpha n)$ reads (Is). Either v is a primary node with $\delta_v=\alpha+1$ or v is a buffering node with $\delta_v\leq\alpha+1$.

For $\delta_v\leq 1$, we have in the worst-case at most $\mathcal{O}(1/\varepsilon)$ blocks in the buffer's list.

For a primary node with $\delta_v=\alpha+1$, v contains x in its block $B(v)$. From Lemma 12, we have that $\sum_{N\geq\alpha+\beta}^{n}N\Pr[|X(v)|=N]=\mathcal{O}(\alpha\log_\alpha n)$. Now, for any given integer N of keys in the subtree, we have from Theorem 9 that the expected number of blocks is $\mathcal{O}(N/\alpha)$ and, from linearity of expectation, that three of the $\alpha+1$ subtrees contain at most $\mathcal{O}(N/\alpha^2)$ blocks. Thus, the contribution towards the expected number of block-writes is at most $\sum_{N\geq\alpha+\beta}\Pr[|X(v)|=N]\ N\cdot\mathcal{O}(1/\alpha^2)=\mathcal{O}(\log_\alpha(n)/\alpha)$. Moreover, using the high-probability bound of Lemma 2, we have that the depth of the subtree of v is less than $\mathcal{O}(\varepsilon^{-1}+\ln n)$ for any given integer $N=|X(v)|\leq n$. Thus, the contribution towards the total number of block-reads, issued by the `top`-queries of the update operation (cf. Observation 1), is at most $\sum_{N\geq\alpha+\beta}\Pr[|X(v)|=N]\ N\cdot(\varepsilon^{-1}+\ln n)\mathcal{O}(1/\alpha^2)=\mathcal{O}(\varepsilon^{-1}\log_\alpha(n)/\alpha+\frac{\ln n}{\alpha}\log_\alpha n)$.

For a buffering node with $\delta_v\geq 2$, v has $|X(v)|=\Theta(\alpha\delta_v/\varepsilon)$. By Theorem 21, subtree rooted at v has expected $\mathcal{O}(\delta_v/\varepsilon)$ blocks and writing three, of its δ_v subtrees, has $\mathbb{E}[m]=\mathcal{O}(\varepsilon^{-1})$ expected block-writes. For the block-reads, consider the number of keys in some active section X_i of v. If $X_i=\Theta(c\alpha/\varepsilon)$ for some $c\geq 2$, then the worst-case depth of this child is $\mathcal{O}(c/\varepsilon)$. By Lemma 14, the probability is at most $\mathcal{O}(e^{-c})$. Thus, $\sum_{c=2}^{\delta_v}\Pr[\delta(X_i)=c]\mathcal{O}(c/\varepsilon)\mathbb{E}[\ m\mid\delta(X_i)=c\ ]=\sum_{c=2}^{\delta_v}\frac{c^2}{e^c}\mathcal{O}(1/\varepsilon^2)=\mathcal{O}(\varepsilon^{-2})\sum_{c>1}c^2/e^c$ shows that the expected block-reads are $\mathcal{O}(\varepsilon^{-2})$ for either of the three affected subtrees of v. ◀

References

1 Arne Andersson and Thomas Ottmann. New Tight Bounds on Uniquely Represented Dictionaries. *SIAM J. Comput.*, 24(5):1091–1101, 1995. doi:10.1137/S0097539792241102.

2 Lars Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003. doi:10.1007/s00453-003-1021-x.

3 Ricardo A. Baeza-Yates. The expected behaviour of b+-trees. *Acta Informatica*, 26(5):439–471, 1989. doi:10.1007/BF00289146.

4 Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972. doi:10.1007/BF00288683.

5 Naama Ben-David, Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, Charles McGuffey, and Julian Shun. Parallel algorithms for asymmetric read-write costs. In *Proc. Symposium on Parallelism in Algorithms and Architectures (SPAA'16)*, pages 145–156, 2016. doi:10.1145/2935764.2935767.

6 Michael A. Bender, Jonathan W. Berry, Rob Johnson, Thomas M. Kroeger, Samuel McCauley, Cynthia A. Phillips, Bertrand Simon, Shikha Singh, and David Zage. Anti-persistence on persistent storage: History-independent sparse tables and dictionaries. In *Proc. 35th Symposium on Principles of Database Systems (PODS'16)*, pages 289–302. ACM, 2016. doi:10.1145/2902251.2902276.

7 Michael A. Bender, Martin Farach-Colton, Jeremy T. Fineman, Yonatan R. Fogel, Bradley C. Kuszmaul, and Jelani Nelson. Cache-oblivious streaming b-trees. In *Proc. Symposium on Parallelism in Algorithms and Architectures (SPAA'07)*, pages 81–92. ACM, 2007. doi:10.1145/1248377.1248393.

8 Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, and Julian Shun. Sorting with asymmetric read and write costs. In *Proc. Symposium on Parallelism in Algorithms and Architectures (SPAA'15)*, pages 1–12, 2015. doi:10.1145/2755573.2755604.

9 Guy E. Blelloch and Daniel Golovin. Strongly History-Independent Hashing with Applications. In *Proc. 48th Symposium on Foundations of Computer Science (FOCS'07)*, 2007. doi:10.1109/FOCS.2007.36.

10 Guy E. Blelloch, Daniel Golovin, and Virginia Vassilevska. Uniquely Represented Data Structures for Computational Geometry. In Joachim Gudmundsson, editor, *Proc. 11th Scandinavian Workshop on Algorithm Theory (SWAT'08)*, pages 17–28. Springer, 2008. doi:10.1007/978-3-540-69903-3_4.

11 Guy E. Blelloch and Margaret Reid-Miller. Fast set operations using treaps. In *Proc. 10th Symposium on Parallel Algorithms and Architectures (SPAA'98)*, pages 16–26, 1998. doi:10.1145/277651.277660.

12 Milutin Brankovic, Nikola Grujic, André van Renssen, and Martin P. Seybold. A simple dynamization of trapezoidal point location in planar subdivisions. In *Proc. 47th International Colloquium on Automata, Languages, and Programming (ICALP'20)*, pages 18:1–18:18, 2020. doi:10.4230/LIPIcs.ICALP.2020.18.

13 Gerth Stølting Brodal and Rolf Fagerberg. Lower bounds for external memory dictionaries. In *Proc. 14th Symposium on Discrete Algorithms (SODA'03)*, pages 546–554, 2003. URL: https://dl.acm.org/doi/10.5555/644108.644201.

14 Douglas Comer. The ubiquitous b-tree. *ACM Comput. Surv.*, 11(2):121–137, 1979. doi:10.1145/356770.356776.

15 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications, 3rd Edition*. Springer, 2008. doi:10.1007/978-3-540-77974-2.

16 James R. Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, 1989. doi:10.1016/0022-0000(89)90034-2.

17 Amr Elmasry, Mostafa Kahla, Fady Ahdy, and Mahmoud Hashem. Red-black trees with constant update time. *Acta Informatica*, 56(5):391–404, 2019. doi:10.1007/s00236-019-00335-9.

18 Eric Demaine et al. Lecture notes on persistent data structures. https://web.archive.org/web/20230601205149/http://courses.csail.mit.edu/6.851/spring21/scribe/lec1.pdf.

19 Daniel Golovin. *Uniquely represented data structures with applications to privacy*. PhD thesis, Carnegie Mellon University, 2008.

20 Daniel Golovin. B-Treaps: A Uniquely Represented Alternative to B-Trees. In *Proc. 36th International Colloquium on Automata, Languages, and Programming (ICALP'09)*, pages 487–499, 2009. doi:10.1007/978-3-642-02927-1_41.

21 Daniel Golovin. The b-skip-list: A simpler uniquely represented alternative to b-trees. *CoRR*, abs/1005.0662, 2010. arXiv:1005.0662.

22 Dana Jansens. Persistent binary search trees. https://web.archive.org/web/20230526175144/https://cglab.ca/~dana/pbst/, 2014. Accessed: 2023-05-26.

23 Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.

24 Klaus Küspert. Storage Utilization in B*-Trees with a Generalized Overflow Technique. *Acta Informatica*, 19:35–55, 1983. doi:10.1007/BF00263927.

25 Ketan Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, 1994.

26 Moni Naor and Vanessa Teague. Anti-presistence: history independent data structures. In *Proc. 33rd Symposium on Theory of Computing (STOC'01)*, pages 492–501, 2001. doi:10.1145/380752.380844.

27 Raimund Seidel and Cecilia R. Aragon. Randomized Search Trees. *Algorithmica*, 16(4/5):464–497, 1996. doi:10.1007/BF01940876.

28 Lawrence Snyder. On uniquely represented data strauctures. In *Proc. 18th Symposium on Foundations of Computer Science (SFCS'77)*, pages 142–146, 1977. doi:10.1109/SFCS.1977.22.

29 Rajamani Sundar and Robert Endre Tarjan. Unique Binary-Search-Tree Representations and Equality Testing of Sets and Sequences. *SIAM J. Comput.*, 23(1):24–44, 1994. doi:10.1137/S0097539790189733.

30 Jeffrey Scott Vitter. External memory algorithms and data structures. *ACM Comput. Surv.*, 33(2):209–271, 2001. doi:10.1145/384192.384193.

31 Jean Vuillemin. A unifying look at data structures. *Commun. ACM*, 23(4):229–239, 1980. doi:10.1145/358841.358852.

32 Andrew Chi-Chih Yao. On Random 2-3 Trees. *Acta Informatica*, 9:159–170, 1978. doi:10.1007/BF00289075.

A Induction Base Case: Expected Size of Buffering Trees

▶ **Lemma 14.** *Let $X_1, \ldots, X_m$ be some non-negative integral random variables where $X_1 + X_2 + \cdots + X_m = n$. For integers $n \geq 1$ and $t \leq n$, we have the tail-bounds*

$$\left(1 - \frac{t}{n+1}\right)^{m-1} \leq \Pr[X_i \geq t] \leq \left(1 - \frac{t}{n+m-1}\right)^{m-1}.$$

Proof.

$$\Pr[X_i \geq t] = \frac{\binom{n-t+m-1}{m-1}}{\binom{n+m-1}{m-1}} = \frac{n-t+m-1}{n+m-1} \cdot \frac{n-t+m-2}{n+m-2} \cdots \frac{n-t+m-(m-1)}{n+m-(m-1)}$$
$$= \left(1 - \frac{t}{n+m-1}\right)\left(1 - \frac{t}{n+m-2}\right)\cdots\left(1 - \frac{t}{n+m-(m-1)}\right)$$

Since for $1 \leq i \leq m-1$, $\frac{t}{n+m-1} \leq \frac{t}{n+m-i} \leq \frac{t}{n+m-(m-1)} = \frac{t}{n+1}$, the lemma holds. ◀

▶ **Lemma 15.** *For $0 < c \leq a \leq b$, we have $\frac{a-c}{b-c} \leq \frac{a}{b}$.*

Proof. $\frac{a-c}{b-c} \leq \frac{a}{b} \iff ab - bc \leq ab - ac \iff a \leq b.$ ◀

For a subtree of size $n \leq \beta + \alpha =: (\alpha+1)\rho + \alpha$, fan-out parameter $\delta(n) = \min\{\alpha + 1, \max\{1, \lceil \frac{n-\alpha}{\rho} \rceil\}\}$ is equal to $\max\{1, \lceil \frac{n-\alpha}{\rho} \rceil\}$. Therefore, for n with $2 \leq \delta(n) \leq \alpha + 1$, we have $(\delta(n)-1)\rho + \alpha < n \leq \delta(n)\rho + \alpha$. Moreover, for $k \in \{2, 3, \ldots, \delta_n\}$, $n > (k-1)\rho + \alpha$ if and only if $\delta(n) \geq k$.

▶ **Lemma 16.** *For a subtree of size $n \leq \beta+\alpha$, where $\delta(n) \leq \alpha+1$, and all $k \in \{2, 3, \ldots, \delta(n)\}$, we have*

$$\Pr[\delta(X_i) \geq k] = \Pr[X_i > (k-1)\rho + \alpha] \leq \left(1 - \frac{k-1}{\delta(n)}\right)^{\delta(n)-1},$$

where X_i is the number of keys in the i-th section beneath the root.

Proof.

$$\Pr[X_i > (k-1)\rho + \alpha] = \Pr[X_i \geq (k-1)\rho + \alpha + 1]$$

Set variables m, t, and n of Lemma 14 to $\delta(n)$, $(k-1)\rho + \alpha + 1$, and $n - \alpha$ respectively. We have

$$\Pr[X_i \geq (k-1)\rho + \alpha + 1] \leq \left(1 - \frac{(k-1)\rho + \alpha + 1}{n - \alpha + \delta(n) - 1}\right)^{\delta(n)-1}.$$

Since $n \leq \delta(n)\rho + \alpha$, we have

$$\Pr[X_i > (k-1)\rho + \alpha] \leq \left(1 - \frac{(k-1)\rho + \alpha + 1}{\delta(n)\rho + \alpha - \alpha + \delta(n) - 1}\right)^{\delta(n)-1}$$

Using Lemma 15, we subtract $\delta(n) - 1$ from the numerator and denominator of $\frac{(k-1)\rho+\alpha+1}{\delta(n)\rho+\alpha-\alpha+\delta(n)-1}$. Thus

$$\Pr[X_i > (k-1)\rho + \alpha] \leq \left(1 - \frac{(k-1)\rho + \alpha + 1 - (\delta(n)-1)}{\delta(n)\rho}\right)^{\delta(n)-1}.$$

Since $\delta(n)$ is at most $\alpha + 1$, we have $\alpha + 1 - (\delta(n) - 1)$ is positive and conclude

$$\Pr[X_i > (k-1)\rho + \alpha] \leq \left(1 - \frac{(k-1)\rho}{\delta(n)\rho}\right)^{\delta(n)-1} = \left(1 - \frac{k-1}{\delta(n)}\right)^{\delta(n)-1},$$

as stated. ◀

We will combine the results of Lemmas 16 and 15 and get the following result.

▶ **Corollary 17.** *For $2 \leq k \leq \delta(n)$, we have*

$$\Pr[\delta(X_i) \geq k] = \Pr[X_i > (k-1)\rho + \alpha] \leq \left(1 - \frac{k-1-1}{\delta(n)-1}\right)^{\delta(n)-1} \leq e^{-(k-2)}.$$

▶ **Observation 18.** *For a subtree with $\delta(n) = 1$, the data structure is a simple list and has at most one non-full block.*

▶ **Lemma 19.** *For a subtree with $\delta(n) = 2$, the expected number of non-full blocks $\mathbb{E}[E_n] \leq 5$.*

Proof. We will design an algorithm calculating an upper bound on the number of non-full blocks. The root has α keys, and the remaining keys are randomly split into two sections with X_1 and X_2 keys, i.e. $X_1+X_2 = n-\alpha$. For $i \in \{1,2\}$, if $\delta(X_i) = 1$, there is at most one non-full block, and the algorithm does not need to proceed in this section anymore. If $\delta(X_i) = 2$, to observe non-full blocks, the $X_i - \alpha$ keys should be partitioned again. It is impossible for both X_1 and X_2 to have $\delta(X_i) = 2$ since it implies that $n = \alpha + X_1 + X_2 > \alpha + 2(\rho+\alpha) = 2\rho + 3\alpha$, which is a contradiction. So at each iteration, either the algorithm stops or continues with one of the sections. In the last step, there are two sections each with at most one non-full block. Thus, the expected number of iterations plus 1 is an upper bound on $\mathbb{E}[E_n]$.

Clearly, $\mathbb{E}[E_n]$ is indeed an increasing function of n, so losing α keys of the root at each iteration results in fewer non-full blocks. To obtain a weaker upper bound, we assume that at each step, the key with the highest priority splits the $n-1$ remaining keys, instead of $n-\alpha$ keys, into two new sections. These keys include the $\alpha - 1$ other keys of the root as well. It suffices to show that the expected number of iterations is at most 4.

At iteration k, k keys with the highest priorities split n keys into k sections with $Y_1^{(k)}, Y_2^{(k)}, \ldots, Y_k^{(k)}$ keys, where $Y_1^{(k)} + Y_2^{(k)} + \cdots + Y_k^{(k)} = n - k$. Round $k+1$ occurs if any of $Y_i^{(k)}$s exceeds $\rho + \alpha$. Since k top keys are chosen uniformly at random, all solutions to the equation $Y_1^{(k)} + Y_2^{(k)} + \cdots + Y_k^{(k)} = n - k$ have equal probabilities. Next, we find an upper bound on $\Pr[Y_i^{(k)} > \rho + \alpha]$ for $i \in \{1, \ldots, k\}$. Set parameters m, t, and n of Lemma 14 to k, $\rho + \alpha$, and $n - k$ respectively. Thus

$$\Pr[Y_i^{(k)} > \rho + \alpha] \leq \left(1 - \frac{\rho + \alpha + 1}{n - k + k - 1}\right)^{k-1} \leq \left(1 - \frac{\rho + \alpha - 1}{n - 1}\right)^{k-1} .$$

Since n is at most $2\rho + \alpha$, we have

$$\Pr[Y_i^{(k)} > \rho + \alpha] \leq \left(1 - \frac{\rho + \alpha - 1}{2\rho + \alpha - 1}\right)^{k-1} \leq \left(1 - \frac{\rho}{2\rho}\right)^{k-1} = 2^{-(k-1)} .$$

The last inequality is an application of Lemma 15. Using the union bound, we have

$$\Pr[\exists Y_i^{(k)} > \rho + \alpha] \leq k/2^{k-1} .$$

This is indeed the probability of round k+1 occurring, conditioned on the probability of rounds k having occurred. Define Z to be the number of iterations and Z_k to be an indicator random variable of whether the k-th iteration happened. If

$$\begin{aligned}\Pr[Z_k = 1] &= \Pr[Z_{k-1} = 1] \Pr[Z_k = 1 | Z_{k-1} = 1] + \Pr[Z_{k-1} = 0] \Pr[Z_k = 1 | Z_{k-1} = 0] \\ &= \Pr[Z_{k-1} = 1] \Pr[Z_k = 1 | Z_{k-1} = 1] \leq k/2^{k-1}\end{aligned}$$

It follows that

$$\begin{aligned}\mathbb{E}[Z] &= \sum_{k=1}^{\infty} \mathbb{E}[Z_k] = \sum_{k=1}^{\infty} \Pr[Z_k = 1] \\ &\leq \sum_{k=1}^{\infty} k/2^{k-1} = 2(\sum_{j=1}^{\infty} \sum_{k=j}^{\infty} 2^{-k}) = 2(\sum_{j=1}^{\infty} \frac{2^{-j}}{1 - \frac{1}{2}}) = 2\frac{\frac{2^{-1}}{1-1/2}}{1/2} = 4 .\end{aligned}$$

◀

▶ **Lemma 20.** *For a subtree with $\delta(n) = 3$, the expected number of non-full blocks $\mathbb{E}[E_n] \leq 10$.*

Proof. $n-\alpha$ keys beneath the root are separated into three sections with X_1, X_2, and X_3 keys. Same as Lemma 19, we can prove that it is impossible to have $\delta(X_i) \geq 2$ for every $i \in \{1,2,3\}$, or $\delta(X_i) = \delta(X_j) = 3$ for some $i \neq j \in \{1,2,3\}$. Due to the symmetric property of the random variables X_i, we can conclude

$$\begin{aligned}1 = \;& 3\Pr[\delta(X_1)=3 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1] \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=2 \wedge \delta(X_3)=1] \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1] \\ & + \Pr[\delta(X_1)=1 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1]\ .\end{aligned}$$

Next, we perform an induction on $2\rho+\alpha < n \leq 3\rho+\alpha$. Using Observation 18, Lemma 19, and induction hypothesis, we get

$$\begin{aligned}\mathbb{E}[E_n] \leq \;& 3\Pr[\delta(X_1)=3 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1](10+1+1) \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=2 \wedge \delta(X_3)=1](5+5+1) \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1](5+1+1) \\ & + \Pr[\delta(X_1)=1 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1](1+1+1) \\ < \;& 3\Pr[\delta(X_1)=3 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1](12-7) \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=2 \wedge \delta(X_3)=1](11-7) \\ & + 7(3\Pr[\delta(X_1)=3 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1] \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=2 \wedge \delta(X_3)=1] \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1] \\ & + \Pr[\delta(X_1)=1 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1]) \\ = \;& 3\Pr[\delta(X_1)=3 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1]5 \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=2 \wedge \delta(X_3)=1]4 + 7 \\ \leq \;& 7 + 3\Pr[\delta(X_1)\geq 3]5 + 3\Pr[\delta(X_1)\geq 2 \wedge \delta(X_2)\geq 2]4\ .\end{aligned}$$

Lemma 16 implies that $\Pr[\delta(X_1) \geq 3] \leq \frac{1}{9}$. (Set $\delta(n)=3$ and $k=3$.) To complete the proof we show that $\Pr[\delta(X_1)\geq 2 \wedge \delta(X_2)\geq 2] \leq \frac{1}{9}$. Consequently, we get the desired inequality $\mathbb{E}[E_n] \leq 7 + 3\frac{5}{9} + 3\frac{4}{9} = 10$.

The two top-priority keys split the keys into three parts with Y_1, Y_2, and Y_3 keys, where $Y_1+Y_2+Y_3 = n-2$. The i-th part includes all X_i keys beneath the root and some extra keys from the root, so for each $i \in \{1,2,3\}$, we have $X_i \leq Y_i$.

$$\begin{aligned}\Pr[\delta(X_1)\geq 2 \wedge \delta(X_2)\geq 2] &= \Pr[X_1 \geq \rho+\alpha+1 \wedge X_2 \geq \rho+\alpha+1] \\ &\leq \Pr[Y_1 \geq \rho+\alpha+1 \wedge Y_2 \geq \rho+\alpha+1] = \binom{n-2(\rho+\alpha+1)}{2} \Big/ \binom{n}{2}\end{aligned}$$

Same technique of Lemma 14 yields $\binom{n-2(\rho+\alpha+1)}{2} \Big/ \binom{n}{2} \leq \left(1 - \frac{2(\rho+\alpha+1)}{n}\right)^2$.

Combining the previous inequalities together with $n \leq 3\rho+\alpha$ implies

$$\Pr[\delta(X_1)\geq 2 \wedge \delta(X_2)\geq 2] \leq \left(1 - \frac{2(\rho+\alpha+1)}{3\rho+\alpha}\right)^2\ .$$

Subtract α from the numerator and denominator. By Lemma 15, we have:

$$\Pr[\delta(X_1)\geq 2 \wedge \delta(X_2)\geq 2] \leq \left(1 - \frac{2\rho+\alpha+2}{3\rho}\right)^2 \leq \left(1 - \frac{2\rho}{3\rho}\right)^2 = \frac{1}{9}\ ,$$

as required. ◀

▶ **Theorem 21.** *For a subtree of size $n \leq \beta + \alpha$, we have that the expected number of non-full blocks $\mathbb{E}[E_n] \leq 27 \cdot \delta(n)$.*

Proof. We will prove the statement by induction on n. Observation 18, Lemma 19, and Lemma 20 provide the bases of the induction. The problem is divided into subproblems by using random variables $X_1, \ldots, X_{\delta(n)}$.

$$\mathbb{E}[E_n] = \mathbb{E}[E_{X_1}] + \mathbb{E}[E_{X_2}] + \cdots + \mathbb{E}[E_{X_{\delta(n)}}] = \delta(n)\mathbb{E}[E_{X_1}]$$

Note that $X_i \leq n - \alpha$, so the induction assumption is applicable to it. For $n \geq 4\rho + \alpha$, we will use the induction assumption and bases to obtain

$$\begin{aligned}
\mathbb{E}[E_{X_i}] &\leq \Pr[\delta(X_i) = 1] \cdot 1 + \Pr[\delta(X_i) = 2] \cdot 5 + \Pr[\delta(X_i) = 3] \cdot 10 + \sum_{k=4}^{\delta(n)} \Pr[\delta(X_i) = k] \cdot (27 \cdot k) \\
&< \Pr[\delta(X_i) = 1] \cdot 10 + \Pr[\delta(X_i) = 2] \cdot 10 + \Pr[\delta(X_i) = 3] \cdot 10 + \sum_{k=4}^{\delta(n)} \Pr[\delta(X_i) = k] \cdot (27 \cdot k) \\
&= 10 \Pr[1 \leq \delta(X_i) \leq 3] + 27 \sum_{k=4}^{\delta(n)} \Pr[\delta(X_i) = k] \cdot k \leq 10 + 27 \sum_{k=4}^{\delta(n)} \Pr[\delta(X_i) = k] \cdot k
\end{aligned}$$

We can rewrite $\sum_{k=4}^{\delta(n)} \Pr[\delta(X_i) = k] \cdot k$ to $3 \Pr[\delta(X_i) \geq 4] + \sum_{k=4}^{\delta(n)} \Pr[\delta(X_i) \geq k]$. Then by using Corollary 17, we get

$$\begin{aligned}
\mathbb{E}[E_{X_i}] &\leq 10 + 27(3 \Pr[\delta(X_i) \geq 4] + \sum_{k=4}^{\delta(n)} \Pr[\delta(X_i) \geq k]) \leq 10 + 27 \left(3e^{-2} + \sum_{k=4}^{\infty} e^{-(k-2)} \right) \\
&\leq 10 + 27 \left(\frac{3}{e^2} + \sum_{k=2}^{\infty} e^{-k} \right) = 10 + 27 \left(\frac{3}{e^2} + \frac{e^{-2}}{1 - e^{-1}} \right) \leq 10 + 0.6202 \times 27 < 27 \,,
\end{aligned}$$

as stated. ◀

B Additional Proofs

▶ **Lemma 22.** *Consider a subtree with $n > \gamma$ keys, where γ is equal to $\beta + \alpha = (\alpha + 1)\rho + \alpha$. We have:*

$$2\frac{\alpha}{\varepsilon}\binom{n-1}{\alpha-i} - 2\frac{\alpha}{\varepsilon}\binom{n-1-\gamma}{\alpha-i} \leq \binom{n}{\alpha+1-i} - \binom{n-\gamma}{\alpha+1-i} - \gamma\binom{n-1-\gamma}{\alpha-i} \,.$$

Proof. The proof is by induction on n. For $n = \gamma + 1$ the statement is

$$\begin{aligned}
& 2\frac{\alpha}{\varepsilon}\binom{\gamma}{\alpha-i} \leq \binom{\gamma+1}{\alpha+1-i} \\
\iff \quad & 2\frac{\alpha}{\varepsilon}\frac{\gamma(\gamma-1)\ldots(\gamma-\alpha+i+1)}{(\alpha-i)!} \leq \frac{(\gamma+1)\gamma\ldots(\gamma-\alpha+i+1)}{(\alpha-i+1)(\alpha-i)!} \qquad (6) \\
\iff \quad & 2\frac{\alpha}{\varepsilon}(\alpha-i+1) \leq \gamma+1 \quad ,
\end{aligned}$$

This is true because $\gamma = (\alpha+1)\rho + \alpha = 108(\alpha+1)\alpha/\varepsilon + \alpha > 2(\alpha+1)\alpha/\varepsilon$.

Assume the lemma holds for each $\gamma < m \leq n$. We will prove the lemma for $n+1$, i.e. we will show:

$$2\frac{\alpha}{\varepsilon}\binom{n}{\alpha-i} - 2\frac{\alpha}{\varepsilon}\binom{n+1-1-\gamma}{\alpha-i} \leq \binom{n+1}{\alpha+1-i} - \binom{n+1-\gamma}{\alpha+1-i} - \gamma\binom{n+1-1-\gamma}{\alpha-i}$$

holds for all i. The inequality is true for $i = \alpha - 1$, since

$$\begin{aligned} 2\frac{\alpha}{\varepsilon}n - 2\frac{\alpha}{\varepsilon}(n-\gamma) &\leq \frac{n(n+1)}{2} - \frac{(n-\gamma)(n-\gamma+1)}{2} - \gamma(n-\gamma) \\ \iff \quad 2\frac{\alpha}{\varepsilon}\gamma &\leq \frac{2\gamma n - \gamma^2 + \gamma}{2} - \frac{2\gamma n - 2\gamma^2}{2} \\ \iff \quad 2\frac{\alpha}{\varepsilon}\gamma &\leq \frac{\gamma^2+\gamma}{2} \quad . \end{aligned}$$

Hence, it suffices to observe that

$$2\frac{\alpha}{\varepsilon}\gamma \leq \frac{\gamma^2+\gamma}{2} \quad \iff \quad 4\frac{\alpha}{\varepsilon} \leq \gamma + 1 = (\alpha+1)\rho + \alpha + 1 = (\alpha+1)108\alpha/\varepsilon + \alpha + 1$$

which is true.

For $i < \alpha - 1$, we will use the identity equation $\binom{m}{k} = \binom{m-1}{k} + \binom{m-1}{k-1}$ together with the induction assumption. We have

$$\begin{aligned} &2\frac{\alpha}{\varepsilon}\binom{n}{\alpha-i} - 2\frac{\alpha}{\varepsilon}\binom{n-\gamma}{\alpha-i} \\ &= 2\frac{\alpha}{\varepsilon}\binom{n-1}{\alpha-(i+1)} + 2\frac{\alpha}{\varepsilon}\binom{n-1}{\alpha-i} - 2\frac{\alpha}{\varepsilon}\binom{n-1-\gamma}{\alpha-(i+1)} - 2\frac{\alpha}{\varepsilon}\binom{n-1-\gamma}{\alpha-i} \\ &= \left[2\frac{\alpha}{\varepsilon}\binom{n-1}{\alpha-(i+1)} - 2\frac{\alpha}{\varepsilon}\binom{n-1-\gamma}{\alpha-(i+1)}\right] + \left[2\frac{\alpha}{\varepsilon}\binom{n-1}{\alpha-i} - 2\frac{\alpha}{\varepsilon}\binom{n-1-\gamma}{\alpha-i}\right] \\ &\leq \binom{n}{\alpha+1-(i+1)} - \binom{n-\gamma}{\alpha+1-(i+1)} - \gamma\binom{n-1-\gamma}{\alpha-(i+1)} \\ &+ \binom{n}{\alpha+1-i} - \binom{n-\gamma}{\alpha+1-i} - \gamma\binom{n-1-\gamma}{\alpha-i} \\ &= \binom{n+1}{\alpha+1-i} - \binom{n+1-\gamma}{\alpha+1-i} - \gamma\binom{n-\gamma}{\alpha-i} \quad . \end{aligned}$$

◀

Farthest-Point Voronoi Diagrams in the Hilbert Metric

Minju Song
Graduate School of Artificial Intelligence, Pohang University of Science and Technology, Republic of Korea

Mook Kwon Jung
Department of Computer Science and Engineering, Pohang University of Science and Technology, Republic of Korea

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

Abstract

The Hilbert metric, introduced by David Hilbert in 1895, is a projective metric defined on a bounded convex domain in a Euclidean space. For a convex polygon with m vertices and n point sites lying inside the polygon in the plane, it is shown that the nearest-point Voronoi diagram in the Hilbert metric has combinatorial complexity of $O(mn)$ [Gezalyan and Mount, SoCG 2023]. In this paper, we show that the farthest-point Voronoi diagram in the Hilbert metric has combinatorial complexity $O(m)$, which is independent of the number of sites. Also, we present an efficient algorithm to compute the farthest-point Voronoi diagram.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Farthest-point Voronoi diagram, Hilbert metric, Complexity, Algorithm

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.48

Funding This research was supported by 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)). This research was partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2023-00219980).

Acknowledgements For helpful comments we thank Auguste H. Gezalyan, David M. Mount and the anonymous reviewers.

1 Introduction

The Hilbert metric, introduced by David Hilbert in 1895 [16], defines a distance function between points in the interior of any convex body in d-dimensional space. Hilbert geometry provides a generalized framework that applies to any convex body, transcending the limitations of Euclidean space [21]. The significance of Hilbert metric is highlighted by a critical role in convex approximation, which is widely used in applications such as approximate nearest neighbor searches in the Euclidean metric and other metric spaces [1, 8], optimal construction of ϵ-kernels [6], approximate closest vectors [12, 13, 19, 23], and approximations of polytopes with low combinatorial complexity [3, 5, 7]. Various elements used in these approximations, such as Macbeath regions and Dikin ellipsoids, act similarly to Hilbert balls [2].

Despite its potential, not much is known about construction algorithms for structures in Hilbert geometry. Nielsen and Shao characterized balls in the Hilbert metric [20]. Bumpus et al. investigated the properties of balls and bisectors in the Hilbert metric [9]. A Voronoi diagram is one of the most fundamental structures in understanding the underlying geometry.

© Minju Song, Mook Kwon Jung, and Hee-Kap Ahn;
licensed under Creative Commons License CC-BY 4.0

Thus, it has been studied under different metrics [10, 17, 18, 22]. Gezalyan and Mount presented an $O(mn \log n)$-time algorithm for computing the nearest-point Voronoi diagram of n point sites in the Hilbert metric defined by a convex m-gon [15] and an $O(n(\log n + \log^3 m))$-time algorithm for computing the Delaunay triangulation [14]. They showed that the diagram has complexity $\Theta(mn)$. No previous work, however, is known about the farthest-point Voronoi diagram in the Hilbert metric. The farthest-point Voronoi diagram may reveal further characteristics of Hilbert geometry and its potential applications.

1.1 Our results and outline

We denote by $\mathsf{FV}(S)$ the farthest-point Voronoi diagram of n point sites in the Hilbert metric defined by a convex m-gon Ω. Unlike Voronoi diagrams in the Euclidean metric, $\mathsf{FV}(S)$ poses a few difficulties in the computation. The Hilbert distance between points is defined as the logarithm of the cross-ratio of them and the intersections of the line through the points with the domain boundary. It takes $O(\log m)$ time to compute the distance between points. The bisector between any two sites is a piecewise conic curve consisting of $O(m)$ segments. Thus, a naïve divide-and-conquer algorithm using bisectors may take $\Omega(\min\{nm, m^2\})$ time.

We show that $\mathsf{FV}(S)$ has combinatorial complexity $\Theta(m)$, which is independent of the number of sites. Moreover, we present an $O(n(\log n + \log^2 m) + m \log n)$-time algorithm for computing the diagram. This is the first algorithm for computing $\mathsf{FV}(S)$.

In Section 2, we introduce the Hilbert metric, Hilbert balls, Hilbert bisectors, and $\mathsf{FV}(S)$. In the Euclidean metric, a site of S has a nonempty cell in the farthest-point Voronoi diagram if and only if it is a vertex of the convex hull of S [11]. This is, however, not always the case in the Hilbert metric. If a site has a nonempty cell in $\mathsf{FV}(S)$, it is a vertex of the convex hull. But not every vertex of the convex hull always has a nonempty cell in $\mathsf{FV}(S)$. In Section 3, we characterize this phenomenon and show that each cell in $\mathsf{FV}(S)$ is connected and incident to the boundary of Ω. We give an ordering lemma for the sites of S and their Voronoi cells appearing along the boundary of Ω.

In Section 4, we analyze the combinatorial complexity of $\mathsf{FV}(S)$. We show that $\mathsf{FV}(S)$ has $O(m)$ edges and each Voronoi cell is connected. Using this, we show that the number of sites with nonempty cells is $O(m)$. By Euler's formula, we show that $\mathsf{FV}(S)$ has $O(m)$ vertices. We conclude that the combinatorial complexity of $\mathsf{FV}(S)$ is $O(m)$. Additionally, since the complexity of $\mathsf{FV}(S)$ is at least the complexity of Ω, a lower bound on the combinatorial complexity of $\mathsf{FV}(S)$ is $\Omega(m)$. As a result, we give a tight upper bound on the combinatorial complexity of $\mathsf{FV}(S)$, which is $\Theta(m)$ and independent of the number of the sites of S.

In Section 5, we give an efficient algorithm of $O(n(\log n + \log^2 m) + m \log n)$ time for computing $\mathsf{FV}(S)$. Our algorithm consists of three stages. The subsections in Section 5 describe these stages of the algorithm. In the first stage, we compute the convex hull of S and remove the points contained in the interior of the convex hull of S. In the second stage, we construct $\mathsf{FV}(S)$ restricted to the boundary of Ω incrementally by adding the sites of S one by one in clockwise orientation along the convex hull of S. In doing so, we maintain a list $\mathcal{L}$ of sites that have nonempty boundary cell. Initially, $\mathcal{L}$ contains the first two consecutive sites of S. When the next site s is added, we check whether s is the farthest from an intersection point of the boundary cells corresponding to the first and the last sites stored in $\mathcal{L}$. If s is not the farthest one from the point, it has an empty cell so we are done and move on to the next iteration. If s is the farthest one, it has a nonempty cell so we append s to $\mathcal{L}$. We compute the boundary cell of s along the boundary of Ω from the intersection point. In this way, we compute all sites of S whose Voronoi cell is nonempty in $O(n(\log n + \log^2 m))$ time.

In the third stage, we construct $\mathsf{FV}(S)$ by subdividing Ω into Voronoi cells using $\mathsf{FV}(S)$ restricted to the boundary of Ω. Our algorithm uses a divide-and-conquer technique similar to the one by Shamos and Hoey [24] that computes the nearest-site Voronoi diagram in the plane under the Euclidean metric. Roughly speaking, we partition a site sequence into two subsequences, S_L and S_R, of roughly equal size, and compute $\mathsf{FV}(S_L \cup S_R)$ by merging $\mathsf{FV}(S_L)$ and $\mathsf{FV}(S_R)$ along their bisector recursively. But the merge step requires a novel algorithmic idea and an in-depth analysis as each bisector is a piecewise conic curve consisting of $O(m)$ segments. In each recursive step, we compute approximate bisectors such that they become the exact bisectors in the last recursive depth. We show that this can be done in $O(m \log \min\{m, n\})$ time in total. Therefore, $\mathsf{FV}(S)$ can be computed in $O(n(\log n + \log^2 m) + m \log n)$ time in total.

2 Preliminaries

For any two points $p, q \in \mathbb{R}^d$, we denote the line segment connecting p and q by pq, and denote the Euclidean distance between p and q by $|pq|$. A set X of points is *convex* if for any two points $p, q \in X$, every point $t \in pq$ is contained in X. The *convex hull* of X is the smallest convex set that contains X, which we denote by $\mathsf{conv}(X)$. We denote the boundary of X by ∂X and the interior of X by $\mathsf{int}(X)$. For any two points $p, q \in \Omega$, let $\chi(p, q)$ denote the intersection of the line through two points p and q with Ω.

2.1 The Hilbert metric and Hilbert balls

▶ **Definition 1** (Hilbert metric). *Given a convex body Ω and two points $p, q \in \mathsf{int}(\Omega)$, let $\bar{p}$ and $\bar{q}$ denote the endpoints of $\chi(p, q)$ such that $\bar{p}, p, q, \bar{q}$ appear in order along $\chi(p, q)$. See Figure 1(a). The* Hilbert distance *$d(p, q)$ between p and q is defined as*

$$d(p, q) = \frac{1}{2} \ln \left(\frac{|\bar{q}p|}{|\bar{q}q|} \frac{|\bar{p}q|}{|\bar{p}p|} \right).$$

The quantity in the logarithm is the cross-ratio of $(p, q; \bar{q}, \bar{p})$. If p or q lies on $\partial\Omega$, $d(p, q)$ can be formally defined to be $+\infty$. This corresponds to a limiting case that a denominator approaches zero. The Hilbert distance d is extended to all pairs of points by letting $d(p, p) = 0$. It satisfies the axioms of a metric, and in particular, it is symmetric and the triangle inequality holds. Observe $d(p, q) + d(q, r) = d(p, r)$ if p, q, and r are collinear in Ω [21].

In the remainder of the paper, we abuse the notation and use Ω to denote a convex polygon with m vertices in the plane. For any two points $p, q \in \mathsf{int}(\Omega)$, a simple binary search makes it possible to determine the two edges of $\partial\Omega$, each containing an endpoint of $\chi(p, q)$. Thus, the Hilbert distance between two points can be computed in $O(\log m)$ time.

For a point $p \in \mathsf{int}(\Omega)$ and $\rho > 0$, let $B_\Omega(p, \rho)$ denote the *Hilbert ball* of radius ρ centered at p. Nielsen and Shao characterized the shape of Hilbert balls and showed how to compute them [20]. Consider the set of $\chi(p, v)$'s for vertices v of Ω, which we call the *spokes* of p. See Figure 1(a). For each $\chi(p, v)$, there are exactly two points lying on $\chi(p, v)$ whose Hilbert distance from p is ρ. Those points are all in convex position. Then $B_\Omega(p, \rho)$ is the convex polygon with vertices on those $2m$ points. We say $B_\Omega(p, \rho)$ is the Hilbert ball centered at p with radius ρ. For a point $x \in B_\Omega(p, \rho)$, $d(p, x) \leq \rho$; the equality holds for x lying on the boundary of $B_\Omega(p, \rho)$. For a point $x \in \Omega \setminus B_\Omega(p, \rho)$, $d(p, x) > \rho$. See Figure 1(b).

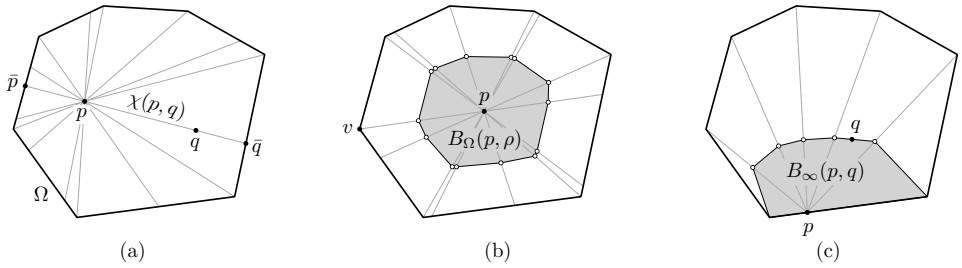


Figure 1 (a) The endpoints $\bar{p}$ and $\bar{q}$ of $\chi(p,q)$ and spokes defined by p. (b) For any point x in the boundary of $B_\Omega(p,\rho)$, $d(p,x)=\rho$. (c) The infinite Hilbert ball $B_\infty(p,q)$.

We extend the concept of Hilbert balls to ones centered at points in $\partial\Omega$, which we call *infinite Hilbert balls*. Let p be a point in $\partial\Omega$, and let q be a point in $\mathsf{int}(\Omega)$. For arbitrarily small $\delta>0$, let p_δ be the point on pq with $|pp_\delta|=\delta$. As δ approaches 0, any sequence of Hilbert balls in the family $\{B_\Omega(p_\delta, d(p_\delta,q))\}$ converges to the infinite Hilbert ball centered at p with radius $d(p_\delta,q)=d(p,q)$, which we denote by $B_\infty(p,q)$. See Figure 1(c).

2.2 Hilbert bisectors

Gezalyan and Mount [15] introduced the *Hilbert bisector* of two points $p,q\in\mathsf{int}(\Omega)$, denoted by $\gamma(p,q)$, which is the set of points $x\in\Omega$ satisfying $d(x,p)=d(x,q)$. For any point $x\in\gamma(p,q)\cap\mathsf{int}(\Omega)$, $q\in\partial B_\Omega(x,d(x,p))$ and $p\in\partial B_\Omega(x,d(x,q))$. For any point $x\in\gamma(p,q)\cap\partial\Omega$, $q\in\partial B_\infty(x,p)$ and $p\in\partial B_\infty(x,q)$. For a point $x\in\Omega$, let p_1,p_2 be two endpoints of $\chi(x,p)$, and let q_1,q_2 be two endpoints of $\chi(x,q)$. Let ℓ_1, ℓ, and ℓ_2 be the lines containing the segments p_1q_1, pq, and p_2q_2, respectively. If $x\in\partial\Omega$, let $x=p_2=q_2$ and let ℓ_2 be the line tangent to Ω at x. It is shown that two cross ratios, $(x,p;p_1,p_2)$ and $(x,q;q_1,q_2)$, are the same if and only if these three lines meet at a common point. It follows $d(x,p)=d(x,q)$, and thus x is on the bisector. See Figure 2(a-b).

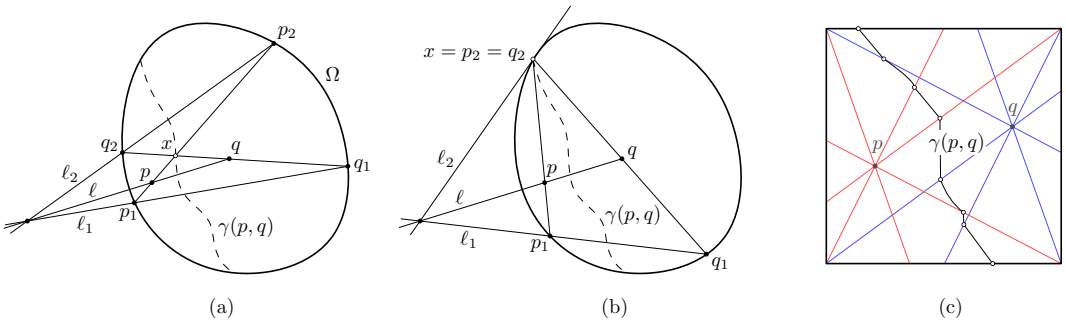


Figure 2 $\gamma(p,q)$ for cases (a) $x\in\mathsf{int}(\Omega)$ and (b) $x\in\partial\Omega$. (c) The subdivision of Ω (square) by the spokes of p (red) and q (blue). $\gamma(p,q)$ is a piecewise conic curve with joints on the spokes of p and q.

The Hilbert distances $d(p,x)$ and $d(q,x)$ of a point $x\in\mathsf{int}(\Omega)$ are determined by the boundary edges of Ω incident to $\chi(p,x)$ and $\chi(q,x)$, respectively. Consider the subdivision of Ω by all spokes of p and all spokes of q. Observe that for every point x in a subregion, the set of edges of $\partial\Omega$ incident to $\chi(p,x)$ and $\chi(q,x)$ remains the same. Thus, $d(p,x)$ and $d(q,x)$ for points x in a subregion can be formulated in explicit functions, and thus we can obtain the Hilbert bisector of p and q restricted to the subregion from the functions. It has

been shown that $\gamma(p, q)$ in a subregion is a conic curve. It follows that $\gamma(p, q)$ is a piecewise conic curve whose joints lie on subregion boundaries [9]. Thus, we say that $\gamma(p, q)$ consists of *bisector segments*. See Figure 2(c).

▶ **Lemma 2** (Lemma 2 of [14]). *For any three points in* $\mathsf{int}(\Omega)$ *not lying on the same line, there is at most one Hilbert ball whose boundary contains them.*

Lemma 2 naturally extends for infinite Hilbert balls. The lemma implies that $\gamma(p, q) \cap \gamma(q, r)$ is at most one point for any three points $p, q, r \in \mathsf{int}(\Omega)$ not lying on the same line.

▶ **Lemma 3.** *Two distinct (infinite) Hilbert balls intersect each other along their boundaries in at most two connected components.*

2.3 Farthest-point Voronoi diagram

Let S be a set of n points lying in $\mathsf{int}(\Omega)$ which we call *sites*. The *farthest-point Voronoi diagram* of S in the Hilbert distance on Ω, denoted by $\mathsf{FV}(S)$, is a subdivision of Ω into cells in which the same point of S is the farthest point in the Hilbert distance. The *farthest-point Voronoi cell* of a site $s \in S$, denoted by $V(s)$, is the set of points whose farthest site is s in the Hilbert distance. Consider a point $p \in \Omega$ contained in $V(s)$. If $p \in \mathsf{int}(\Omega)$, $d(p, s) \geq d(p, s')$ for all sites $s' \in S$. If $p \in \partial\Omega$, there is a point sequence $\{q_i\}_{i \in \mathbb{N}}$ contained in $\mathsf{int}(\Omega)$ such that the sequence converges to p and $d(q_i, s) \geq d(q_i, s')$ for all sites $s' \in S$ and for all $i \in \mathbb{N}$. Not every point of S has a nonempty cell in $\mathsf{FV}(S)$. We will investigate this in Section 3.

General Position Assumption. We assume that no three sites in S are collinear, and in particular, the line passing through any pair of sites of S and the lines extending any two edges of Ω are not coincident at a common point, including all three being parallel. If this assumption does not hold, the bisectors separating Voronoi cells can widen into 2-dimensional regions. These general position conditions were also assumed by Gezalyan and Mount [15].

3 Conditions for valid sites

We study the conditions for a site to have a nonempty cell in $\mathsf{FV}(S)$. We say a site $s \in S$ is *valid* for $\mathsf{FV}(S)$ if $V(s) \neq \emptyset$. As for the farthest-point Voronoi diagram in the Euclidean metric, a site contained in the interior of $\mathsf{conv}(S)$ cannot have a nonempty cell in $\mathsf{FV}(S)$.

▶ **Lemma 4.** *If a site* s *is contained in* $\mathsf{int}(\mathsf{conv}(S))$*,* s *is not valid for* $\mathsf{FV}(S)$*.*

Proof. Suppose that $s \in \mathsf{int}(\mathsf{conv}(S))$ is valid for $\mathsf{FV}(S)$. Then there is a point $p \in V(s)$ such that $d(p, s) \geq d(p, s')$ for all $s' \in S$. This implies that $B_\Omega(p, d(p, s))$ contains all sites of S. Since $B_\Omega(p, d(p, s))$ is convex, $\mathsf{conv}(S)$ is contained in $B_\Omega(p, d(p, s))$. Then $s \in \mathsf{int}(\mathsf{conv}(S)) \subseteq \mathsf{int}(B_\Omega(p, d(p, s)))$, contradicting that s lies on the boundary of $B_\Omega(p, d(p, s))$. ◀

In the Euclidean metric, every site has a nonempty cell in the farthest-point Voronoi diagram if it appears as a vertex of the convex hull of the sites. This is not necessarily true for $\mathsf{FV}(S)$ in the Hilbert metric. For two distinct points $p, q \in \mathsf{int}(\Omega)$, let B_1 and B_2 be two infinite Hilbert balls, each of which contains both p and q on its boundary. Let $Z(p, q) = B_1 \cap B_2$. Gezalyan et al. [14] showed that no Hilbert ball contains p, q, and a point $r \in \mathsf{int}(Z(p, q))$ on its boundary. Using Lemma 2, they showed that every Hilbert ball centered at a point in $\gamma(p, q)$ and containing both p and q on its boundary contains $Z(p, q)$. See Figure 3(a).

▶ **Lemma 5.** *If a site is contained in* $Z(p, q)$ *for sites* p *and* q*, it is not valid for* $\mathsf{FV}(S)$*.*

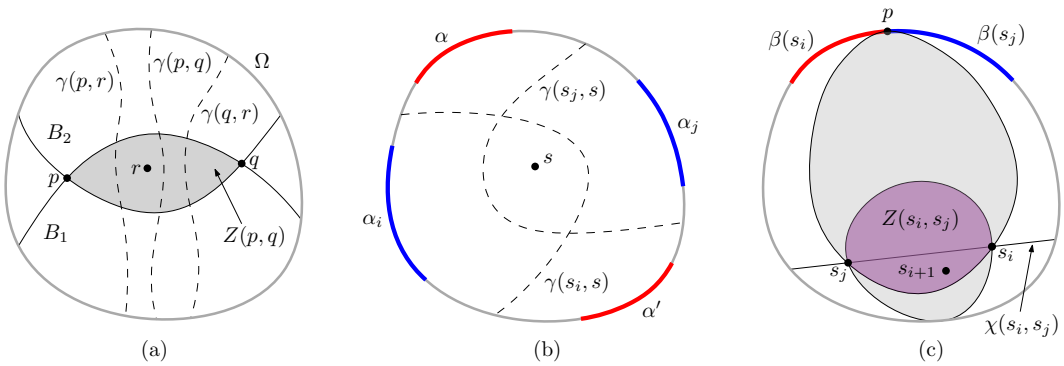


Figure 3 (a) $Z(p,q) = B_1 \cap B_2$ for two infinite Hilbert balls B_1 and B_2 with $p \in \partial B_1$ and $q \in \partial B_2$. $\gamma(p,q)$ separates $\gamma(p,r)$ from $\gamma(q,r)$. (b) $\alpha_i, \alpha, \alpha_j, \alpha'$ appear in clockwise orientation along $\partial\Omega$. (c) Two infinite Hilbert balls, each containing both s_i and s_j on its boundary. Then $s_{i+1} \in Z(s_i, s_j)$.

Proof. Let r be a site contained in $Z(p,q)$ for some sites p and q. Then no Hilbert ball contains p, q, and r on its boundary [14]. This implies that $\gamma(p,r)$ does not intersect $\gamma(q,r)$. Without loss of generality, we assume that $\gamma(p,q)$ separates p and r from q. Let x be a point contained in $\gamma(p,q)$. Since the Hilbert ball centered at x and containing p and q on its boundary contains r, $d(x,r) < d(x,p) = d(x,q)$. This means that $\gamma(p,r)$ separates r and x from p, and $\gamma(q,r)$ separates r and x from q. Since $\gamma(p,q)$ does not intersect $\gamma(p,r)$ and $\gamma(q,r)$, $\gamma(p,q)$ separates $\gamma(p,r)$ from $\gamma(q,r)$. Then the side of $\gamma(p,r)$ containing p does not intersect the side of $\gamma(q,r)$ containing q. Since $V(r) = \{x \in \Omega \mid \cap_{s \in S} d(x,s) \leq d(x,r)\}$, we have $V(r) = \emptyset$, and thus r is not valid for $\mathsf{FV}(S)$. See Figure 3(a) for an illustration. ◀

By Lemma 5, a site $r \in S$ may be not valid for $\mathsf{FV}(S)$ even for $r \notin \mathsf{int}(\mathsf{conv}(S))$.

3.1 $\mathsf{FV}(S)$ restricted to the domain boundary

To find all valid sites of S for $\mathsf{FV}(S)$ and compute $\mathsf{FV}(S)$ efficiently, we compute $\mathsf{FV}(S)$ restricted to the boundary of Ω. We denote it by $\mathsf{bFV}(S)$. The Voronoi cell in $\mathsf{bFV}(S)$ of a site $s \in S$, denoted by $\beta(s)$, is the intersection of $V(s)$ and $\partial\Omega$. We call a site $s \in S$ *valid* for $\mathsf{bFV}(S)$ if $\beta(s) \neq \emptyset$.

▶ **Lemma 6.** *Let p be a point in $V(s)$ and $\bar{p}$ be the endpoint of $\chi(s,p)$ closer to p than to s. Then $p\bar{p} \subseteq V(s)$.*

Proof. Suppose there is a point $r \in p\bar{p}$ but $r \notin V(s)$. Let t be a site of S with $r \in V(t)$. By definition, $d(s,p) \geq d(t,p)$ and $d(t,r) > d(s,r)$. Combining these we have $d(s,p) + d(t,r) > d(t,p) + d(s,r)$, or equivalently $d(t,r) > d(t,p) + d(s,r) - d(s,p) = d(t,p) + d(p,r)$ as s,p,r are collinear [21]. But this violates the triangle inequality, yielding a contradiction. ◀

Since $\mathsf{bFV}(S)$ is the restriction of $\mathsf{FV}(S)$ on the boundary of Ω, a site valid for $\mathsf{bFV}(S)$ is also valid for $\mathsf{FV}(S)$. Lemma 6 implies that a site valid for $\mathsf{FV}(S)$ is also valid for $\mathsf{bFV}(S)$. Thus, we can identify all valid sites of S for $\mathsf{FV}(S)$ from $\mathsf{bFV}(S)$.

▶ **Lemma 7.** *$\beta(s)$ is connected for a site $s \in S$.*

Proof. If s is not valid for $\mathsf{FV}(S)$, $\beta(s) = \emptyset$ and we are done. Assume for the contradiction that s is valid for $\mathsf{FV}(S)$ and $\beta(s)$ is not connected for a site $s \in S$. Let α and α' be two connected components of $\beta(s)$. Then there are two sites $s_i, s_j \in S \setminus \{s\}$, not necessarily

distinct, with connected components α_i of $\beta(s_i)$ and α_j of $\beta(s_j)$ such that $\alpha_i, \alpha, \alpha_j, \alpha'$ appear in order along $\partial\Omega$ in clockwise orientation, as shown in Figure 3(b). Observe that α_i and α are separated by $\gamma(s_i, s)$, and α_j and α' are separated by $\gamma(s_j, s)$. Since both the side of $\gamma(s_i, s)$ containing α_i and the side of $\gamma(s_j, s)$ containing α_j contain s, $\gamma(s_i, s)$ and $\gamma(s_j, s)$ intersect at least two times. Since s, s_i, and s_j are not collinear, this contradicts Lemma 2. ◀

By Lemmas 6 and 7, $\beta(s)$ is nonempty and connected for every valid site s of S. Aronov et al. [4] gave an *Ordering Lemma* for the farthest-point geodesic Voronoi diagram for point sites in a simple polygon. We extend the ordering lemma for $\mathsf{FV}(S)$ with respect to $\partial\Omega$.

▶ **Lemma 8.** *The order of valid sites along their convex hull is the same as the order of their Voronoi cells along the boundary of* Ω.

Proof. For ease of description, we relabel the sites such that $\langle s_0, s_1 \ldots, s_{t-1}\rangle$ is the sequence of sites valid for $\mathsf{FV}(S)$ in clockwise orientation along their convex hull. We use modulo t for indices. First, we show that $\beta(s_i)$ and $\beta(s_{i+1})$ are adjacent to each other for every $i = 0, \ldots, t-1$. Suppose that $\beta(s_i)$ and $\beta(s_{i+1})$ are not adjacent to each other for some i. Let s_j be a site such that $\beta(s_j)$ and $\beta(s_i)$ are adjacent to each other and $\beta(s_j)$ appears right after $\beta(s_i)$ in clockwise orientation along $\partial\Omega$. Since $p = \beta(s_i) \cap \beta(s_j)$ is an endpoint of $\gamma(s_i, s_j)$, $B_\infty(p, s_i)$ and $B_\infty(p, s_j)$ are the same. Then $s_{i+1}, s_{j+1} \in B_\infty(p, s_i)$. See Figure 3(c).

Observe that $\chi(s_i, s_j)$ separates s_{i+1} from s_{j+1} since all sites $\langle s_0, s_1 \ldots, s_{t-1}\rangle$ appear in clockwise orientation along their convex hull. Without loss of generality, assume that s_{i+1} is contained in the side of $\chi(s_i, s_j)$ not containing p. Let q be the endpoint of $\gamma(s_i, s_j)$ other than p. Thus, q is contained in the side of of $\chi(s_i, s_j)$ containing s_{i+1}. Then $B_\infty(q, s_i)$ and $B_\infty(q, s_j)$ are the same, and thus $B_\infty(q, s_i)$ contains both s_i and s_j on its boundary. By Lemma 3, $B_\infty(p, s_i)$ and $B_\infty(q, s_i)$ intersect each other along their boundaries at most twice. Thus, s_{i+1} is contained in $B_\infty(q, s_i)$. Since $s_{i+1} \in B_\infty(p, s_i) \cap B_\infty(q, s_i)$, it follows from Lemma 5 that s_{i+1} is not valid for $\mathsf{FV}(S)$. This is a contradiction that s_{i+1} is valid. Thus, $\beta(s_i)$ and $\beta(s_{i+1})$ are adjacent to each other.

There are two cases: $\langle \beta(s_0), \ldots, \beta(s_{t-1})\rangle$ appear in order in clockwise orientation along $\partial\Omega$ or they appear in order in counterclockwise orientation along $\partial\Omega$. Suppose the latter case. Then for some i, the side of $\chi(s_{i-1}, s_{i+1})$ containing s_i contains a point in $\beta(s_i)$. Let p and q be the endpoints of $\gamma(s_{i-1}, s_{i+1})$ such that $\beta(s_{i-1}), p, \beta(s_{i+1})$, and q appear in clockwise orientation along $\partial\Omega$. By definition, s_i is contained in $B_\infty(p, s_{i+1})$. Observe that s_i is contained in $B_\infty(q, s_{i+1})$ by the argument in the previous paragraph. Thus, s_i is contained in $Z(s_{i-1}, s_{i+1})$. By Lemma 5, s_i is not valid for $\mathsf{FV}(S)$, a contradiction. Thus, $\beta(s_1), \ldots, \beta(s_n)$ appear in order in clockwise orientation along $\partial\Omega$. ◀

4 Complexity

We analyze the combinatorial complexity of $\mathsf{FV}(S)$ by counting the numbers of vertices, bisector segments, and edges of $\mathsf{FV}(S)$. Let $\hat{S} = \{s_0, s_1 \ldots, s_{t-1}\}$ denote the set of sites valid for $\mathsf{FV}(S)$ such that $s_0, s_1 \ldots, s_{t-1}$ appear in clockwise orientation along their convex hull. We use modulo t for the indices of the sites in $\hat{S}$. Obviously, $\mathsf{FV}(S)$ and $\mathsf{FV}(\hat{S})$ are the same.

For two sites $s, s' \in \hat{S}$, we call each connected portion of $\gamma(s, s')$ that appears in $\mathsf{FV}(\hat{S})$ a *Voronoi edge* defined by s and s'. Thus, a Voronoi edge is a curve consisting of bisector segments of $\gamma(s, s')$, with the exception that the first and last segments might be truncated. Each endpoint of a Voronoi edge is a *Voronoi vertex*, which has degree 3 (if contained in the interior of Ω) or 1 (if contained in $\partial\Omega$). Each endpoint of a bisector segment is either a Voronoi vertex or a vertex with degree 2. We show that $\mathsf{FV}(\hat{S})$ has $O(m)$ Voronoi edges,

and that $\mathsf{FV}(\hat{S})$ consists of $O(m)$ Voronoi cells and $O(m)$ Voronoi vertices. This implies that $\mathsf{FV}(\hat{S})$ has complexity $O(m)$. Since the complexity of $\mathsf{FV}(\hat{S})$ is at least the complexity of Ω, $\mathsf{FV}(\hat{S})$ has complexity $\Omega(m)$. Thus, the combinatorial complexity of $\mathsf{FV}(\hat{S})$ is $\Theta(m)$.

4.1 Complexity of $\mathsf{FV}(S)$

Gezalyan and Mount [15] showed that the combinatorial complexity of the nearest-point Voronoi diagram is $O(mn)$. We show that the combinatorial complexity of the farthest-point Voronoi diagram is $O(m)$. Recall that every bisector is a piecewise conic with joints lying on spokes. To analyze the combinatorial complexity of $\mathsf{FV}(S)$, we count the number of Voronoi vertices and the total number of bisector segments in $\mathsf{FV}(S)$. Let $\kappa(p)$ denote the set of farthest sites for a point p in $\partial\Omega$. If $\kappa(p)$ consists of a single element, we use $\kappa(p)$ to denote the element. The following lemma shows an upper bound on the number of bisector segments.

▶ **Lemma 9.** *There are $O(m)$ bisector segments in $\mathsf{FV}(S)$.*

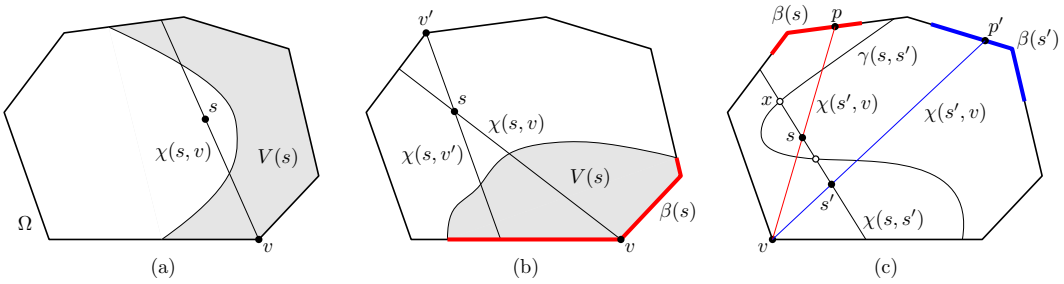


Figure 4 (a) $\chi(s_i, v)$ crosses $\partial V(s_i)$ at most twice. (b) $\chi(s, v)$ for $v \in \beta(s)$ and $\chi(s, v')$ for $v' \notin \beta(s)$ cross $\partial V(s)$. (c) A point $x \in \chi(s, s') \cap \gamma(s, s')$ not contained in ss'.

Proof. We count the number of bisector segments in $\mathsf{FV}(S)$ by counting the spokes of s that cross the boundary of $V(s)$ for each $s \in S$. Consider a bisector segment σ of $\gamma(s_i, s_j)$ in $\mathsf{FV}(S)$ for two sites s_i and s_j. Since $\gamma(s_i, s_j)$ is a piecewise conic with joints on spokes of s_i and s_j, each endpoint of σ lies on a spoke of s_i or s_j unless it is a Voronoi vertex. Let v be a vertex of Ω such that an endpoint of σ lies on $\chi(s_i, v)$. Then $\chi(s_i, v)$ intersects $V(s_i)$. By Lemma 6, $\chi(s_i, v)$ intersects at most two bisector segments in $\partial V(s_i)$. See Figure 4(a). This implies that the number of bisector segments in $\mathsf{FV}(S)$ can be obtained by counting the total number of spokes of s that intersect $\partial V(s)$ for each $s \in S$.

Consider $\chi(s, v)$ for a site s and a vertex v of Ω such that $\chi(s, v) \cap V(s) \neq \emptyset$. Then either $v \in \beta(s)$ or $v \notin \beta(s)$. See Figure 4(b). If $v \in \beta(s)$, v is incident to at most two Voronoi cells, and thus the total number of such spokes is $O(m)$.

Consider the case that $v \notin \beta(s)$. Assume for the contradiction that there is another site s' $(\neq s)$ such that $\chi(s', v) \cap V(s') \neq \emptyset$ but $v \notin \beta(s')$. Let p be the intersection of $\chi(s, v)$ and $\beta(s)$, and let p' be the intersection of $\chi(s', v)$ and $\beta(s')$. See Figure 4(c). If v, s, and s' are collinear, $p = p'$ and $\kappa(p) = \kappa(p')$, implying $s = s'$.

So assume that v, s, and s' are not collinear. Since $\gamma(s, s')$ separates $\beta(s)$ and s' from $\beta(s')$ and s, $\gamma(s, s')$ crosses each of ps, ss', and $s'p'$ in an odd number of times. Thus, there is a point x on $\chi(s, s') \cap \gamma(s, s')$ that is not contained in ss'. By definition, $d(x, s) = d(x, s')$. Since x, s, and s' are all contained in $\chi(s, s')$, we have $d(x, s') = d(x, s) + d(s, s')$ or $d(x, s) = d(x, s') + d(s, s')$, implying that $d(s, s') = 0$ and thus $s = s'$.

Hence, the total number of such spokes is $O(m)$. Since there are $O(m)$ spokes that cross cell boundaries in $\mathsf{FV}(S)$, the total number of bisector segments in $\mathsf{FV}(S)$ is $O(m)$. ◀

It remains to count the number of Voronoi vertices, which can be done by Euler's formula.

▶ **Lemma 10.** *For a site $s \in S$, $V(s)$ is connected.*

Proof. Assume for the contradiction that $V(s)$ is not connected for a site $s \in S$. Let C_1 and C_2 be two connected components of $V(s)$. For each $i = 1, 2$, let p_i be a point lying inside C_i, and let q_i be the point closer to p_i than s among two endpoints of $\chi(s, p_i)$. By Lemma 6, $p_i q_i \subseteq V(s)$. However, q_1 and q_2 are not connected in $\beta(s)$, which contradicts Lemma 7. ◀

▶ **Lemma 11.** *There are $O(m)$ sites in $\mathsf{FV}(S)$.*

Proof. By Lemmas 6 and 9, there are $O(m)$ sites valid for $\mathsf{bFV}(S)$. Thus, there are $O(m)$ sites in $\mathsf{FV}(S)$ by Lemma 8. ◀

▶ **Theorem 12.** *The farthest-point Voronoi diagram of n point sites lying inside a convex m-gon in the Hilbert metric has combinatorial complexity $O(m)$.*

Proof. By Lemmas 10 and 11, the number of Voronoi cells is $O(m)$. Since the number of bisector segments is $O(m)$ by Lemma 9, the number of Voronoi edges is $O(m)$. By Euler's formula, the number of Voronoi vertices is $O(m)$. ◀

5 Algorithm

In this section, we present an algorithm to compute $\mathsf{FV}(S)$. Our algorithm computes $\mathsf{bFV}(S)$ and identifies all valid sites. Then it computes $\mathsf{FV}(S)$ in a divide-and-conquer manner.

5.1 Computing $\mathsf{FV}(S)$ restricted to the domain boundary

By Lemma 4, every site of S contained in $\mathsf{int}(\mathsf{conv}(S))$ is not valid for $\mathsf{FV}(S)$. Hence, we compute $\mathsf{conv}(S)$ and remove all sites contained in its interior from S. This can be done in $O(n \log n)$ time. Thus, all remaining sites in S are in convex position. We relabel them such that $\langle s_1, s_2 \ldots, s_n \rangle$ is the sequence of sites in clockwise orientation along their convex hull.

Some sites of S, however, may not be valid for $\mathsf{FV}(S)$. We will identify all valid sites of S by computing $\mathsf{bFV}(S)$. We do this incrementally by inserting sites one by one in order, checking if the newly inserted one is valid for the Voronoi diagram of the sites inserted so far, and handling the cases accordingly. We maintain a few structures. Let S_i be the subset of S consisting of the first i sites of S for $i = 2, \ldots, n$. Let $\beta_i(s)$ denote the Voronoi cell of $s \in S_i$ in $\mathsf{bFV}(S_i)$. Let $\mathcal{L}_i$ be the list that maintains all valid sites for $\mathsf{FV}(S_i)$.

Initially, let $S_2 = \{s_1, s_2\}$ and $\mathcal{L}_2 = [s_1, s_2]$. We compute $\mathsf{bFV}(S_2)$ by computing the endpoints of $\gamma(s_1, s_2)$. In the i-th iteration with $i \geq 3$, we compute $\mathsf{bFV}(S_i)$ and maintain $\mathcal{L}_i$ as follows. Let s_l and s_r be the first element and the last element in $\mathcal{L}_{i-1}$, respectively. Then there is a point p shared by $\beta_{i-1}(s_l)$ and $\beta_{i-1}(s_r)$. By Lemma 8, s_i is valid if and only if $p \in \beta_i(s_i)$. If $d(s_i, p) < d(s_l, p)$, s_i is not valid for $\mathsf{FV}(S_i)$, and thus it is not valid for $\mathsf{FV}(S)$. We set $\mathcal{L}_i = \mathcal{L}_{i-1}$ and move on to the next iteration. If $d(s_i, p) \geq d(s_l, p)$, s_i is valid for $\mathsf{FV}(S_i)$. We set $\mathcal{L}_i = \mathcal{L}_{i-1}$. Then we update $\mathsf{bFV}(S_i)$ from $\mathsf{bFV}(S_{i-1})$ by computing the endpoints of $\beta_i(s_i)$. This can be done by searching them along the boundary of Ω from p, once in clockwise orientation and once in counterclockwise orientation. In doing so, we update $\mathcal{L}_i$ accordingly, and then append s_i to $\mathcal{L}_i$.

Consider the case that we search for an endpoint of $\beta_i(s_i)$ along the boundary of Ω in counterclockwise orientation from p. Let q be the endpoint of $\beta_{i-1}(s_r)$ other than p. Observe that $q \in \beta_{i-1}(s_{r'})$ for site $s_{r'}$ previous to s_r in $\mathcal{L}_{i-1}$. If $d(s_i, q) < d(s_{r'}, q)$, one endpoint of

$\beta_i(s_i)$ lies on $\beta_{i-1}(s_r)$. We compute the endpoint of $\gamma(s_r, s_i)$ on $\beta_{i-1}(s_r)$. Otherwise, s_r is not valid for $\mathsf{FV}(S_i)$, and thus it is not valid for $\mathsf{FV}(S)$. We remove s_r from $\mathcal{L}_i$ and move on to $s_{r'}$. We repeat this until one endpoint of $\beta_i(s_i)$ is found. See Figure 5.

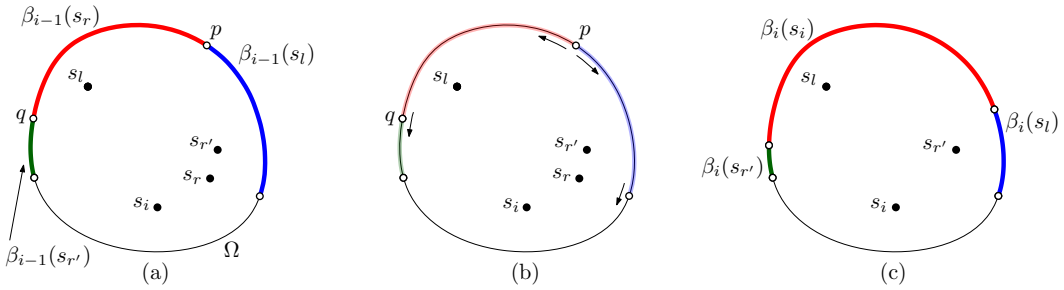


Figure 5 (a) In the i-th iteration, s_i is inserted. (b) If $d(s_i, p) \geq d(s_l, p)$, s_i is valid for $\mathsf{FV}(S_i)$. We compute the endpoints of $\beta_{i-1}(s_i)$ along $\partial\Omega$ from p in both orientations. (c) $\mathsf{bFV}(S_i)$.

We find the other endpoint of $\beta_i(s_i)$ in clockwise orientation similarly. After computing both endpoints of $\beta_i(s_i)$, we append s_i to $\mathcal{L}_i$.

▶ **Lemma 13** (Lemma 7 of [14]). *Given an m-sided convex polygon Ω and any two points $p, q \in \mathit{int}(\Omega)$, the endpoints of $\gamma(p, q)$ on $\partial\Omega$ can be computed in $O(\log^2 m)$ time.*

We can compute $d(s_i, p)$ and $d(s_l, p)$ in $O(\log m)$ time. If $d(s_i, p) < d(s_l, p)$, we are done. If $d(s_i, p) \geq d(s_l, p)$, we identify all sites not valid for $\mathsf{FV}(S_i)$ among sites valid for $\mathsf{FV}(S_{i-1})$, and remove them from $\mathcal{L}_i$. This takes $O(n \log m)$ time in total over all insertion steps. Then we compute the endpoints of $\beta_i(s_i)$. Observe that one endpoint of $\beta_i(s_i)$ is an endpoint of $\gamma(s_i, s)$ and the other endpoint of $\beta_i(s_i)$ is an endpoint of $\gamma(s_i, s')$, where s and s' are the first and the last elements in $\mathcal{L}_i$. We can compute the endpoints of $\beta_i(s_i)$ in $O(\log^2 m)$ time by Lemma 13. Since the number of sites is n, it takes $O(n \log^2 m)$ time in total to compute $\mathsf{bFV}(S)$.

▶ **Lemma 14.** *$\mathit{bFV}(S)$ can be computed in $O(n(\log n + \log^2 m))$ time.*

5.2 Computing $\mathsf{FV}(S)$

We have preprocessed $\mathsf{bFV}(S)$ of size $O(m)$ and all $O(\min\{m, n\})$ sites valid for $\mathsf{FV}(S)$ and $\mathsf{bFV}(S)$ by Lemma 14. We give an algorithm for computing $\mathsf{FV}(S)$ in $O(\min\{m \log m, m \log n\})$ time by subdividing Ω into Voronoi cells using $\mathsf{bFV}(S)$. Our algorithm is based on a divide-and-conquer technique similar to the one by Shamos and Hoey [24] that computes the nearest-site Voronoi diagram in the plane under the Euclidean metric. But the merge step requires a novel algorithmic idea and an in-depth analysis to reduce the running time as each bisector is a piecewise conic curve consisting of $O(m)$ segments.

Let $\hat{S} = \langle s_0, \ldots, s_{t-1} \rangle$ denote the sequence of sites valid for $\mathsf{FV}(S)$ such that $s_0, \ldots, s_{t-1}$ appear in clockwise orientation along $\mathsf{conv}(\hat{S})$. Observe $t = O(\min\{m, n\})$. We use modulo t for the indices. For a site $s \in \hat{S}$, let $R(s)$ denote the region in Ω bounded by sp, sq, $\beta(s)$, where p and q are two endpoints of $\beta(s)$. By Lemma 6, $V(s) \subseteq R(s)$. Observe that every spoke of s intersecting $R(s)$ intersects $V(s)$. Let $n(s)$ denote the number of the spokes of s intersecting $R(s)$. The total number of $n(s)$ for all $s \in \hat{S}$ is $O(m)$ by Lemma 9.

We compute $\mathsf{FV}(\hat{S})$, which is the same as $\mathsf{FV}(S)$. Roughly speaking, we partition a site sequence into two subsequences, S_L and S_R, of roughly equal size, and compute $\mathsf{FV}(S_L \cup S_R)$ by merging $\mathsf{FV}(S_L)$ and $\mathsf{FV}(S_R)$ along $\gamma(S_L, S_R)$ recursively. The bisector $\gamma(S_L, S_R)$ satisfies

the property that any point of Ω lying on one side of $\gamma(S_L, S_R)$ is farthest from some site in S_L and any point of Ω lying on the other side of $\gamma(S_L, S_R)$ is farthest from some site in S_R. Observe that $\gamma(S_L, S_R)$ is connected in Ω with two distinct points on $\partial\Omega$. If $\gamma(S_L, S_R)$ was not connected or it had more than two points on $\partial\Omega$, it partitions $\partial\Omega$ into four or more pieces, each belonging to either $\mathsf{bFV}(S_L)$ or $\mathsf{bFV}(S_R)$. Then the cells of $\mathsf{bFV}(S_L)$ or $\mathsf{bFV}(S_R)$ are not consecutive along $\partial\Omega$, contradicting Lemma 8. Also, $\gamma(S_L, S_R)$ has no closed loop since no Voronoi cell in the loop is incident to $\partial\Omega$.

Consider the k-th recursive step of the algorithm. Let S_L and S_R be two input sequences of sites in the step. We have $\mathsf{FV}(S_L)$ and $\mathsf{FV}(S_R)$. For a site $s \in S_L \cup S_R$, let $V_k(s)$ denote the cell of s in $\mathsf{FV}(S_L \cup S_R)$, and let $V_{k-1}(s)$ denote the cell of s in $\mathsf{FV}(S_L)$ for $s \in S_L$ and the cell of s in $\mathsf{FV}(S_R)$ for $s \in S_R$. Let $\beta_k(s) = V_k(s) \cap \partial\Omega$ and $R_k(s) = R(s) \cap V_k(s)$. Let $R_{k-1}(S_L) = \bigcup_{s \in S_L} R_{k-1}(s)$ and $R_{k-1}(S_R) = \bigcup_{s \in S_R} R_{k-1}(s)$.

If three consecutive sites s_{i-1}, s_i, s_{i+1} are contained in $S_L \cup S_R$ for some i, $\beta_k(s_i)$ and $\beta(s_i)$ are the same. Thus, $V_k(s_i) \subseteq R(s_i)$ by Lemma 6, implying that $R_k(s_i)$ is $V_k(s_i)$.

▶ **Lemma 15.** *The followings hold.*

1. $\gamma(S_L, S_R) \cap R_{k-1}(S_L) \cap R_{k-1}(S_R)$ *is connected.*
2. $\gamma(S_L, S_R) \setminus (R_{k-1}(S_L) \cap R_{k-1}(S_R))$ *does not appear in* $\mathsf{FV}(S)$.
3. $\gamma(S_L, S_R) \cap R_{k-1}(s)$ *is connected for any* $s \in S_L \cup S_R$.

Proof. Let $\gamma = \gamma(S_L, S_R)$. Consider Claim 1. Since γ has an endpoint in $R_{k-1}(S_L) \cap R_{k-1}(S_R)$, it suffices to show that $\gamma \cap R_{k-1}(S_L)$ is connected and $\gamma \cap R_{k-1}(S_R)$ is connected. Assume for the contradiction that $\gamma \cap R_{k-1}(S_L)$ is not connected. Then there is a curve C contained in $\gamma \setminus R_{k-1}(S_L)$ which has an endpoint on $\partial R_{k-1}(s_i)$ for some site $s_i \in S_L$. Consider the case $\{s_{i-1}, s_i, s_{i+1}\} \subseteq S_L \cup S_R$. Then there is a point $p \in \Omega \setminus R_{k-1}(s_i)$ such that the farthest site in $S_L \cup S_R$ of p is s_i, contradicting $V_k(s_i) \subseteq R_k(s_i) \subseteq R_{k-1}(s_i)$.

Thus, $s_{i-1} \notin S_L \cup S_R$ or $s_{i+1} \notin S_L \cup S_R$. By definition, $\partial R_{k-1}(s_i)$ consists of a line segment ℓ and a part of $\gamma(s_i, s_l)$'s for some $s_l \in S_L$. If C has an endpoint on $\gamma(s_i, s_l)$, there is a point $p \in \Omega \setminus R_{k-1}(S_L)$ such that the farthest site of p is s_l. This contradicts $p \in R_k(s_l) \subseteq R_{k-1}(s_l)$. Thus, γ intersects ℓ at least three times, contradicting Lemma 6. Therefore, $\gamma \cap R_{k-1}(S_L)$ is connected. We can show that $\gamma \cap R_{k-1}(S_R)$ is connected by exchanging the roles of S_L and S_R. Thus, Claim 1 holds.

Consider Claim 2. Let C be the part of $\gamma(s_l, s_r)$ for $s_l \in S_L$ and $s_r \in S_R$ that appears in $\mathsf{FV}(S)$. Then $C \subseteq V(s_l) \subseteq V_k(s_l)$ and $C \subseteq V(s_r) \subseteq V_k(s_r)$. Since $V(s) \subseteq R_{k-1}(s)$ for every $s \in S_L \cup S_R$, $C \subseteq R_{k-1}(s_l) \cap R_{k-1}(s_r)$. Thus, $C \subseteq R_{k-1}(S_L) \cap R_{k-1}(S_R)$. By contraposition, Claim 2 holds.

Consider Claim 3. Let s be a site in S_L. Let $\tilde{\gamma}$ be the maximal portion of γ contained in $R_{k-1}(S_L)$. By Claim 1, it suffices to show that $\tilde{\gamma} \cap R_{k-1}(s)$ is connected. Assume for the contradiction that $\tilde{\gamma} \cap R_{k-1}(s)$ is not connected. Then there is a region R bounded by $\tilde{\gamma}$ and $R_{k-1}(s)$ satisfying $R \not\subseteq R_{k-1}(s)$. Since $\tilde{\gamma}$ is connected, R does not intersect $\partial\Omega$. By Lemma 10, there is a site $s' \in S_L \cup S_R$ with $V_k(s') \subseteq R$. This contradicts that each Voronoi cell intersects $\partial\Omega$. Similarly, we can show the case for $s \in S_R$. ◀

Lemma 15.1 guarantees that once we compute the endpoint of $\gamma(S_L, S_R)$ contained in $R_{k-1}(S_L) \cap R_{k-1}(S_R)$, we can compute $\gamma(S_L, S_R)$ within $R_{k-1}(S_L) \cap R_{k-1}(S_R)$ by tracing the bisector from the endpoint. Also, Lemma 15.2 implies that it suffices to trace $\gamma(S_L, S_R)$ within $R_{k-1}(S_L) \cap R_{k-1}(S_R)$ in computing $\mathsf{FV}(S)$.

5.2.1 Divide-and-conquer algorithm

Now we are ready to describe our algorithm for computing $\mathsf{FV}(\hat{S})$ from $\mathsf{bFV}(S)$. In the k-th recursive step, we maintain a region $U_k(s)$ which contains $V_k(s)$ for $s \in \hat{S}$. As the base case, we initialize $U_0(s)$ to $R(s)$. For the k-th recursive step, we compute $U_k(s)$ from $U_{k-1}(s)$ for $k \geq 1$ inductively. In the last recursive step, we have $U_k(s) = V(s)$ for every $s \in \hat{S}$.

Consider the recursive step with $S_L = \langle s_i \rangle$ and $S_R = \langle s_{i+1} \rangle$. We compute the point $p \in \beta(s_i) \cap \beta(s_{i+1})$. Then we trace $\gamma(s_i, s_{i+1})$ starting from p until the bisector meets $\partial U_0(s_i)$ or $\partial U_0(s_{i+1})$. Consider the case that $\gamma(s_i, s_{i+1})$ meets $\partial U_0(s_i)$ at q first. We shoot a ray from q towards s_{i+1} and find the point r of $\partial U_0(s_{i+1})$ hit by the ray. Let $\tilde{\gamma}$ be the curve consisting of the part of $\gamma(s_i, s_{i+1})$ from p to q, and qr. Then $\tilde{\gamma}$ partitions $U_0(s_i)$ and $U_0(s_{i+1})$ into two subregions. We set $U_1(s_j)$ to the subregion of $U_0(s_j)$ containing $\beta(s_j)$ for each $j = i, i+1$. See Figure 6. We handle the case that $\gamma(s_i, s_{i+1})$ meets $\partial U_0(s_{i+1})$ first analogously.

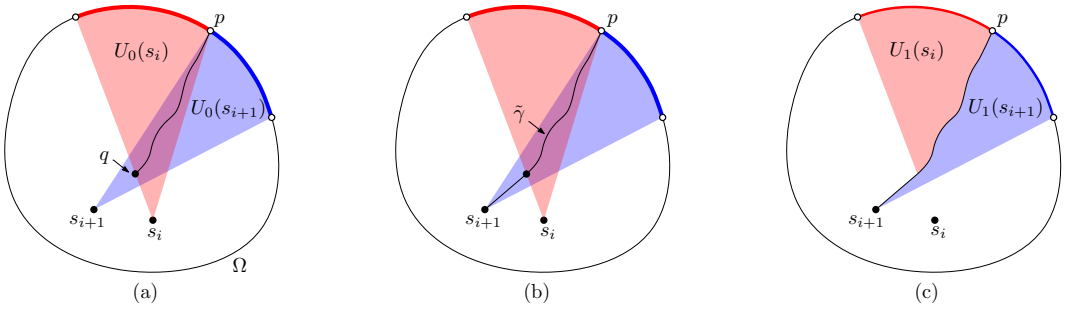


Figure 6 (a) Tracing $\gamma(s_i, s_{i+1})$ from p. (b) Ray from q towards s_{i+1}. (c) $U_1(s_i)$ and $U_1(s_{i+1})$.

Consider the k-th recursive step. Let S_L and S_R be the two input sequences of sites. Without loss of generality, let s_i be the last site of S_L and s_{i+1} be the first site of S_R. Let $U_{k-1}(S_L) = \bigcup_{s \in S_L} U_{k-1}(s)$ and $U_{k-1}(S_R) = \bigcup_{s \in S_R} U_{k-1}(s)$. We find the point $p \in \beta(s_i) \cap \beta(s_{i+1})$. Then we trace $\gamma(s_i, s_{i+1})$ until it leaves (1) $U_{k-1}(s_i)$, (2) $U_{k-1}(s_{i+1})$, (3) $U_{k-1}(S_L)$, or (4) $U_{k-1}(S_R)$.

Consider the case that (1) holds but (3) does not hold. Let q be the point where $\gamma(s_i, s_{i+1})$ leaves $U_{k-1}(s_i)$ and $\gamma(s_i, s_{i+1})$ enters $U_{k-1}(s')$ for $s' \in S_L$. If $U_{k-1}(s')$ has been visited before, we trace a ray from q towards $s'' \in S_R$ satisfying $q \in U_{k-1}(s'')$, until the first intersection with $\partial U_{k-1}(s'')$. If $U_{k-1}(s')$ has not been visited before, we repeat tracing $\gamma(s_{i+1}, s')$ from q inwards $U_{k-1}(s')$ as before until it leaves one of those four regions. See Figure 7(a). We handle the case that (2) holds but (4) does not hold analogously.

Consider the case that (3) holds. Let q be the point where $\gamma(s_i, s_{i+1})$ leaves $U_{k-1}(S_L)$. We trace a ray from q towards $s'' \in S_R$ satisfying $q \in U_{k-1}(s'')$ until the first intersection point with $\partial U_{k-1}(s'')$. See Figure 7(a). We handle the case that (4) holds analogously.

Let $\tilde{\gamma}$ be the resulting curve from the tracing above. Then $\tilde{\gamma}$ partitions $U_{k-1}(s)$ into two subregions for some $s \in S_L \cup S_R$. We set $U_k(s)$ to the subregion of $U_{k-1}(s)$ containing $\beta(s)$.

Observe that $R_k(s) \subseteq U_k(s)$ for all $s \in \hat{S}$ by the construction of the algorithm. Since $p \in R_{k-1}(S_L) \cap R_{k-1}(S_R)$, the following lemma holds.

▶ **Lemma 16.** *Let S_L and S_R be two input sequences in the k-th recursive step. Then $\gamma(S_L, S_R)$ and $\tilde{\gamma}$ are the same in $R_{k-1}(S_L) \cap R_{k-1}(S_R)$.*

We show how to maintain $U_k(s)$ for all $s \in \hat{S}$ in a map $D_k(s)$. Each region in $D_k(s)$ has at most four sides. In the base case, we store the subdivision of $U_0(s)$ by the spokes of s in $\mathcal{D}_0(s)$ for each $s \in \hat{S}$. We construct $\mathcal{D}_k(s)$ from $\mathcal{D}_{k-1}(s)$ by shaving off the edges in $\mathcal{D}_{k-1}(s)$

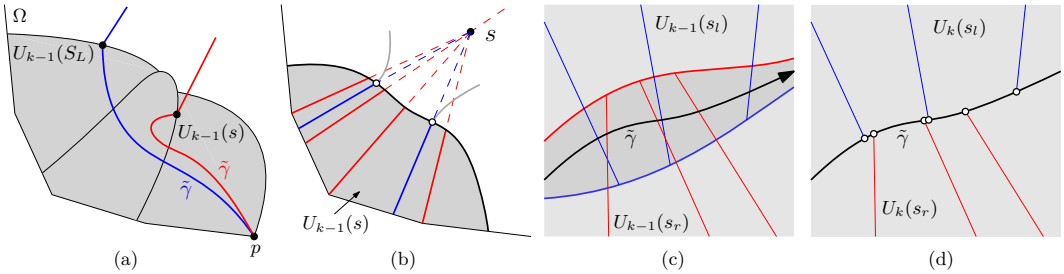


Figure 7 (a) $\tilde{\gamma}$ (blue) leaves $U_{k-1}(S_L)$ or $\tilde{\gamma}$ (red) visits $U_{k-1}(s)$ twice. (b) Edges of type (1) (red), edges of type (2) (thick black), and edges of type (3) (blue). (c) $U_{k-1}(s_l)$ and $U_{k-1}(s_r)$ for $s_l \in S_L$ and $s_r \in S_R$. Updating $U_{k-1}(s_l)$ and $U_{k-1}(s_r)$ along $\tilde{\gamma}$. (d) $U_k(s_l)$ and $U_k(s_r)$ after update.

along $\tilde{\gamma}$ from an endpoint $p \in \partial\Omega$. Each edge of $D_{k-1}(s)$ belongs to one of three types: (1) part of a spoke of s, (2) an edge connecting a Voronoi vertex and $\beta(s)$, and (3) part of $\partial U_{k-1}(s) \setminus \beta(s)$. See Figure 7(b). Observe that $\tilde{\gamma} \cap U_{k-1}(s)$ is connected and it is part of the boundary of $U_k(s)$. When $\tilde{\gamma}$ crosses an edge of type (1), we shave off the part of the edge not contained in $U_k(s)$. The resulting edge is incident to the intersection point. See Figure 7(c-d). Then we continue tracing $\tilde{\gamma}$. When $\tilde{\gamma}$ crosses an edge of type (2), we remove the edge and continue tracing $\tilde{\gamma}$. When $\tilde{\gamma}$ crosses an edge of type (3), we insert a Voronoi vertex v at the intersection and insert an edge $\chi(s, v) \cap U_k(s)$ of type (3). If $\tilde{\gamma}$ leaves $\mathcal{D}_{k-1}(s)$, we finish the construction of $\mathcal{D}_k(s)$ by removing the boundary curve of $\mathcal{D}_{k-1}(s)$ from p to v.

5.2.2 Analysis

We analyze the running time for the k-th recursive step. Since we already have $\mathsf{bFV}(S)$, we can find an endpoint of $\gamma(S_L, S_R)$ in $O(1)$ time. By Lemmas 15.3 and 16, $\gamma(S_L, S_R)$ in $U_{k-1}(S_L) \cap U_{k-1}(S_R)$ has complexity $\sum_{s \in S_L \cup S_R} O(n(s))$. Thus, $\bigcup_{s \in S_L \cup S_R} U_{k-1}(s)$ has complexity $\sum_{s \in S_L \cup S_R} O(n(s))$. The same $U_{k-1}(s)$ is visited $O(1)$ times. We can check whether γ meets the same $U_{k-1}(s)$ for the second time in $O(1)$ additional time. The ray shooting and partitioning take time linear to the complexity of each cell.

Now we analyze the running time of the algorithm. The recursion depth is $O(\log t) = O(\log \min\{m, n\})$ and $\sum_{s \in S} n(s) = O(m)$ by Lemma 9. The total complexity of $U_{k-1}(s)$ for sites s over all recursive calls at a fixed level k of recursion is $O(m)$. Thus, the algorithm runs in $O(m \log \min\{m, n\})$ time, after computing $\mathsf{bFV}(S)$ in $O(n \log n + n \log^2 m)$ time.

▶ **Theorem 17.** *The farthest-point Voronoi diagram of n point sites lying inside a convex m-gon in the Hilbert metric can be computed in $O(n(\log n + \log^2 m) + m \log n)$ time.*

References

1 Ahmed Abdelkader, Sunil Arya, Guilherme D. da Fonseca, and David M. Mount. Approximate nearest neighbor searching with non-Euclidean and weighted distances. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 355–372, 2019. URL: `https://dl.acm.org/doi/10.5555/3310435.3310458`.

2 Ahmed Abdelkader and David M. Mount. Economical Delone sets for approximating convex bodies. In *Proceedings of the 16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, volume 101, pages 4:1–4:12, 2018. `doi:10.4230/LIPICS.SWAT.2018.4`.

3 Ahmed Abdelkader and David M. Mount. Convex approximation and the Hilbert geometry. In *Proceedings of the 7th Symposium on Simplicity in Algorithms (SOSA 2024)*, pages 286–298, 2024. `doi:10.1137/1.9781611977936.26`.

4 Boris Aronov, Steven Fortune, and Gordon Wilfong. The furthest-site geodesic Voronoi diagram. *Discrete & Computational Geometry*, 9:217–255, 1993. doi:10.1007/BF02189321.

5 Rahul Arya, Sunil Arya, Guilherme D. da Fonseca, and David M. Mount. Optimal bound on the combinatorial complexity of approximating polytopes. *ACM Transactions on Algorithms*, 18(4):1–29, 2022. doi:10.1145/3559106.

6 Sunil Arya, Guilherme D. da Fonseca, and David M. Mount. Near-optimal ϵ-kernel construction and related problems. In *Proceedings of 33rd International Symposium on Computational Geometry (SoCG 2017)*, volume 77, pages 10:1–10:15, 2017. doi:10.4230/LIPIcs.SoCG.2017.10.

7 Sunil Arya, Guilherme D. da Fonseca, and David M. Mount. On the combinatorial complexity of approximating polytopes. *Discrete & Computational Geometry*, 58:849–870, 2017. doi:10.1007/S00454-016-9856-5.

8 Sunil Arya, Guilherme D. da Fonseca, and David M. Mount. Optimal approximate polytope membership. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 270–288, 2017.

9 Madeline Bumpus, Xufeng Caesar Dai, Auguste H. Gezalyan, Sam Munoz, Renita Santhoshkumar, Songyu Ye, and David M. Mount. Analysis of dynamic Voronoi diagrams in the Hilbert metric, 2024. arXiv:2304.02745.

10 L. Paul Chew and Robert L. Dyrsdale III. Voronoi diagrams based on convex distance functions. In *Proceedings of the 1st Annual Symposium on Computational geometry (SoCG 1985)*, pages 235–244, 1985.

11 Mark De Berg, Otfried Cheong, Marc Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer Berlin, Heidelberg, 3rd edition, 2008.

12 Friedrich Eisenbrand, Nicolai Hähnle, and Martin Niemeier. Covering cubes and the closest vector problem. In *Proceedings of the 27th Annual Symposium on Computational geometry (SoCG 2011)*, pages 417–423, 2011. doi:10.1145/1998196.1998264.

13 Friedrich Eisenbrand and Moritz Venzin. Approximate CVP_p in time $2^{0.802n}$. *Journal of Computer and System Sciences*, 124:129–139, 2022. doi:10.1016/J.JCSS.2021.09.006.

14 Auguste H. Gezalyan, Soo H. Kim, Carlos Lopez, Daniel Skora, Zofia Stefankovic, and David M. Mount. Delaunay triangulations in the Hilbert metric. In *Proceedings of the 19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024)*, volume 294, pages 25:1–25:17, 2024. doi:10.4230/LIPICS.SWAT.2024.25.

15 Auguste H. Gezalyan and David M. Mount. Voronoi diagrams in the Hilbert metric. In *Proceedings of the 39th International Symposium on Computational Geometry (SoCG 2023)*, volume 258, pages 35:1–35:16, 2023. doi:10.4230/LIPICS.SOCG.2023.35.

16 David Hilbert. Ueber die gerade Linie als kürzeste Verbindung zweier Punkte. *Mathematische Annalen*, 46(1):91–96, 1895.

17 Rolf Klein. Abstract Voronoi diagrams and their applications. In *Proceedings of Computational Geometry and its Applications (CG 1988)*, pages 148–157, 1988. doi:10.1007/3-540-50335-8_31.

18 Der-Tsai Lee. Two-dimensional Voronoi diagrams in the L_p-metric. *Journal of the ACM*, 27(4):604–618, 1980. doi:10.1145/322217.322219.

19 Márton Naszódi and Moritz Venzin. Covering convex bodies and the closest vector problem. *Discrete & Computational Geometry*, 67(4):1191–1210, 2022. doi:10.1007/S00454-022-00392-X.

20 Frank Nielsen and Laetitia Shao. On balls in a Hilbert polygonal geometry. In *Proceedings of the 33rd International Symposium on Computational Geometry (SoCG 2017)*, volume 77, pages 67:1–67:4, 2017. doi:10.4230/LIPIcs.SoCG.2017.67.

21 Athanase Papadopoulos and Marc Troyanov. *Handbook of Hilbert geometry*. European Mathematical Society, 2014.

22 Evantha Papadopoulou and D. T. Lee. The L_∞ Voronoi diagram of segments and VLSI applications. *International Journal of Computational Geometry & Applications*, 11(05):503–528, 2001.

23 Thomas Rothvoss and Moritz Venzin. Approximate CVP in time $2^{0.802n}$ – Now in any norm! In *Proceedings of the 23rd International Conference on Integer Programming and Combinatorial Optimization (IPCO 2022)*, pages 440–453, 2022.

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

On the I/O Complexity of the Cocke-Younger-Kasami Algorithm and of a Family of Related Dynamic Programming Algorithms

Lorenzo De Stefani[1]
Department of Computer Science, Brown University, Providence, RI, USA

Vedant Gupta
Department of Computer Science, Brown University, Providence, RI, USA

Abstract

Asymptotically tight lower bounds are derived for the Input/Output (I/O) complexity of a class of dynamic programming algorithms, including matrix chain multiplication, optimal polygon triangulation, and the construction of optimal binary search trees. Assuming no recomputation of intermediate values, we establish an $\Omega\left(\frac{n^3}{\sqrt{M}B}\right)$ I/O lower bound, where n denotes the size of the input and M denotes the size of the available fast memory (cache). When recomputation is allowed, we show that the same bound holds for $M < cn$, where c is a positive constant. In the case where $M \geq 2n$, we show an $\Omega(n/B)$ I/O lower bound. We also discuss algorithms for which the number of executed I/O operations matches asymptotically each of the presented lower bounds, which are thus asymptotically tight.

Additionally, we refine our general method to obtain a lower bound for the I/O complexity of the Cocke-Younger-Kasami algorithm, where the size of the grammar impacts the I/O complexity. An upper bound with asymptotically matching performance in many cases is also provided.

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

Keywords and phrases I/O complexity, Dynamic Programming Algorithms, Lower Bounds, Recomputation, Cocke-Younger-Kasami

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.49

Related Version *Full Version*: `https://arxiv.org/abs/2410.20337` [21]

1 Introduction

The performance of computing systems is significantly influenced by data movement, affecting both time and energy consumption. This ongoing technological trend [37] is expected to continue as fundamental limits on minimum device size and maximum message speed lead to inherent costs associated with data transfer, be it across the levels of a hierarchical memory system or between processing elements in a parallel system [10]. While the communication requirements of algorithms have been extensively studied, deriving significant lower bounds based on these requirements and matching upper bounds remains a critical challenge.

Dynamic Programming (DP) is a classic algorithmic framework introduced by Bellman [4] that is particularly effective in optimization tasks where problems can be broken down into overlapping subproblems with optimal substructure and storing the solutions to these subproblems to prevent redundant work. DP is widely used in computer science and other important fields such as control theory, operations research, and computational biology.

[1] Corresponding author

© Lorenzo De Stefani and Vedant Gupta;
licensed under Creative Commons License CC-BY 4.0

In this work, we analyze the memory access cost (*I/O complexity*) of a family of DP algorithms when executed in a two-level storage hierarchy with M words of fast memory (cache). The analyzed algorithms share a similar structure of dependence on the results of subproblems and include, among others, the classic DP algorithms for matrix chain multiplication, boolean parenthesization, optimal polygon triangulation, and construction of optimal binary search trees. We obtain asymptotically tight lower bounds for the I/O complexity of these algorithms both for schedules in which no intermediate values are computed more than once and for general schedules allowing recomputation. Our analysis reveals that, for a range of values of the size of the cache, recomputing intermediate values can lead to an asymptotically lower number of I/O operations. We further refine our analysis to derive an I/O lower bound for the Cocke-Younger-Kasami (CYK) algorithm [20, 47, 32] and an (almost) asymptotically matching upper bound.

Previous and related work. I/O complexity was introduced in the seminal work by Hong and Kung [31]; it denotes the number of data transfers between the two levels of a memory hierarchy with a cache of M words and a slow memory of unbounded size. Hong and Kung presented techniques to develop lower bounds for the I/O complexity of computations modeled by *Computational Directed Acyclic Graphs* (CDAGs).

While the "*edge expansion technique*" of [3], the "*path routing technique*" of [40], and the "*closed dichotomy width*" technique of [11] all yield I/O lower bounds that apply only to computational schedules for which no intermediate result is ever computed more than once (*nr-computations*) it is also important to investigate what can be achieved with recomputation. In fact, for some CDAGs, recomputing intermediate values reduces the space and/or the I/O complexity of an algorithm [39]. In [8], it is shown that some algorithms admit a *portable schedule* (i.e., a schedule that achieves optimal performance across memory hierarchies with different access costs) only if recomputation is allowed. Recomputation can also enhance the performance of simulations among networks (see [34] and references therein) and plays a key role in the design of efficient area-universal VLSI architectures. Several lower-bound techniques that allow recomputation have been presented in the literature, including the "*S-partition* technique" [31], the "*S-span* technique" [39], the "*S-covering* technique" [9] (which merges and extends aspects from both [31] and [39]), the "*G-flow* technique" [5, 6], and the "*visit partition technique* [7]. However, to the best of our knowledge, none of these have been previously applied to the family of DP algorithms considered in this work.

A systematic study of *Dynamic Programming* was introduced by Bellman [4], and has since become a popular problem-solving methodology. Galil and Park [27] produce a survey and classifications of popular dynamic programming algorithms and optimization methods. Several results have been presented in the literature aimed at optimizing the running time of Dynamic Programming algorithms [28, 33, 46, 27]. Notably, Valiant [45] showed that context-free recognition can be carried out as fast as Boolean matrix multiplication.

Cherng and Ladner [13] provide a cache-oblivious divide-and-conquer algorithm and a cache-aware blocked algorithm for *simple dynamic programs*, which includes matrix chain multiplication and the construction of optimal binary search trees. Their approach is based on Valiant's algorithm [45] and incurs $\mathcal{O}(\frac{n^3}{B\sqrt{M}})$ cache misses. In [16, 17], Choudhury and Ramachandran introduce the *Gaussian Elimination Paradigm (GEP)* that provides cache-oblivious algorithms for problems including all-pairs shortest path. The authors also provide a way to transform simple DP problems into GEP problems. A proof of the optimality of the proposed algorithm for GEP problems is provided based on a similar result given in [31] for the matrix multiplication problem. As we show, this result does not extend to simple

DP algorithms, as fewer cache misses can be achieved in some scenarios. Cache-oblivious DP algorithms for other problems have been explored [36, 19, 22]. There has also been work on parallelizing cache-efficient DP algorithms [18, 14, 30, 44, 43, 25, 23, 12], experimental evaluations of the resulting performance improvement [38], and on the automated discovery of cache-efficient DP algorithms [15, 29].

Dunlop et al. [24] provide an empirical analysis of various strategies to reduce the number of accesses to the grammar for the *CYK* algorithm. There have also been other works on modifying a grammar representation to improve efficiency. Notably, Song et al. [42] provides an empirical analysis of different grammar binarization approaches to improve efficiency, while Lange et al. [35] argue for the merits of a modified version of CYK to work with grammars in *binary normal form* instead of *Chomsky normal form*.

Summary of results. In this work, we analyze the I/O complexity of a family of DP algorithms that follow the general strategy outlined in *Prototype Algorithm* 1 when executed in a two-level storage hierarchy with M words of cache. The analyzed algorithms share a similar structure of dependence on the results of subproblems while allowing for diverse implementation choices and optimal problem substructure.

We analyze both schedules in which no intermediate values are computed more than once (no-recomputation schedules) and general ones that allow for recomputation. For both settings, we derive an $\Omega\left(\frac{n^3}{\sqrt{M}B}\right)$ lower bound to the number of I/O operations required by sequential execution of these algorithms on an input of size n in a two-level storage hierarchy with a cache of size $M \leq cn$, for some positive constant value c and such that $B \geq 1$ memory words can be moved between consecutive memory locations of the cache and of the slow memory with a single I/O operation. However, we show that while no-recomputation schedules require $\Omega\left(\frac{n^3}{\sqrt{M}B}\right)$ I/O operations even when using a cache of size at least $2n$ and $o\left(n^2\right)$, schedules using recomputation achieve an asymptotic reduction of the number of required I/O operations by a factor $\Theta\left(n^2/\sqrt{M}\right)$. This is particularly significant as in many cases of interest (*e.g.,* Matrix Multiplication [2, 5, 31], Fast Fourier Transform [31], Integer Multiplication [6]) recomputation has shown to enable a reduction of the I/O cost by at most a constant multiplicative factor. These results are obtained by analyzing a representation of the considered algorithms as Computational Directed Acyclic Graphs (CDAGs) whose structure captures the structure of dependence between subproblems, which is common to the DP algorithms considered in our work. We provide a general construction of such CDAGs and analyze their internal connection properties.

We refine our general method to obtain an $\Omega\left(\frac{n^3\Gamma}{\sqrt{M}B}\right)$ lower bound for the I/O complexity of the Cocke-Younger-Kasami algorithm when deciding whether an input string of length n is a member of the language generated by a given Context-Free Grammar with Γ rules with distinct right-hand-sides not including terminals. While the CYK algorithm exhibits a structure of subproblem dependencies similar to that of previously analyzed problems, it presents several challenging differences which we address by modifying our lower-bound methods by further refining its CDAG representation. Finally, we present a cache-oblivious implementation of the CYK algorithm for which, depending on the composition of the rules of the considered CFG, the number of I/O operations to be executed *almost* asymptotically matches the lower bound.

Algorithm 1 Prototype DP algorithm $\mathcal{A}^*$.

```
1: Input {x_0, x_1, ..., x_n}
2: Output S(1, n)
3: for i = 1 to n do                                   ▷ Initialization of subproblems S(i, i)
4:     S(i, i) ← INITIALIZATION_VALUE
5: for l = 2 to n do
6:     for i = 1 to n − l + 1 do
7:         j ← i + l − 1
8:         S(i, j) ← LEAST_OPTIMAL_VALUE               ▷ Initialization of accumulator subproblem S(i, j)
9:         for k = i to j − 1 do
10:            q ← COMBINE(S(i, k), S(k + 1, j))
11:            S(i, j) ← AGGREGATE(S(i, j), q)
12: return S(1, n)
```

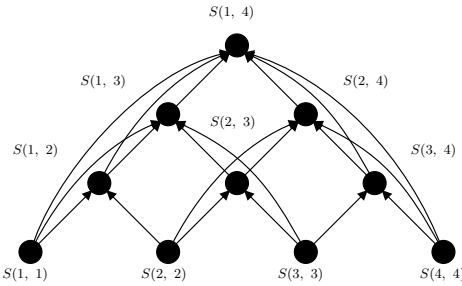
2 Preliminaries

We consider a family of Dynamic Programming (DP) algorithms following the general strategy outlined in *Prototype Algorithm* 1: Given an input of size n, the solution, denoted as $S(1, n)$ is computed bottom up. First, the values corresponding to the solution of subproblems of input size one, $S(i, i)$ for $i \in \{1, 2, \ldots, n\}$, are initialized to some set default value. The results of subproblems corresponding to parts of the input of growing size ℓ, that is $S(i, i + \ell)$ for $i \in \{1, 2, \ldots, n\}$ and $\ell \in \{1, 2, \ldots, n - i\}$, are computed by first evaluating the ℓ compositions of pairs of subproblems $S(i, i + k)$ and $S(i + k + 1, i + \ell)$, for $k \in \{0, 1, \ldots \ell - 1\}$ and then composing them according to the optimal subproblem structure used in the algorithm.

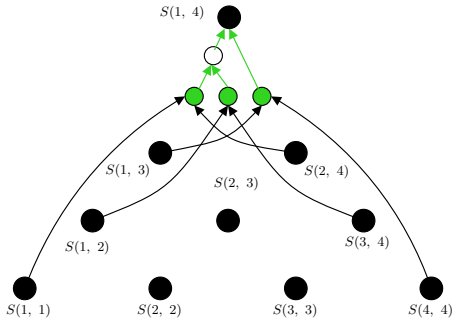
Depending on the specific problems, the default initialization values (`INITIALIZATION_VALUE` in the Prototype Algorithm) of $S(i, i)$, the way that the pairs $S(i, i + k)$ and $S(i+k+1, i+\ell)$ are combined (`COMBINE`), and the way these results are composed to evaluate $S(i, i + \ell)$ (`AGGREGATE`) may differ. However, `AGGREGATE` is assumed to be commutative and associative. Besides these differences, all the considered subproblems share the subproblem dependence structure outlined in the Prototype Algorithm, for which a visual representation is provided in Figure 1a. Examples of algorithms following such a structure include the classic DP algorithm for the matrix chain multiplication problem, the optimal convex polygon triangulation, the construction of optimal binary search trees, and the Cocke-Younger-Kasami algorithm (Algorithm 5).

We provide a general analysis of the I/O complexity of algorithms following the structure of the Prototype Algorithm $\mathcal{A}^*$, and we then refine our analysis for the mentioned algorithms of interest.

Computational Directed Acyclic Graphs. Our analysis is based on modeling the execution of the DP algorithms of interest as a *Computational Directed Acyclic Graph* (CDAG) $G = (V, E)$. Each vertex $v \in V$ represents either an input value or the result of an operation (i.e., an intermediate result or one of the output values) which is stored using a single memory word. The *directed* edges in E represent data dependencies. That is, vertices $u, v \in V$ are connected by an edge (u, v) directed from u to v if and only if the value corresponding to u is an operand of the unit time operation computing the value corresponding to v. Vertex u is said to be a *predecessor* of v and v is said to be a *successor* of u. For a given vertex v, the *set of its predecessors* (resp., *successors*) is defined as $pre(v) = \{u \in V \ s.t. (u, v) \in E\}$ (resp., $suc(v) = \{u \in V \ s.t. (v, u) \in E\}$. We refer to vertices with no predecessors (resp., successors) as the *input* (resp., *output*) vertices of a CDAG. We say that $G' = (V', E')$ is a *sub-CDAG* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$. We say that two sub-CDAGs $G' = (V', E')$ and $G'' = (V'', E'')$ of G are *vertex-disjoint* if $V' \cap V'' = \emptyset$.



(a) Representation of the subproblem dependency structure for the Prototype Algorithm and other DP algorithms considered in this work for an input of size $n = 4$.



(b) Construction of G from the simplified DAG. Root vertices are in black and leaves belonging to subproblem $S(1, 4)$ are in shown in green.

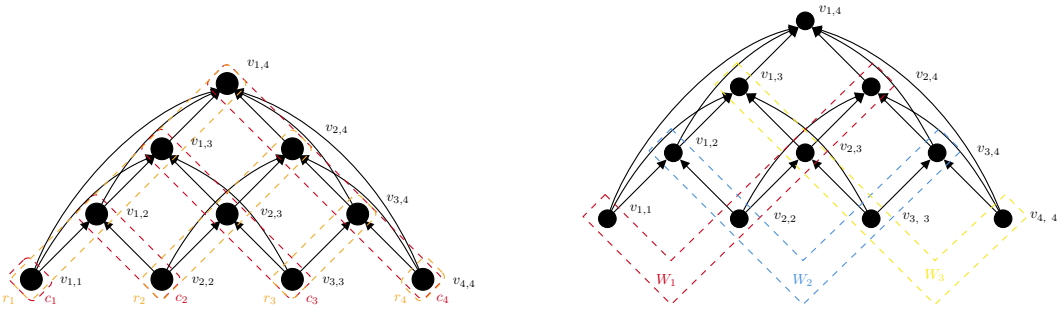
Figure 1 Construction of CDAG in $\mathcal{G}(n)$.

I/O model. We assume that sequential computations are executed on a system with a two-level memory hierarchy, consisting of a cache of size M (measured in memory words) and a *slow memory* of unlimited size. An operation can be executed only if all its operands are in the cache. We assume that the processor is equipped with standard elementary logic and algebraic operations. Data can be moved from the slow memory to the cache by `read` operations, and, in the other direction, by `write` operations. These operations are called *I/O operations*. We assume that the input data is stored in the slow memory at the beginning of the computation. The evaluation of a CDAG in this model can be analyzed using the "*red-blue pebble game*" [31]. The number of I/O operations executed when evaluating a CDAG depends on the "*computational schedule*", that is, on the order in which vertices are evaluated and on which values are kept in/discarded from the cache. The *I/O complexity* of a CDAG G corresponding to the execution of algorithm $\mathcal{A}$ for an input of size n, denoted as $IO_{\mathcal{A}}(M)$, is defined as the minimum number of I/O operations over all possible computational schedules. We further consider a generalization of this model known as the "*External Memory Model*" by Aggarwal and Vitter [1] where $B \geq 1$ memory words can be moved between consecutive memory locations of the cache and of the slow memory with a single I/O operation.

3 CDAG construction

In this section, we give the construction for a family of CDAGs representing the execution of a class of DP algorithms exhibiting a subproblem optimality structure as the one outlined for the Prototype Algorithm 1.

Given input size n, the CDAG G corresponding to the execution of $\mathcal{A}^*$ is constructed using the CDAG in Figure 1a as the base structure. The actual CDAG is obtained by refining the basic structure by replacing each vertex corresponding to a subproblem $S(i, j)$ with a directed binary tree DAG with $j - i$ leaves with all edges directed towards the root. The *root vertex* (R-vertex for short) of the tree DAG, henceforth referred to as $v_{i,j}$, corresponds to the computation of the solution of the associated subproblem $S(i, j)$. The k-th "*leaf vertex*" (L-vertex for short) of the tree, for $k \in \{0, \ldots, j - i - 1\}$ results from the combination of subproblems $S(i, i + k)$ and $S(i + k + 1, j)$ and has as predecessors the R-vertices of the tree DAGs corresponding respectively to $S(i, i + k)$ and $S(i + k + 1, j)$ (*i.e.*, $v_{i,i+k}$ and $v_{i+k+1,j}$) (a visual representation is provided in Figure 1b). Notice that binary trees corresponding to different R-vertices are vertex disjoint. For each such tree DAG, we say that its $j - i$ L-vertices *belong to the root* $v_{i,j}$.



(a) Rows and columns. **(b)** W-cover.

Figure 2 Representation of rows, columns, and the W-cover for input size $n = 4$.

Our lower bound results hold for *any possible structure* of these tree DAGs. This is a *feature* of our model meant to accommodate a variety of possible implementations for $\mathcal{A}^*$ and any other DP sharing the same substructure.

As an example, for the classic DP algorithm for the matrix chain multiplication problem (presented in the appendix of the extended online version [21]), the root vertex of each tree sub-CDAG $S(i, j)$ corresponds to the computation of the minimum number of operations required to compute the chain product of the matrices from the i-th to the j-th, and the k-th leaf, for $k \in \{0, \ldots, j-i-1\}$, corresponds to the computation of $S(i, i+k)+S(i+k+1, j)+d_{i-1}d_{i+k}d_j$.

We denote the family of CDAGs constructed in such a way with n input vertices and differing in tree sub-DAGs used to accumulate the values associated to L-vertices into the values associated with the corresponding R-vertex as $\mathcal{G}(n)$. This family of CDAGs captures the dependence structure between subproblems common to the family of DP algorithms that are the focus of our analysis.

Analysis of CDAG properties. Given a CDAG G from the family $\mathcal{G}(n)$, let R (resp. L) denote the set its of R-vertices (resp. L-vertices). Then, $|R| = \frac{n(n+1)}{2}$ and $|L| = \frac{n^3-n}{6}$.

The "*i-th row*" (resp., the "*j-th column*") of G, written r_i (resp., c_j), is defined as the subset of R including all of the R-vertices $v_{i,j}$ each corresponding to the computation of subproblem $S(i, j)$ for $j \in \{i, \ldots, n\}$ (resp., $i \in \{1, \ldots, j\}$). For any $v \in R$, if $v \in r_i$ and $v \in c_j$ then $i \leq j$. By construction, the rows $(r_1, r_2, \ldots, r_n)$ (resp., the columns $(c_1, c_2, \ldots, c_n)$) partition R, and $|r_i| = n - i + 1$ (resp., $|c_i| = i$) for $1 \leq i \leq n$. A visual representation of rows and columns can be found in Figure 2a.

By construction, each L-vertex has a distinct pair of R-vertices as its predecessors. Such a pair is said to *interact* in the computation. Given $v_{i,j} \in R$, it interacts with all and only the vertices $v_{k,i-1}$ for $1 \leq k \leq i-1$, and the vertices $v_{j+1,l}$ for $j+1 \leq l \leq n$. All such $n+i-j-1$ interactions correspond to the computation of a L-vertex belonging to a distinct sub-tree corresponding to the subproblem $S(k, j)$ or $S(i, l)$ (*i.e.,* to the R-vertex $v_{k,j}$ or $v_{i,l}$). Pairs of vertices in the same row (or column) do not interact.

We define $W_i = r_{i+1} \cup c_i$, for $1 \leq i \leq n-1$ as the subsets of R such that each vertex in c_i interacts with all the vertices of r_{i+1} (a visual representation can be found in Figure 2b). By construction, $|W_i| = n$. A vertex $v_{1,j} \in r_1$ (resp, $v_{i,n} \in c_n$) is only a member of W_j (resp., W_{i-1}). All remaining vertices $v_{i,j} \in R$ are members of both W_j and W_{i-1}, which are distinct as, by construction, for any $v_{i,j} \in R$ we have $i \leq j$. We refer to the collection of sets $(W_1, W_2, \ldots, W_{n-1})$ as the "*W-cover*" of the R-vertices of CDAG G. By its definition, and by the properties of the interactions between vertices according to placement in rows and columns discussed in the previous paragraphs, we have the following property:

▶ **Lemma 1.** *Given a CDAG $G \in \mathcal{G}(n)$, let $W_1, W_2, \ldots, W_{n-1}$ denote the W-cover of its R-vertices. For any set $X \subseteq W_i$ there are at most $\frac{|X|^2}{4} \leq \frac{n^2}{4}$ pairs of vertices (u, v), where $u \in r_{i+1}$ and $v \in c_i$, that interact as predecessors of distinct L-vertices each of which belongs to a distinct R-vertex.*

4 I/O analysis for computations of CDAGs in $\mathcal{G}(n)$

We analyze the I/O complexity of CDAGs in the class $\mathcal{G}(n)$ according to the *Red-Blue Pebble Game* by Hong and Kung [31]. To simplify the presentation, we assume that n and M are powers of 2, *i.e.,* $n = 2^i$ and $M = 2^j$ for integers $i, j \geq 3$. If that is not the case, our analysis can be adapted with minor, albeit tedious, modifications resulting in different constant terms.

4.1 I/O lower bound for computations with no recomputation

In this section, we present a lower bound to the I/O complexity of algorithms corresponding to CDAGs in $\mathcal{G}(n)$, assuming that no intermediate value is ever recomputed. (*i.e.,* under the "*no-recomputation assumption*"). We first introduce a useful technical Lemma:

▶ **Lemma 2.** *Given $n \in N^+$ non-negative real numbers $a_1, a_2, \ldots, a_n$ such that $\sum_{i=1}^{n} a_i^2 \geq T$, where $T \in \mathbb{R}^+$, and $\max_{i=1}^{n} a_i^2 \leq L$, where $L \in \mathbb{R}^+$ and $L < T$, then, $\sum_{i=1}^{n} a_i \geq \frac{T}{\sqrt{L}}$.*

Proof. The lower bound comes from the intuitive strategy of making each a_i as large as possible, i.e. if $q = \lfloor \frac{T}{L} \rfloor$, then assign $a_1, ..., a_q = \sqrt{L}$ and $a_{q+1} = \sqrt{T - qL}$. In this case, $\sum_{i=1}^{q+1} a_i = q\sqrt{L} + \sqrt{T - qL} = q\sqrt{L} + \sqrt{\frac{T}{L} - q}\sqrt{L} \geq q\sqrt{L} + (\frac{T}{L} - q)\sqrt{L}$ (as $\frac{T}{L} - q < 1$) $= \frac{T}{\sqrt{L}}$. To show the optimality of this strategy, we show that any other configuration is suboptimal.

Consider any other assignment of $a_1, ..., a_n$ that is not just a permutation of the same values. By assumption, any such assignment must have at least two non-zero values $a_i, a_j < \sqrt{L}$. Let $k = a_i + a_j$. If $k \leq \sqrt{L}$, since $k^2 > a_i^2 + a_j^2$, there exists some $c \in \mathbb{R}^+$ such that $c^2 = a_i^2 + a_j^2$ and $c < a_i + a_j$, showing that our assignment $a_1, ..., a_n$ does not minimize $\sum_{i=1}^{n} a_i$.

Alternatively, if $k > \sqrt{L}$, then $a_i^2 + a_j^2 = a_i^2 + (k - a_i)^2 < \sqrt{L}^2 + (k - \sqrt{L})^2$ (the last step follows from the fact that $f(x) = x^2 + (k - x)^2$ defines an upwards-facing parabola with an axis of symmetry about $x = \frac{k}{2}$). This means that there exists some $c < k - \sqrt{L}$ such that $\sqrt{L}^2 + c^2 = a^2 + b^2$ and $L + c < a + b$, again showing that our assignment $a_1, ..., a_n$ does not minimize $\sum_{i=1}^{n} a_i$. ◀

The following property captures the relation between subsets of L-vertices and their number of distinct R-vertex predecessors:

▶ **Lemma 3.** *Let $L' \subset L$ such that vertices in L' belong to the tree sub-CDAGs of at most $0 < \rho \leq |L'|$ distinct R-vertices. The set $R' \subseteq R$ containing all R-vertices that are predecessors of at least one vertex in L' has cardinality at least $\lfloor |L'|/\sqrt{\rho} \rfloor$.*

Proof. Consider the W-cover of G as defined in Section 3. As any R-vertex can be a member of at most two distinct W_i's we have:

$$|R'| \geq \frac{1}{2} \sum_{i=1}^{n-1} |R' \cap W_i|. \tag{1}$$

By definition, each vertex in L has a distinct pair of R-vertex predecessors which, by construction, are both included in exactly one set W_i. Let a_i be the number of vertices from L' with both predecessors in W_i. By the assumptions we have:

$$\sum_{i=1}^{n-1} a_i = |L'| \tag{2}$$

By Lemma 1, for any W_i, each interacting pair of R-vertices in it are the predecessors of a distinct L-vertex belonging to the tree sub-CDAG of a distinct R-vertex. From the assumption that vertices in L' belong to at most ρ distinct roots, we have:

$$\max_{i=1,\ldots,n-1} a_i \leq \rho, \tag{3}$$

Furthermore, from Lemma 1, we know that vertices in $|R' \cap W_i|$ can interact to produce at most $|R' \cap W_i|^2/4$ vertices in L'. Thus,

$$\sum_{i=1}^{n-1} \frac{|R' \cap W_i|^2}{4} \geq \sum_{i=1}^{n-1} a_i \Rightarrow \sum_{i=1}^{n-1} |R' \cap W_i| \geq \sum_{i=1}^{n-1} 2\sqrt{a_i} \tag{4}$$

By (1), (4), (2), (3), and Lemma 2 (in this order) we have:

$$|R'| \geq \frac{1}{2} \sum_{i=1}^{n-1} |R' \cap W_i| \geq \sum_{i=1}^{n-1} \sqrt{a_i} \geq \frac{|L'|}{\sqrt{\rho}}.$$ ◀

Lemma 3 allows us to claim that to compute a set of L-vertices, it is necessary to access a minimum number of R vertices, which, due to the non-recomputation assumption, must either be in the cache or loaded into it using a `read` I/O operation.

▶ **Theorem 4.** *For any CDAG $G = (V, E) \in \mathcal{G}(n)$ let $\mathcal{A}$ denote an algorithm corresponding to it. The I/O complexity of an Algorithm $\mathcal{A}$ when run without recomputing any intermediate value using a cache memory of size M and where for each I/O operation it is possible to move up to B memory words stored in consecutive memory locations from the cache to slow memory or vice versa, is:*

$$IO_G(n, M, B) \geq \max\left\{\frac{\left(n^3 - n\right)}{16\sqrt{M}} - \frac{n\left(n+1\right)}{2} - 3M, n\right\}\frac{1}{B} \tag{5}$$

Proof. We prove the result for $B = 1$. The result then trivially generalizes for a generic B.

The fact that $IO_G(n, M, 1) \geq n$ follows trivially from our assumption that the input is initially stored in the slow memory and therefore must be loaded into the cache at least once using n `read` I/O operations.

Let $\mathcal{C}$ be any computation schedule for the sequential execution of $\mathcal{A}$ using a cache of size M in which no value corresponding to any vertex of G is computed more than once. We partition $\mathcal{C}$ into non-overlapping segments $\mathcal{C}_1, \mathcal{C}_2, \ldots$ such that during each $\mathcal{C}_i$ exactly $8M^{1.5}$ values corresponding to distinct L-vertices, denoted as L_i, are evaluated. Since $|L| = \left(n^3 - n\right)/6$ there are $\left\lfloor \left(\frac{n^3}{6} - \frac{n}{6}\right) \frac{1}{8M^{1.5}} \right\rfloor$ such segments. Given L_i, we refer to the set of R-vertices to whom at least one of the vertices in L_i belongs as R_i. These correspond to vertices which are *active* during $\mathcal{C}_i$. For each interval, we denote as h_i the number of R-vertices whose value is completely evaluated during $\mathcal{C}_i$. As no value is computed more than once $\sum_i h_i = |R| = n\left(n+1\right)/2$. Let g_i denote the number of I/O operations executed during $\mathcal{C}_i$. We consider two cases:

- The vertices of L_i belong to at least $4M+1$ distinct R-vertices: For each of the $4M+1-h_i$ values corresponding to R-vertrices that were *partially computed* in $\mathcal{C}_i$, a partial accumulator of the values corresponding to their leaves computed during $\mathcal{C}_i$ must either be in the cache at the end of $\mathcal{C}_i$ or must be written into slow memory by means of a `write` I/O operation. Hence, $g_i \geq 4M+1-M-h_i$.
- The vertices of L_i belong to at most $4M$ distinct R-vertices: By Lemma 3, the vertices in L_i have at least $4M$ R-vertex predecessors, of whom at most h_i are computed during $\mathcal{C}_i$. As the considered schedule's values are not computed more than once, each of the remaining $4M-h_i$ values corresponding to predecessor vertices of L_i must either be in the cache at the beginning of $\mathcal{C}_i$, or read into the cache by means of a read I/O operation. Hence, $g_i \geq 4M-M-h_i$.

The I/O requirement of the algorithm is obtained by combining the cost of each non-overlapping segment:

$$IO_G(n,M,1) \geq \sum_{i=1}^{\left\lfloor \frac{n^3-n}{48M^{1.5}} \right\rfloor} g_i \geq \sum_{i=1}^{\left\lfloor \frac{n^3-n}{48M^{1.5}} \right\rfloor} (3M-h_i) \geq \left\lfloor \frac{n^3-n}{48M^{1.5}} \right\rfloor 3M - \sum_{i=1}^{\left\lfloor \frac{n^3-n}{48M^{1.5}} \right\rfloor} h_i$$
$$\geq \frac{(n^3-n)}{16\sqrt{M}} - 3M - |R|. \qquad \blacktriangleleft$$

By Theorem 4, we have that for $M \leq cn^2$, for a sufficiently small constant value $c > 0$, computations of CDAGs in $\mathcal{G}(n)$ in which no value is recomputed require $\Omega\left(\frac{n^3}{\sqrt{M}B}\right)$ I/O operations.

On the tightness of the bound

In [13], Cherng and Ladner presented the *Valiant's DP-Closure Algorithm*, henceforth referred to as `VDP`, which allows computing algorithms following the general structure outlined in Prototype Algorithm 1 *cache-obliviously* and without recomputation. When $M < |R| = \frac{n(n+1)}{2}$, `VDP` executes $\mathcal{O}\left(\frac{n^3}{\sqrt{M}B}\right)$ I/O operations, and for $M \geq |R|$ it executes $\mathcal{O}(n/B)$ I/O operations. Thus, we can conclude that our I/O lower bound for schedules with no recomputation provided in Theorem 4 is asymptotically tight.

`VDP` utilizes a divide and conquer approach, making recursive calls to smaller subproblems. Here we give a simplified presentation of the Valiant's DP-Closure Algorithm. Please refer to Cherng and Ladner [13] for a detailed presentation.

Consider any algorithm following the Prototype Algorithm 1 with input of size $n-1$, where we assume $n = 2^a$ for $a \in \mathbb{N}$. All subproblems $S(i,j)$ computed by the algorithm are associated with entries of an $n \times n$ matrix $\mathbf{X}$ so that $S(i,j)$ is represented at $\mathbf{X}[i,j+1]$. Initially, the values $X[i,i+1]$ corresponding to values $S(i,i)$ for $1 \leq i \leq n-1$ are computed from the input while the remaining entries in $\mathbf{X}$ are set to the `LEAST_OPTIMAL_VALUE` (*e.g.*, 0). To fully compute all the subproblems (*i.e.*, to compute the *DP-closure*), `VDP` splits $\mathbf{X}$ into 16 equally-sized submatrices of dimension $\frac{n}{4}$[2] labeled as follows:

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_{11} & \mathbf{X}_{12} & \mathbf{X}_{13} & \mathbf{X}_{14} \\ & \mathbf{X}_{22} & \mathbf{X}_{23} & \mathbf{X}_{24} \\ & & \mathbf{X}_{33} & \mathbf{X}_{34} \\ & & & \mathbf{X}_{44} \end{bmatrix}$$

[2] We assume that n is a power of 2. This can always be made the case by padding $\mathbf{X}$

We omit the representation of the lower-triangular submatrices as these will have all of their entries set to 0. Valiant's DP-Closure Algorithm (whose pseudocode is presented in Algorithm 2) is given the matrix $\mathbf{X}$ as input, with the only non-zero values corresponding to subproblems $S(i,i)$ for $1 \leq i \leq n-1$. The values of the $S(i,j)$ (*i.e.,* the R-vertices) corresponding to the top-left and bottom-right submatrices are computed through recursive calls, exploiting the fact that these submatrices correspond to a smaller instantiation of the same DP problem. The remaining values in $\mathbf{X}$ are then computed by invoking a subroutine called "*Valiant's Star Algorithm*" (`VSTAR`).

Algorithm 2 Valiant's DP-Closure Algorithm (`VDP`).

1: **Input** X
2: **Output** X ▷ DP-Closure of X is computed in-place
3: **if** $\dim(X) = 2$ **then** ▷ Input is already computed
4: **return**
5: $\begin{bmatrix} \mathbf{X}_{11} & \mathbf{X}_{12} \\ & \mathbf{X}_{22} \end{bmatrix} \leftarrow \texttt{VDP}\left(\begin{bmatrix} \mathbf{X}_{11} & \mathbf{X}_{12} \\ & \mathbf{X}_{22} \end{bmatrix}\right)$
6: $\begin{bmatrix} \mathbf{X}_{33} & \mathbf{X}_{34} \\ & \mathbf{X}_{44} \end{bmatrix} \leftarrow \texttt{VDP}\left(\begin{bmatrix} \mathbf{X}_{33} & \mathbf{X}_{34} \\ & \mathbf{X}_{44} \end{bmatrix}\right)$
7: $\mathbf{X} \leftarrow \texttt{VSTAR}(\mathbf{X})$

`VSTAR` is given as input a matrix $\mathbf{X}$ where the subproblems in the top-left and bottom-right quadrants are already computed. The algorithm then recursively computes the values corresponding to the remaining entries of $\mathbf{X}$ (*i.e.,* the values of the R-vertices). The pseudocode for this function can be found in Algorithm 3. We represent `COMBINE` with "$\cdot$" and `AGGREGATE` with "$+$". Operations involving these operators in Lines 6,8,10, and 11 are computed using a subroute called "*Matrix multiply and accumulate*" that, given three equally sized square matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, computes $A + B \cdot C$. In the context of `VDP`, $\mathbf{A}$ contains partially computed values $S(i,j)$'s (*i.e.,* corresponding to R-vertices), and $\mathbf{B} \cdot \mathbf{C}$ represents the computation of combinations of subproblems contributing to $S(i,j)$ (*i.e.,* $S(i,k) \cdot S(k+1,j)$ for $i \leq k < j$), that is, the values corresponding to the L-vertices, which are then used to update $\mathbf{A}$. A recursive in-place implementation of this function is presented in detail in [13, 26].

4.2 I/O lower bound for computations with allowed recomputation

We extend our I/O lower bound analysis to general computations in which values *may* be recomputed by combining it with the *dominator* I/O lower bound technique by Hong and Kung [31].

▶ **Definition 5** ([31]). *Given $G = (V, E)$, a set $D \subseteq V$ is a* dominator set *for $V' \subseteq V$ if every path directed from any input vertex of G to a vertex in V' contains a vertex of D.*

Dominator sets of R-vertices of a CDAG $G \in \mathcal{G}(n)$ must satisfy the following property:

▶ **Lemma 6.** *Given $G = (V, E) \in \mathcal{G}(n)$, let $A = \{v_{i_1,j_1}, v_{i_2,j_2}, \ldots\}$ be a subset of R-vertices where v_{i_k,j_k} is the R-vertex corresponding to subproblem $S(i_k, j_k)$. Any dominator set D of A has minimum cardinality $|D| \geq \min(|A|, \min j_k - i_k)$*

Algorithm 3 Valiant's Star Algorithm (`VSTAR`).

Input $\mathbf{X}$ ▷ Top-left and bottom-right are assumed to be fully computed
Output $\mathbf{X}$ ▷ DP-Closure of $\mathbf{X}$ is computed in-place
if $\dim(\mathbf{X}) = 2$ **then** ▷ No Computation required
  return
$\begin{bmatrix} \mathbf{X}_{22} & \mathbf{X}_{23} \\ & \mathbf{X}_{33} \end{bmatrix} \leftarrow \texttt{VSTAR}\left(\begin{bmatrix} \mathbf{X}_{22} & \mathbf{X}_{23} \\ & \mathbf{X}_{33} \end{bmatrix}\right)$
$X_{13} \leftarrow X_{13} + X_{12} \cdot X_{13}$
$\begin{bmatrix} \mathbf{X}_{11} & \mathbf{X}_{13} \\ & \mathbf{X}_{33} \end{bmatrix} \leftarrow \texttt{VSTAR}\left(\begin{bmatrix} \mathbf{X}_{11} & \mathbf{X}_{13} \\ & \mathbf{X}_{33} \end{bmatrix}\right)$
$X_{24} \leftarrow X_{24} + X_{23} \cdot X_{34}$
$\begin{bmatrix} \mathbf{X}_{22} & \mathbf{X}_{24} \\ & \mathbf{X}_{44} \end{bmatrix} \leftarrow \texttt{VSTAR}\left(\begin{bmatrix} \mathbf{X}_{22} & \mathbf{X}_{24} \\ & \mathbf{X}_{44} \end{bmatrix}\right)$
$X_{14} \leftarrow X_{14} + X_{12} \cdot X_{24}$
$X_{14} \leftarrow X_{14} + X_{13} \cdot X_{34}$
$\begin{bmatrix} \mathbf{X}_{11} & \mathbf{X}_{14} \\ & \mathbf{X}_{44} \end{bmatrix} \leftarrow \texttt{VSTAR}\left(\begin{bmatrix} \mathbf{X}_{11} & \mathbf{X}_{14} \\ & \mathbf{X}_{44} \end{bmatrix}\right)$

Proof. Suppose for every $v_{i,j} \in A$, D contains a vertex internal to the tree-CDAG rooted at v_{i_k,j_k}. As the tree-CDAGs of different R-vertices are disjoint, this would imply $|D| \geq |A|$.

Otherwise, there must be some $v_{i,j} \in A$, such that D does not contain any of its internal vertices. Let $L_{i,j} = \{l_0, l_1, \ldots, l_{j-i-1}\}$ denote the set of L-vertices corresponding to $v_{i,j}$. It is sufficient to show that there are $j - i$ vertex disjoint paths from the inputs of G to vertices in $L_{i,j}$. By construction, each leaf $l_k \in L_{i,j}$ has a predecessor belonging to a unique column c_{i+k}. In turn, each of these predecessors is connected through its internal tree-CDAG to the input vertex in the same column ($v_{i+k,i+k}$). In this manner, $j - i$ paths ($v_{i+k,i+k} \to v_{i,i+k} \to l_k$) for $0 \leq k \leq j - i - 1$ are obtained. Since each path contains R-vertices from a different column, and the tree-CDAGs of different R-vertices are disjoint, the paths outlined above are vertex-disjoint. ◀

▶ **Theorem 7.** *For any CDAG $G = (V, E) \in \mathcal{G}(n)$ let $\mathcal{A}$ denote an algorithm corresponding to it. The I/O-complexity of $\mathcal{A}$ when run using a cache memory of size M and where for each I/O operation it is possible to move up to B memory words stored in consecutive memory locations from cache to slow memory or vice versa, is:*

$$IO_G(n, M, B) \geq \max\left\{\frac{(n - 6M - 1)^3}{18\sqrt{M}} - 2M, n\right\}\frac{1}{B} \tag{6}$$

Proof. To simplify our analysis, we only consider *parsimonious execution schedules* such that each time an intermediate result is computed, the result is then used to compute at least one of the values of which it is an operand before being removed from the memory (either the cache or slow memory). Any non-parsimonious schedule $\mathcal{C}$ can be reduced to a parsimonious schedule $\mathcal{C}'$ by removing all the steps that violate the definition of parsimonious computation. $\mathcal{C}'$ has therefore less computational or I/O operations than $\mathcal{C}$. Hence, restricting the analysis to parsimonious computations leads to no loss of generality.

We prove the result for the case $B = 1$. The result then trivially generalizes for a generic B. The fact that $IO_{\mathcal{A}}(n, M, 1) \geq n$ follows trivially from our assumption that input is initially stored in slow memory and therefore must be loaded into the cache at least once using n `read` I/O operations.

Let $L(x)$ be the set of L-vertices in G whose predecessor R-vertices both correspond to some subproblems $S(i,j)$ such that $x \leq j-i$. By the construction, for each subproblem $S(i,j)$ the corresponding tree sub-CDAG has $\max\{0, j-i-2x\}$ such L-vertices. Since the sub-CDAGs corresponding to each $S(i,j)$ do not share vertices, we have:

$$\begin{aligned}
|L(x)| &= \sum_{j-i=1}^{n-1} (n-(j-i)) \max\{0, j-i-2x\} \\
&= \sum_{j-i=2x+1}^{n-1} (n-(j-i))(j-i-2x) \\
&= \sum_{y=1}^{n-1-2x} (n-2x-y)\, y \\
&= \frac{(n-2x-1)(n-2x)(n-2x+1)}{6} \\
&> \frac{(n-2x-1)^3}{6}.
\end{aligned}$$

Let $\mathcal{C}$ be any computation schedule for the sequential execution of $\mathcal{A}$ using a cache of size M. We partition $\mathcal{C}$ into segments $\mathcal{C}_1, \mathcal{C}_2, \ldots$ such that during each $\mathcal{C}_l$ exactly $6M^{1.5}$ values corresponding to *distinct vertices* in $L(3M)$ are computed from their operands (*i.e.,* not read from the cache). Let L_l denote the set of vertices corresponding to these values. Below we argue that the number g_l of I/O operations executed during each of the $\lfloor |L(3M)|/6M^{1.5} \rfloor$ non-overlapping segments $\mathcal{C}_l$ is at least $2M$, from whence the statement follows.

Let A denote the set of vertices corresponding to the values computed during $\mathcal{C}_l$. Clearly $L_l \subseteq A$. Let D denote the set of vertices corresponding to either the at most M values that are either stored in the cache or the values that are read into the cache during $\mathcal{C}_l$ by means of `read` I/O operations. Thus $g_l = |D| - M$. In order to be possible to compute the values corresponding to vertices in A during $\mathcal{C}_l$ there must be no paths from input vertices of G to vertices in A, that is, D must be a dominator set of A.

We refer to the set of R-vertices to whom at least one of the vertices in L_l belongs as R_l. These correspond to vertices which are *active* during $\mathcal{C}_l$. We consider three mutually exclusive cases:

(a) At least one of the values corresponding to a vertex v' in R_l is *entirely computed* during $\mathcal{C}_l$. That is no accumulator of the values corresponding to the leaves of v' is in the cache or it is loaded in it by means of an `read` I/O operation during $\mathcal{C}_l$. Thus, $v' \in A$ and D do not include any non-leaf vertex internal to the tree-sub-CDAG rooted in v'. Since $L_l \subseteq L(3M)$, by definition, v corresponds to a subproblem $S(i,j)$ such that $j - i \geq 3M$. From Lemma 6, $|D| \geq 3M$ which implies $g_j \geq 2M$.

(b) $|R_l| > 4M$: As none of the values in R_l is entirely computed during $\mathcal{C}_l$ (case (a)), and as we are considering parsimonious computations, for each value R_l at least one partial accumulator of its value is either in the cache at the beginning (resp., end) of $\mathcal{C}_l$ or is loaded into the cache during (resp., saved to the slow memory) during it. Hence $g_l \geq 4M - 2M$.

(c) $|R_l| \leq 4M$: By Lemma 3, the vertices in L_l have at least $3M$ distinct R-vertices predecessors. Since $L_l \subseteq L(3M)$ by definition, all such R vertices corresponds to subproblems $S(i,j)$ where $j - i \geq 3M$. As the values of these vertices are either in the cache at the beginning of $\mathcal{C}_l$, or `read` to the cache during $\mathcal{C}_l$, or computed during $\mathcal{C}_l$, D must be dominator set for the corresponding vertices. From Lemma 6, we can thus conclude $|D| \geq 3M$ from whence $g_l \geq 2M$. ◀

Algorithm 4 Top-Down recursive computation schedule for an algorithm $\mathcal{A}$ corresponding to $G \in \mathcal{G}(n)$.

Input $\{x_0, x_1, ..., x_n\}$
Output $S(1, n)$
for $i = 1$ **to** n **do** ▷ Initialization of subproblems $S(i, i)$
    $S(i, i) \leftarrow$ READ i-TH INPUT FROM SLOW MEMORY
procedure COMPUTE(i, j)
    if $i = j$ **then**
        return $S(i, j)$
    else
        $S(i, j) \leftarrow$ LEAST_OPTIMAL_VALUE ▷ Initialization of accumulator subproblem $S(i, j)$
        for $k = i$ **to** $j - 1$ **do**
            if $k - i \geq j - k - 1$ **then**
                $S(i, k) \leftarrow$ COMPUTE (i, k)
                $S(k + 1, j) \leftarrow$ COMPUTE $(k + 1, j)$
            else
                $S(k + 1, j) \leftarrow$ COMPUTE $(k + 1, k)$
                $S(i, k) \leftarrow$ COMPUTE (i, k)
            $S(i, j) \leftarrow$ AGGREGATE $(S(i, j),$ COMBINE $(S(i, k), S(k + 1, j)))$
    REMOVE FROM CACHE ALL VALUES USED BY THE PROCEDURE EXCEPT FOR $S(i, j)$
    return $S(i, j)$
return COMPUTE $(1, n)$

By Theorem 7, we have that for $M \leq c_1 n$, where c_1 is a sufficiently small positive constant value, computations of CDAGs in $\mathcal{G}(n)$ require $\Omega\left(\frac{n^3}{\sqrt{M}B}\right)$ I/O operations. This is in contrast with the result given in Theorem 4 for schedules with no recomputation that exhibit I/O complexity $\Omega\left(\frac{n^3}{\sqrt{M}B}\right)$ for values of M up to $c_2 n^2$ for an appropriately chosen constant c_2.

On the tightness of the bound. When *allowing recomputation*, if $M < c_1 n$, where c_1 is a sufficiently small positive constant, the number of I/O operations executed by Valiant's DP-Closure Algorithm asymptotically matches our lower bound in Theorem 7. Thus, for $M < c_1 n$, allowing recomputation does not asymptotically change the required number of I/O operations. On the other hand, if $M > 2n$, our lower bound simplifies to $\Omega(n/B)$, diverging from the complexity of Valiant's DP-closure Algorithm.

In Theorem 8, we show how for any algorithm corresponding to $G = \mathcal{G}(n)$ given a cache memory of suitable size $M \geq 2n$, there exists an execution schedule in which the results of subproblems are computed according to a recursive *top-down* order that executes only $\lceil n/B \rceil + 1$ I/O operations, thus asymptotically matching the general lower bound given in Theorem 7. Crucially, such a schedule involves the recomputation of intermediate results in order to reduce the number of I/O operations otherwise necessary, as established in Theorem 4.

▶ **Theorem 8.** *Using a cache memory of size $M \geq 2n$, there exists a computational schedule executing $\mathcal{A}^*$ incurring $\lceil n/B \rceil + 1$ I/O operations, where for each I/O operation it is possible to move up to B memory words stored in consecutive memory locations from the cache to slow memory or vice versa.*

Proof. The proposed schedule follows the steps outlined in Algorithm 4: in the initialization phase the n I/O values are read into the cache using $\lceil n/B \rceil$ `read` I/O operations. These values are held in the cache until the very end of the schedule, occupying n out of the $M \geq 2n$ available memory locations. Once the final result $S(1, n)$ is computed, it is written to the secondary memory by means of a `write` I/O operation.

We argue that the invocation of the procedure `COMPUTE`(i, n) uses at most n cache memory locations, thus not requiring the execution of any additional I/O operation. Let $\mathcal{M}(j-i)$ denote the memory usage of an invocation of `COMPUTE`(i, j). In the base case $\mathcal{M}(1) = 1$, as `COMPUTE` simply combines and returns the values $S(i, i)$ and $S(j, j)$ which are already stored in the cache after the initialization phase. In the general case, due to the use of accumulators and to the sequence of recursive calls and the memory management used by the schedule, we have:

$$\begin{aligned}\mathcal{M}(j-i) &= 1 + \max\{\mathcal{M}(j-i-1), 1 + \mathcal{M}(j-i-2))\} \\ &= 1 + \mathcal{M}(j-i-1) \\ &= j-i-1+\mathcal{M}(1) \\ &= j-i.\end{aligned}$$

The lemma follows as $\mathcal{M}(n-1) = n-1$. ◀

Interestingly, while the proposed computation schedule allows to achieve an asymptotic polynomial reduction of the number of I/O operations by a $\Theta\left(\frac{n^2}{\sqrt{M}}\right)$ multiplicative factor using recomputation, this is at the cost of a exponential increase of the number of computation operations executed ($2^{\mathcal{O}(n)}$), in comparison with Valiant's DP-Closure Algorithm which requires only $\mathcal{O}(n^3)$ computations.

4.3 I/O analysis for notable Dynamic Programming algorithms

The general result in Section 4 can be adapted to derive asymptotically tight bounds on the I/O complexity of classic Dynamic Programming algorithms for notable problems such as Matrix Chain Multiplication, Optimal Polygon Triangulation, and the construction of optimal binary search trees. This is achieved by mapping each of these algorithms onto Prototype Algorithm 1, defining the `COMBINE`, `AGGREGATE`, and `LEAST_OPTIMAL_VALUE` appropriately. Due to space limitations, these results are presented in the online full version of this work [21, Appendix A].

5 I/O analysis for the Cocke-Younger-Kasami algorithm

In this section, we present I/O bounds for the Cocke-Younger-Kasami algorithm (CYK). First, we provide some background on the algorithm. We then discuss how our lower bound analysis method must be enhanced to better capture the complexity of this algorithm. Finally, we present an upper bound.

5.1 Background on the CYK algorithm

▶ **Definition 9** ([41]). *A Context-Free Grammar is a 4-tuple* $(V, \mathcal{R}, \Sigma, T)$ *where*

- V *is a finite set called the* variables*;*
- Σ *is a finite set, disjoint from* V*, called the* terminals*;*
- $\mathcal{R}$ *is a finite relation in* $V \times (V \cup \Sigma \cup \epsilon)^*$*, where the* * *represents the Kleene star operation and* ϵ *denotes the empty string;*
- $T \in V$ *is the start variable.*

Elements of $\mathcal{R}$ are referred to as *rules* and are generally given in the form $A \to w$, where $A \in V$, and $w \in (V \cup \Sigma \cup \epsilon)^*$. If $u, v, w \in (V \cup \Sigma \cup \epsilon)^*$ and if $A \to w$ is a rule of the grammar, we say that uAv *yields* uwv, written as $uAv \Rightarrow uwv$. We say that u *derives* v, written $u \Rightarrow^* v$,

Algorithm 5 Psudocode for the Cocke-Younger-Kasami algorithm.

```
Input {w_1, ..., w_n}, (V, Σ, R, T)
Output parse
parse ← False
S ← False
if n = 0 then
    if (T → ϵ) ∈ R then
        parse ← True
    return
for i = 1 to n do
    for each unit production (A → a) ∈ R do
        if a = w_i then
            S(i, i)[A] ← True
for l = 2 to n do
    for i = 1 to n − l + 1 do
        j ← i + l − 1
        for k = i to j − 1 do
            for each binary production (A → BC) ∈ R do
                if S(i, k)[B] and S(k + 1, j)[C] then
                    S(i, j)[A] ← True
if T ∈ S(1, n) then
    parse ← True
```

if $u = v$ or if there exists a sequence $u_1, u_2, \ldots, u_k$ for $k \geq 1$ such that $u_i \in (V \cup \Sigma \cup \epsilon)^*$ and $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v$. The *language* of a CFG is the set of all strings $w \in \Sigma^*$ that can be derived from the starting variable T, that is $\{w \in \Sigma^* \; s.t. \; T \Rightarrow^* w\}$. A CFG is in Chomsky's Normal Form (CNF) if every rule is in the form: $A \to BC$, or $A \to a$, or $T \to \epsilon$, where A is any variable in V, B and C can be any pair in $(V \setminus S) \times (V \setminus S)$, and a can be any terminal in Σ. Any CFG can be transformed into an equivalent one in CNF through a simple iterative algorithm [41]. We refer to the number of rules $|\mathcal{R}|$ as the *size* of a CFG. We denote the set of binary rules $(A \to BC)$ in a CFG with $\mathcal{R}_B \subset \mathcal{R}$, the set of terminal rules $(A \to a)$ with $\mathcal{R}_T \subset \mathcal{R}$, and the number of distinct right-hand sides in $\mathcal{R}_B$ with Γ.

Given a CFG $(V, \mathcal{R}, \Sigma, T)$ in CNF, and a string $w = w_1 w_2 \ldots w_n$, the Cocke–Younger–Kasami (CYK) algorithm, henceforth referred to as $\mathcal{A}_{CYK}$, decides whether w is part of the language of the CFG. We refer to the standard implementation provided in Algorithm 5. Let $S(i, j)$ be the set of variables that derive the substring $w_i w_{i+1} \ldots w_j$. CYK uses a DP approach to compute all sets $S(i, j)$ for $1 \leq i \leq j \leq n$ using an adaptation of prototype algorithm $\mathcal{A}^*$. $\mathcal{A}_{CYK}$ determines whether w is a member of the language generated by the given CFG by checking whether $T \in S(1, n)$ (*i.e.,* whether T derives w).

Memory representation model. We assume that the variables in V can be referenced by index and that such index can be stored in a single memory word. We represent subsets of V (the results of subproblems $S(i, j)$) using a one-hot encoding with $\lceil |V|/s \rceil$ memory words, where s denotes the number of bits in a memory word. Thus, given the representation of one such subset in the memory, to check whether a variable V_i is a member of the subset it is sufficient to access the $(i \mod s)$-th bit of the $\lfloor i/s \rfloor$-th memory word.

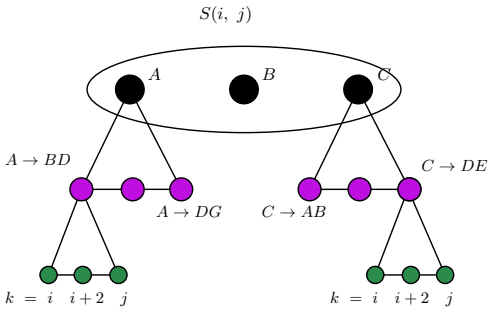


Figure 3 Each subproblem $S(i,j)$ is represented by multiple *variable roots* (in black), which is the result of the composition multiple *grammar roots* (in purple), which in turn are the composition of multiple *leaves* (in green).

We assume that each grammar rule is represented utilizing at most three memory words each used to represent the index of the left-hand side and at most two to represent the right-hand side (either the index of the two variables or the code for a terminal symbol).

5.2 I/O lower bound

While the CYK algorithm shares the underlying structure of relations between subproblems of the prototype algorithm $\mathcal{A}^*$, it presents several non-trivial complications due to the application of the grammar rules that, in turn, require a refinement of our analysis methods and, in particular, of the CDAG construction. In this section, we will assume that the set of grammar rules $\mathcal{R}$ are such that no two rules have the same right-hand side. While this might not always hold in practice, our analysis can still be applied by considering a subset of rules of the considered grammar which satisfies this condition.

CYK CDAG construction. The construction of the CDAG representing CYK's execution for an input string w with n characters and a CFG in CNF $(V, \mathcal{R}, \Sigma, T)$, called $G_{CYK}\,(n, (V, \mathcal{R}, \Sigma, T))$, follows a recursive structure which uses as a basis $G = (V, E) \in \mathcal{G}(n)$. We describe the necessary modification focusing on the sub-CDAG for a single $S(i,j)$ subproblem. A visual depiction of the ensuing description is provided in Figure 3:

- For the base of the construction, for each $S(i,i)$, the CDAG has $|V|/s$ vertices, each corresponding to the encoding of one of the possible variables of the grammar. These vertices serve as the input of the CDAG.
- Each R-vertex of G, corresponding to the computation of the solution of one of the subproblems $S(i,j)$, is replaced by $|V|/s$ vertices, each representing a part of the encoding of the set of variables V. We refer to this set of vertices as "*variable roots*" *VR*.
- Consider a variable root v corresponding to subproblem $S(i,j)$ and variable $A \in V$. v forms the root of a binary tree, which has as its leaves vertices for each rule having A as its left-hand side. We denote the set of these vertices as "*grammar roots*" *GR*. Note that as only one variable can appear on the left-hand side of a rule, the trees composed in this way are vertex disjoint. Further, for each $S(i,j))$ there will be at least Γ/s such vertices.
- For each subproblem $S(i,j)$, consider one of its grammar roots u corresponding to rule $\gamma \in \mathcal{R}_B$. u forms the root of a binary tree combining the results of $j-i$ "leaf-vertices". Each leaf vertex corresponds to a combination of the subproblems $S(i,k)$ and $S(k+1,j)$, for $i \leq k < j$. Such a vertex has as predecessors the variable root vertices of subproblems $S(i,k)$ and $S(k+1,j)$ encoding the variables appearing in the right-hand side of rule γ.

Given the CDAG $G_{CYK}(n, (V, \mathcal{R}, \Sigma, T))$, we generalize the notion of rows (resp., columns) introduced in Section 3 by having row r_i (resp. column c_j) include all vertices in VR corresponding to the encoding of the results of subproblems $S(i, j)$ for $1 \leq j \leq n$ (resp., for $1 \leq i \leq n$).The definition of W-cover generalizes to $G_{CYK}(n, (V, \mathcal{R}, \Sigma, T))$. We observe that a property analogous to that outlined in Lemma 1 holds for $G_{CYK}(n, (V, \mathcal{R}, \Sigma, T))$ as well:

▶ **Lemma 10.** *Given a CDAG $G_{CYK}(n, (V, \mathcal{R}, \Sigma, T))$, let $W_1, \ldots, W_{n-1}$ denote its W-cover. Any set of x VR-vertices in W_i contain at most $x^2/4$ interacting pairs, each of which forms the predecessors of a unique L-vertex belonging to a unique GR-vertex.*

Proof. Consider any set of x VR-vertices in W_i containing x_c vertices in column i and $x - x_c$ vertices in row $i + 1$. Since vertices in the same row (or column) don't interact, there are at most $x(x - x_c) \leq x^2/4$ interacting pairs.

Consider any two such distinct interacting pairs: $\left(v^A_{k,i}, v^B_{i+1,l}\right)$ and $\left(v^C_{m,i}, v^D_{i+1,n}\right)$ (where $v^A_{k,i}$ is a VR-vertex corresponding to variable A in row k and column i). Any leaf produced by $\left(v^A_{k,i}, v^B_{i+1,l}\right)$ must correspond to a rule with right-hand side "AB". From the assumption that every rule in $\mathcal{R}$ has a unique right-hand side, it follows that there is only one such rule and therefore $\left(v^A_{k,i}, v^B_{i+1,l}\right)$ must produce a single leaf.

To show that the leaves produced by $\left(v^A_{k,i}, v^B_{i+1,l}\right)$ and $\left(v^C_{m,i}, v^D_{i+1,n}\right)$ belong to distinct GR-vertices, consider the following two cases:

1. If $k \neq m$ or $l \neq i$, it follows from Lemma 1 that the leaves produced belong to different roots, and therefore must also belong to different grammar roots.
2. Otherwise, it must be the case that $A \neq C$ or $B \neq D$ (since the two interacting pairs are assumed to be distinct). It follows that the two leaves produced correspond to different rules in $\mathcal{R}$ and therefore belong to distinct grammar roots. ◀

I/O lower bound proof. We analyze the I/O complexity of any sequential computation for the CYK algorithm by analyzing the CDAG corresponding to the given CFG and the input string to be considered. We first state two results that correspond each to a modification of the results in Lemma 3 and, respectively, of Lemma 6 for the family of CDAG corresponding to execution of the CYK algorithm:

▶ **Lemma 11.** *Given a CDAG $G_{CYK}(n, (V, \mathcal{R}, \Sigma, T))$, let $L' \subset L$ be such that vertices in L' belong to the tree sub-DAGs of at most ρ distinct GR-vertices. The set $R' \subseteq VR$ that includes all VR-vertices that are predecessors of at least one vertex in L' has cardinality at least $|L'|/\sqrt{\rho}$.*

Proof. Consider the W-cover of $G_{CYK}(n, (V, \mathcal{R}, \Sigma, T))$ with the adapted definition for the CYK CDAG discussed in Section 5.2. As any VR-vertex can be a member of at most two distinct W_i's we have:

$$|R'| \geq \frac{1}{2} \sum_{i=1}^{n-1} |R' \cap W_i|. \tag{7}$$

By definition, each vertex in L has a distinct pair of VR-vertex predecessors which, by construction, are both included in exactly one set W_i. Let a_i be the number of vertices from L' with both predecessors in W_i. By the assumptions we have:

$$\sum_{i=1}^{n-1} a_i = |L'| \tag{8}$$

By Lemma 10, for any W_i, each interacting pair of VR-vertices in it are the predecessors of a distinct L-vertex belonging to the tree sub-CDAG of a distinct GR-vertex. From the assumption that vertices in L' belong to at most ρ distinct roots, we have:

$$\max_{i=1,\ldots,n-1} a_i \leq \rho, \tag{9}$$

Furthermore, from Lemma 10, we know that vertices in $|R' \cap W_i|$ can interact to produce at most $|R' \cap W_i|^2/4$ vertices in L'. Thus,

$$\sum_{i=1}^{n-1} \frac{|R' \cap W_i|^2}{4} \geq \sum_{i=1}^{n-1} a_i \Rightarrow \sum_{i=1}^{n-1} |R' \cap W_i| \geq \sum_{i=1}^{n-1} 2\sqrt{a_i} \tag{10}$$

By (7), (10), (8), (9), and Lemma 2 (in this order) we have:

$$|R'| \geq \frac{1}{2} \sum_{i=1}^{n-1} |R' \cap W_i| \geq \sum_{i=1}^{n-1} \sqrt{a_i} \geq \frac{|L'|}{\sqrt{\rho}}.$$ ◀

The proof follows a reasoning analogous to that used to prove Lemma 3 adapted for the CDAG $G_{CYK}(n, (V, \mathcal{R}, \Sigma, T))$ by using the property of L-vertices of the CYK CDAG states in Lemma 10 in place of Lemma 1.

▶ **Lemma 12.** *Given $G_{CYK}(n, (V, \mathcal{R}, \Sigma, T))$, let $A = \{v_{i_1,j_1}, v_{i_2,j_2}, \ldots\}$ be a subset of GR-vertices where v_{i_l,j_k} is a GR-vertex corresponding to subproblem $S(i_k, j_k)$. Any dominator set D of A has cardinality at least $|D| \geq \min(|A|, \min j_k - i_k)$.*

The proof corresponds to the one of Lemma 6 where the tree sub-CDAGs rooted in GR-vertices are considered instead of the R vertices of $\mathcal{G}(n)$.

A lower bound to the I/O complexity of the CYK algorithm can be obtained by following steps analogous to those used in the proof of Theorem 7 with opportune adjustments matching the modifications in the *enhanced* CDAG described in this section.

▶ **Theorem 13.** *Consider a string w of length n and a CFG $(V, \mathcal{R}, \Sigma, T)$ in CNF. The number of I/O operations executed by $\mathcal{A}_{CYK}$ when deciding whether w is a member of the language of the given grammar using a machine equipped with a cache memory of size M and such that B contiguous can be moved with each I/O operation is bounded as:*

$$IO_{G_{CYK}}(n, m, B) \geq \max\left\{\frac{(n - 6M - 1)^3}{18\sqrt{M}}\Gamma - 2M\Gamma, n\right\}\frac{1}{B}$$

where Γ denotes the number of distinct right-hand sides in $\mathcal{R}_B$.

By Theorem 13, we have that for $M \leq c_1 n$, where c_1 is a sufficiently small positive constant value, computations of the CYK algorithm require $\Omega\left(\frac{n^3\Gamma}{\sqrt{M}B}\right)$ I/O operations. Note that if the considered CFG is such that $\Gamma = \Theta(|\mathcal{R}_B|)$ our lower bound shows a direct dependence of the I/O complexity of execution of the CYK algorithm on the *size* of the considered grammar. Indeed there are many possible grammars for which this is the case. Finally, while the result of Theorem 7 holds for computations with recomputation, it is also possible to obtain an I/O lower bound for schedules in which recomputation is not allowed by modifying the analysis given in Theorem 4.

5.3 On the tightness of the bounds

We refine Valiant's DP-Closure Algorithm [13] to obtain an I/O efficient recursive divide-and-conquer implementation of the CYK algorithm, henceforth referred to as $\mathcal{A}_{CYK}$. While the CYK algorithm follows the general structure of the Prototype Algorithm 1, and is hence amenable to the approach in [13], the analysis in [13] does not directly apply to the case in which the size of the given CNF cannot be considered as constant with respect to the size of the cache. Our algorithm, $\mathcal{A}_{CYK}$, manages I/O efficiently by opportunely iterating over the grammar rules and by removing redundant computations due to grammar rules with the same right-hand side.

Given an input string w of length $n-1$ and a CFG $(V, \mathcal{R}, \Sigma, T)$ in CNF, we use an $n \times n$ matrix $\mathbf{X}$ to represent the the values of the subproblems computed by the CYK algorithm (corresponding to R-vertices as defined in Section 5.2): $\mathbf{X}[i, j+1]$ corresponds to the solution of subproblem $S(i, j)$, *i.e.,* a sequence of length $|V|$ with binary values, where the index corresponding to variable $A \in V$ represents whether variable A can derive the substring $w_i w_{i+1} \dots w_j$ of the input.

First, $\mathcal{A}_{CYK}$ computes the input values corresponding to subproblems $S(i, i)$ for $1 \leq i \leq n$ (*i.e.,* the values of $\mathbf{X}[i, i+1]$): Each character in the input string w is compared with the right-hand side of each terminal rule in $\mathcal{R}_T$, and the appropriate values are stored. The rest of the computation involves executing Valiant's DP-Closure Algorithm (Algorithm 2) using a modified version of Valiant's Star Algorithm (Algorithm 3, called "*Valiant's Star Algorithm for CYK*" ($\mathcal{A}^*_{CYK}$) which is presented in Algorithm 6.

Algorithm 6 $\mathcal{A}^*_{CYK}$, Valiant's Star Algorithm for CYK.

1: **Input** X ▷ Top-left and bottom-right are assumed to be fully computed
2: **Output** X ▷ DP-Closure of X is computed in-place
3: **if** $\dim(X) = 2$ **then** ▷ Root has been fully computed. Update all variable roots
4: **for** $A \in V$ matched with placeholder P **do**
5: $X^A \leftarrow X^P$
6: **return**
7: $\begin{bmatrix} \mathbf{X}_{22} & \mathbf{X}_{23} \\ & \mathbf{X}_{33} \end{bmatrix} \leftarrow \mathcal{A}_{CYK^*}\left(\begin{bmatrix} \mathbf{X}_{22} & \mathbf{X}_{23} \\ & \mathbf{X}_{33} \end{bmatrix}\right)$
8: **for** $P \to BC$ **in** $\mathcal{R}^{\Gamma}_B$ **do**
9: $X^P_{13} \leftarrow X^P_{13} + X^B_{12} \cdot X^C_{13}$ ▷ Computed using `Matrix Multiply and Accumulate`
10: $\begin{bmatrix} \mathbf{X}_{11} & \mathbf{X}_{13} \\ & \mathbf{X}_{33} \end{bmatrix} \leftarrow \mathcal{A}_{CYK^*}\left(\begin{bmatrix} \mathbf{X}_{11} & \mathbf{X}_{13} \\ & \mathbf{X}_{33} \end{bmatrix}\right)$
11: **for** $P \to BC$ **in** $\mathcal{R}^{\Gamma}_B$ **do**
12: $X^P_{24} \leftarrow X^P_{24} + X^B_{23} \cdot X^C_{34}$
13: $\begin{bmatrix} \mathbf{X}_{22} & \mathbf{X}_{24} \\ & \mathbf{X}_{44} \end{bmatrix} \leftarrow \mathcal{A}_{CYK^*}\left(\begin{bmatrix} \mathbf{X}_{22} & \mathbf{X}_{24} \\ & \mathbf{X}_{44} \end{bmatrix}\right)$
14: **for** $P \to BC$ **in** $\mathcal{R}^{\Gamma}_B$ **do**
15: $X^P_{14} \leftarrow X^P_{14} + X^B_{12} \cdot X^C_{24}$
16: $X^P_{14} \leftarrow X^P_{14} + X^B_{13} \cdot X^C_{34}$
17: $\begin{bmatrix} \mathbf{X}_{11} & \mathbf{X}_{14} \\ & \mathbf{X}_{44} \end{bmatrix} \leftarrow \mathcal{A}_{CYK^*}\left(\begin{bmatrix} \mathbf{X}_{11} & \mathbf{X}_{14} \\ & \mathbf{X}_{44} \end{bmatrix}\right)$

$\mathcal{A}^*_{CYK}$ introduces two modifications over Valiant's Star Algorithm: First, the algorithm iterates over the rules of the grammar with distinct right-hand-sides in lines 8, 11, and 14, calling the subroutine Matrix Multiply and Accumulate with the variable roots corresponding to each rule in the grammar. Second, to avoid computing rules with identical right-hand sides more than once, let $\mathcal{R}^{\Gamma}_B$ denote a set of binary rules each having a unique right-hand side from $\mathcal{R}_B$ of maximum cardinality. In this set, the corresponding left-hand sides are replaced with placeholder variables. For any subproblem, once the value corresponding to a

rule in $\mathcal{R}_B^\Gamma$ is fully computed, it can be used to update the value corresponding to all variable roots producing the same right-hand side through a rule in $\mathcal{R}_B$. This is done in lines 4 and 5 as the base case implies that all leaves for the subproblem have been computed.

▶ **Theorem 14.** *Given an input string w of length n and a CFG $(V, \mathcal{R}, \Sigma, T)$ in CNF, algorithm $\mathcal{A}_{CYK}$ decides whether w is a member of the language of the given grammar. When run on a machine equipped with a cache of size M, the number of I/O operations executed by $\mathcal{A}_{CYK}$ can be bound as:*

$$IO_{\mathcal{A}_{CYK}}(n, M) \leq \mathcal{O}\left(\left(\frac{n^3\Gamma}{\sqrt{M}} + n^2\Gamma \log M + n^2|\mathcal{R}_B| + n|\mathcal{R}_T|\right)\frac{1}{B}\right).$$

Proof. When computing $S(i, i)$ for $1 \leq i \leq n$, $\mathcal{A}_{CYK}$ compares each character of input string w with the right-hand side of each terminal rule in $\mathcal{R}_T$. Doing so incurs at most $O(n|\mathcal{R}|_T/B)$ I/O operations.

In the following, we use n to denote the dimension of the input (*i.e.,* $n = dim(X)$). The number of I/O operations executed by Valiant's Star Algorithm for CYK is at most:

$$IO_{\mathcal{A}_{CYK^*}}(n) \leq \begin{cases} \mathcal{O}\left(\frac{|\mathcal{R}_B|}{B}\right) & \text{if } n \leq 2, \\ 4IO_{\mathcal{A}_{CYK^*}}\left(\frac{n}{2}\right) + 4\Gamma(IO_{MMA}\left(\frac{n}{4}\right) + O(1)) & \text{otherwise} \end{cases} \tag{11}$$

By [13, Lemma 2.1], the number of I/O operations executed by `Matrix Multiply and Accumulate` is:

$$IO_{MMA}(n) \leq \begin{cases} \mathcal{O}(\frac{n^2}{B}) & \text{if } n^2 \leq M, \\ \mathcal{O}(\frac{n^3}{B\sqrt{M}}) & \text{otherwise} \end{cases} \tag{12}$$

Solving the recursion in (11) using (12) yields:

$$\begin{aligned} IO_{\mathcal{A}_{CYK^*}}(n) &\leq \mathcal{O}\left(\frac{n^3\Gamma}{B\sqrt{M}}\right) + 4^{\log(n/\sqrt{M})}\mathcal{O}\left(\frac{M\Gamma}{B}\log M + \frac{M|\mathcal{R}_B|}{B}\right) \\ &\leq \mathcal{O}\left(\frac{n^3\Gamma}{B\sqrt{M}} + \frac{n^2\Gamma}{B}\log M + \frac{n^2|\mathcal{R}_B|}{B}\right) \end{aligned} \tag{13}$$

Finally, by adapting that analysis in [13] we can bound the number of I/O operations executed by the proposed algorithm $\mathcal{A}_{CYK}$ as:

$$IO_{\mathcal{A}_{CYK}}(n) \leq \begin{cases} \mathcal{O}(1) & \text{if } n \leq 2, \\ 2IO_{\mathcal{A}_{CYK}}(n/2) + IO_{\mathcal{A}^*_{CYK}}(n) + \mathcal{O}(1) & \text{otherwise} \end{cases}$$

Expanding the recurrence, by (13), and adding the cost of processing the input, we have:

$$IO_{\mathcal{A}_{CYK}}(n) \leq \mathcal{O}\left(\frac{n^3\Gamma}{B\sqrt{M}} + \frac{n^2\Gamma}{B}\log M + \frac{n^2|\mathcal{R}_B|}{B} + \frac{n|\mathcal{R}_T|}{B}\right)$$ ◀

When $M < c_1 n$, where c_1 is a sufficiently small positive constant, the lower bound in Theorem 13 simplifies to $\Omega\left(\frac{n^3\Gamma}{B\sqrt{M}}\right)$, while the upper bound in Theorem 14 becomes $\mathcal{O}(\frac{n^3\Gamma}{B\sqrt{M}} + \frac{n^2|\mathcal{R}_B|}{B} + \frac{n|\mathcal{R}_T|}{B})$. Hence, if $(n|\mathcal{R}_B| + |\mathcal{R}_T|)/\Gamma \in \mathcal{O}(n^2/\sqrt{M})$, the lower-bound asymptotically matches the upper-bound. When disallowing recomputation, a similar analysis shows our upper and lower bounds match under the same condition for $M < c_2 n$, where c_2 is a sufficiently small positive constant.

6 Conclusion

This work contributed to the characterization of the I/O complexity of Dynamic Programming algorithms by establishing asymptotically tight lower bounds for a general class of DP algorithms sharing a common structure of sub-problem dependence. Our technique exploits common properties of the CDAGs corresponding to said algorithms, which makes it promising for the analysis of other families of recursive algorithms, in particular other families of DP algorithms of interest. The generality of our technique is further showcased by the ability to extend it to more complex algorithms, such as the Cocke-Younger-Kasami, for which we provide an (almost) asymptotically tight I/O lower bound and a matching algorithm.

Our analysis yields lower bounds both for computations in which no intermediate value is computed more than once and for general computations allowing recomputation. By doing so we reveal how when the size of the available cache is greater than $2n$ and $o\left(n^2\right)$, schedules using recomputation can achieve an asymptotic reduction of the number of required I/O operations by a factor $\Theta\left(n^2/\sqrt{M}\right)$. This is particularly significant as in many cases of interest (*e.g.,* Matrix Multiplication, Fast Fourier Transform) recomputation has been shown to enable a reduction of the I/O cost by at most a constant multiplicative factor.

Although it is known that recomputation can decrease the space and I/O complexity of certain CDAGs, we are still far from characterizing those CDAGs for which recomputation proves effective. This overarching objective remains a challenge for any efforts aimed at developing a general theory of the communication requirements of computations.

References

1. Alok Aggarwal and S Vitter, Jeffrey. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988. doi:10.1145/48529.48535.
2. Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Communication-optimal parallel and sequential Cholesky decomposition. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 245–252, 2009.
3. Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Graph expansion and communication costs of fast matrix multiplication. *Journal of the ACM (JACM)*, 59(6):1–23, 2013. doi:10.1145/2395116.2395121.
4. Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.
5. Gianfranco Bilardi and Lorenzo De Stefani. The I/O complexity of Strassen's matrix multiplication with recomputation. In *Workshop on Algorithms and Data Structures*, pages 181–192. Springer, 2017. doi:10.1007/978-3-319-62127-2_16.
6. Gianfranco Bilardi and Lorenzo De Stefani. The I/O complexity of Toom-Cook integer multiplication. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2034–2052. SIAM, 2019. doi:10.1137/1.9781611975482.123.
7. Gianfranco Bilardi and Lorenzo De Stefani. The DAG Visit Approach for Pebbling and I/O Lower Bounds. In *In Proc. FSTTCS 2022*, volume 250, pages 7:1–7:23, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2022.7.
8. Gianfranco Bilardi and Enoch Peserico. A characterization of temporal locality and its portability across memory hierarchies. In *International Colloquium on Automata, Languages, and Programming*, pages 128–139. Springer, 2001. doi:10.1007/3-540-48224-5_11.
9. Gianfranco Bilardi, Andrea Pietracaprina, and Paolo D'Alberto. On the space and access complexity of computation DAGs. In *Graph-Theoretic Concepts in Computer Science: 26th International Workshop, WG 2000 Konstanz, Germany, June 15–17, 2000 Proceedings 26*, pages 47–58. Springer, 2000. doi:10.1007/3-540-40064-8_6.

10 Gianfranco Bilardi and Franco P Preparata. Horizons of parallel computation. *Journal of Parallel and Distributed Computing*, 27(2):172–182, 1995. doi:10.1006/JPDC.1995.1080.

11 Gianfranco Bilardi and Franco P Preparata. Processor—time tradeoffs under bounded-speed message propagation: Part II, lower bounds. *Theory of Computing Systems*, 32(5):531–559, 1999. doi:10.1007/S002240000131.

12 Guy E Blleloch and Yan Gu. Improved parallel cache-oblivious algorithms for dynamic programming and linear algebra. *arXiv preprint*, 2018. arXiv:1809.09330.

13 Cary Cherng and Richard E Ladner. Cache efficient simple dynamic programming. *Discrete Mathematics & Theoretical Computer Science*, 2005.

14 Rezaul Chowdhury, Pramod Ganapathi, Yuan Tang, and Jesmin Jahan Tithi. Provably efficient scheduling of cache-oblivious wavefront algorithms. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 339–350, 2017. doi:10.1145/3087556.3087586.

15 Rezaul Chowdhury, Pramod Ganapathi, Stephen Tschudi, Jesmin Jahan Tithi, Charles Bachmeier, Charles E Leiserson, Armando Solar-Lezama, Bradley C Kuszmaul, and Yuan Tang. Autogen: Automatic discovery of efficient recursive divide-8-conquer algorithms for solving dynamic programming problems. *ACM Transactions on Parallel Computing (TOPC)*, 4(1):1–30, 2017. doi:10.1145/3125632.

16 Rezaul Alam Chowdhury and Vijaya Ramachandran. Cache-oblivious dynamic programming. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 591–600. Citeseer, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109622.

17 Rezaul Alam Chowdhury and Vijaya Ramachandran. The cache-oblivious gaussian elimination paradigm: theoretical framework, parallelization and experimental evaluation. In *Proceedings of the nineteenth annual ACM Symposium on Parallel Algorithms and Architectures*, pages 71–80, 2007. doi:10.1145/1248377.1248392.

18 Rezaul Alam Chowdhury and Vijaya Ramachandran. Cache-efficient dynamic programming algorithms for multicores. In *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, pages 207–216, 2008. doi:10.1145/1378533.1378574.

19 Rezaul Alan Chowdhury, Hai-Son Le, and Vijaya Ramachandran. Cache-oblivious dynamic programming for bioinformatics. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(3):495–510, 2008.

20 John Cocke. *Programming languages and their compilers: Preliminary notes*. New York University, 1969.

21 Lorenzo De Stefani and Vedant Gupta. On the I/O complexity of the CYK algorithm and of a family of related DP algorithms. *arXiv preprint arXiv:2410.20337*, 2024. doi:10.48550/arXiv.2410.20337.

22 Erik D Demaine, Andrea Lincoln, Quanquan C Liu, Jayson Lynch, and Virginia Vassilevska Williams. Fine-grained I/O complexity via reductions: New lower bounds, faster algorithms, and a time hierarchy. *arXiv preprint arXiv:1711.07960*, 2017. arXiv:1711.07960.

23 Xiangyun Ding, Yan Gu, and Yihan Sun. Parallel and (nearly) work-efficient dynamic programming. In *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 219–232, 2024. doi:10.1145/3626183.3659958.

24 Aaron Dunlop, Nathan Bodenstab, and Brian Roark. Reducing the grammar constant: an analysis of CYK parsing efficiency. Technical report, Technical report CSLU-2010-02, OHSU, 2010.

25 Reza Farivar, Harshit Kharbanda, Shivaram Venkataraman, and Roy H Campbell. An algorithm for fast edit distance computation on GPUs. In *2012 Innovative Parallel Computing (InPar)*, pages 1–9. IEEE, 2012.

26 Matteo Frigo, Charles E Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 285–297. IEEE, 1999. doi:10.1109/SFFCS.1999.814600.

27 Zvi Galil and Kunsoo Park. Dynamic programming with convexity, concavity and sparsity. *Theoretical Computer Science*, 92(1):49–76, 1992. doi:10.1016/0304-3975(92)90135-3.

28 Te Chiang Hu and Man Tak Shing. Computation of matrix chain products. Part I. *SIAM Journal on Computing*, 11(2):362–373, 1982. doi:10.1137/0211028.

29 Shachar Itzhaky, Rohit Singh, Armando Solar-Lezama, Kuat Yessenov, Yongquan Lu, Charles Leiserson, and Rezaul Chowdhury. Deriving divide-and-conquer dynamic programming algorithms using solver-aided transformations. *ACM SIGPLAN Notices*, 51(10):145–164, 2016.

30 Mohammad Mahdi Javanmard, Pramod Ganapathi, Rathish Das, Zafar Ahmad, Stephen Tschudi, and Rezaul Chowdhury. Toward efficient architecture-independent algorithms for dynamic programs. In *High Performance Computing: 34th International Conference, ISC High Performance 2019, Frankfurt/Main, Germany, June 16–20, 2019, Proceedings 34*, pages 143–164. Springer, 2019. doi:10.1007/978-3-030-20656-7_8.

31 Hong Jia-Wei. I/O complexity: The red-blue pebble game. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 326–333, 1981.

32 Tadao Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. *Coordinated Science Laboratory Report no. R-257*, 1966.

33 Donald E. Knuth. Optimum binary search trees. *Acta informatica*, 1:14–25, 1971. doi:10.1007/BF00264289.

34 Richard Koch, Tom Leighton, Bruce Maggs, and Satish Rao. Work-preserving emulations of fixed-connection networks. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 227–240, 1989.

35 Martin Lange and Hans Leiß. To CNF or not to CNF? an efficient yet presentable version of the cyk algorithm. *Informatica Didactica*, 8(2009):1–21, 2009. URL: http://www.informatica-didactica.de/cmsmadesimple/index.php?page=LangeLeiss2009.

36 J-S Park, Michael Penner, and Viktor K Prasanna. Optimizing graph algorithms for improved cache performance. *IEEE Transactions on parallel and distributed systems*, 15(9):769–782, 2004. doi:10.1109/TPDS.2004.44.

37 Cynthia A Patterson, Marc Snir, and Susan L Graham. *Getting up to speed: The future of supercomputing*. National Academies Press, 2005.

38 Vijaya Ramachandran. Cache-oblivious computation: Algorithms and experimental evaluation. In *2007 International Conference on Computing: Theory and Applications (ICCTA'07)*, pages 20–26. IEEE, 2007. doi:10.1109/ICCTA.2007.34.

39 John E Savage. Extending the Hong-Kung model to memory hierarchies. In *International Computing and Combinatorics Conference*, pages 270–281. Springer, 1995. doi:10.1007/BFB0030842.

40 Jacob Scott, Olga Holtz, and Oded Schwartz. Matrix multiplication I/O-complexity by path routing. In *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*, pages 35–45, 2015. doi:10.1145/2755573.2755594.

41 Michael Sipser. Introduction to the Theory of Computation. *ACM Sigact News*, 27(1):27–29, 1996. doi:10.1145/230514.571645.

42 Xinying Song, Shilin Ding, and Chin-Yew Lin. Better binarization for the CYK parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 167–176, 2008. URL: https://aclanthology.org/D08-1018/.

43 Shanjiang Tang, Ce Yu, Jizhou Sun, Bu-Sung Lee, Tao Zhang, Zhen Xu, and Huabei Wu. Easypdp: An efficient parallel dynamic programming runtime system for computational biology. *IEEE Transactions on Parallel and Distributed Systems*, 23(5):862–872, 2011. doi:10.1109/TPDS.2011.218.

44 Yuan Tang, Ronghui You, Haibin Kan, Jesmin Jahan Tithi, Pramod Ganapathi, and Rezaul A Chowdhury. Cache-oblivious wavefront: Improving parallelism of recursive dynamic programming algorithms without losing cache-efficiency. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 205–214, 2015.

45 Leslie Valiant. General context-free recognition in less than cubic time. *J. Comput. Syst. Sci.*, 1974.

46 F Frances Yao. Efficient dynamic programming using quadrangle inequalities. In *Proceedings of the twelfth annual ACM symposium on Theory of computing*, pages 429–435, 1980. doi:10.1145/800141.804691.

47 Daniel H Younger. Recognition and parsing of context-free languages in time n^3. *Information and control*, 10(2):189–208, 1967. doi:10.1016/S0019-9958(67)80007-X.

Deterministic $(2/3 - \varepsilon)$-Approximation of Matroid Intersection Using Nearly-Linear Independence-Oracle Queries

Tatsuya Terao
Research Institute for Mathematical Sciences, Kyoto University, Japan

Abstract

In the matroid intersection problem, we are given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ defined on the same ground set V of n elements, and the objective is to find a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of largest possible cardinality, denoted by r. In this paper, we consider a deterministic matroid intersection algorithm with only a nearly linear number of independence oracle queries. Our contribution is to present a deterministic $O(\frac{n}{\varepsilon} + r \log r)$-independence-query $(2/3 - \varepsilon)$-approximation algorithm for any $\varepsilon > 0$. Our idea is very simple: we apply a recent $\tilde{O}(n\sqrt{r}/\varepsilon)$-independence-query $(1 - \varepsilon)$-approximation algorithm of Blikstad [ICALP 2021], but terminate it before completion. Moreover, we also present a semi-streaming algorithm for $(2/3 - \varepsilon)$-approximation of matroid intersection in $O(1/\varepsilon)$ passes.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Mathematics of computing → Matroids and greedoids; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Matroid intersection, approximation algorithm, streaming algorithm

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.50

Related Version *Full Version*: `https://arxiv.org/abs/2410.18820`

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 The author thanks Yusuke Kobayashi for his generous support and helpful comments on the manuscript.

1 Introduction

The *matroid intersection problem* is one of the most fundamental problems in combinatorial optimization. In this problem, we are given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ defined on the same ground set V of n elements, and the objective is to find a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of largest possible cardinality, denoted by r. This problem generalizes many important combinatorial optimization problems such as bipartite matching, packing spanning trees, and arborescences in directed graphs. Furthermore, it has also several applications outside of traditional combinatorial optimization such as electrical engineering [51, 55].

To design an algorithm for arbitrary matroids, it is common to consider an oracle model: an algorithm accesses a matroid through an oracle. The most standard and well-studied oracle is an *independence oracle*, which takes as input a subset $S \subseteq V$ and outputs whether S is independent or not. Many studies consider the following research question: how few queries can we solve the matroid intersection problem with?

© Tatsuya Terao;
licensed under Creative Commons License CC-BY 4.0

Starting the work of Edmonds [22,23], many algorithms with polynomial query complexity for the matroid intersection problem have been studied [4, 9, 11, 13–15, 18, 20, 21, 36, 48, 49, 52, 53, 57, 60]. Edmonds [22] developed the first polynomial-query algorithm by reduction to the matroid partitioning problem. This algorithm requires $O(n^4)$ independence oracle queries. More direct algorithms were given by Aigner–Dowling [4] and Lawler [48]. The algorithm of Lawler requires $O(nr^2)$ independence oracle queries. In 1986, Cunningham [21] presented an $O(nr^{3/2})$-independence-query algorithm by using a blocking flow approach, which is akin to the bipartite matching algorithm of Hopcroft–Karp [35]. Chekuri–Quanrud [20] pointed out that, for any $\varepsilon > 0$, an $O(nr/\varepsilon)$-independence-query $(1-\varepsilon)$-approximation algorithm can be obtained by terminating Cunningham's algorithm early. This idea comes from the well-known fact that, for the bipartite matching problem, a linear-time $(1-\varepsilon)$-approximation algorithm can be obtained by terminating Hopcroft–Karp's algorithm early. Recently, Nguyễn [52] and Chakrabarty–Lee–Sidford–Singla–Wong [18] independently presented a new binary search technique that can efficiently find edges in the exchange graph and developed a combinatorial $\tilde{O}(nr)$-independence-query exact algorithm.[1] Chakrabarty et al. also presented a new augmenting sets technique and developed an $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$-independence-query $(1-\varepsilon)$-approximation algorithm. The techniques developed by Nguyễn and Chakrabarty et al. have been used in recent several studies on fast algorithms for other matroid problems [12, 16, 43, 54, 58, 61]. Blikstad–van den Brand-Mukhopadhyay–Nanongkai [15] first broke the $\tilde{O}(n^2)$-independence-query bound for exact matroid intersection algorithms. Blikstad [11] improved the independence query complexity of a $(1-\varepsilon)$-approximation algorithm to $\tilde{O}(n\sqrt{r}/\varepsilon)$. Blikstad's improvement on the $(1-\varepsilon)$-approximation algorithm resulted in a randomized $\tilde{O}(nr^{3/4})$-independence-query exact algorithm and a deterministic $\tilde{O}(nr^{5/6})$-independence-query exact algorithm. Recently, Quanrud [53] presented a randomized $\tilde{O}_\varepsilon(n + r^{3/2})$-independence-query $(1-\varepsilon)$-approximation algorithm[2]. Remarkably, Blikstad-Tu [14] very recently presented a randomized $\tilde{O}_\varepsilon(n)$-independence-query $(1-\varepsilon)$-approximation algorithm.[3]

Both the recent $(1-\varepsilon)$-approximation algorithms by Quanrud [53] and Blikstad–Tu [14] are randomized. Therefore, it remains an open question whether a deterministic $\tilde{O}_\varepsilon(n)$ independence-query $(1-\varepsilon)$-approximation algorithm can be achieved for almost the entire range of r.[4] Then, we consider a deterministic matroid intersection algorithm with only a nearly linear number of independence oracle queries. It is well-known that any maximal common independent set is at least half the size of a maximum common independent set (see e.g., [47, Proposition 13.26]). Thus, the natural greedy algorithm – pick an element whenever possible – can be regarded as a deterministic 1/2-approximation algorithm using a linear number of independence oracle queries. In fact, even beating the trivial 1/2-approximation ratio for deterministic nearly-linear-independence-query algorithms remains an open problem for almost the entire range of r.[4]

In this work, we present a deterministic $(2/3-\varepsilon)$-approximation algorithm using a nearly linear number of independence oracle queries.

▶ **Theorem 1.** *Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, for any $\varepsilon > 0$, there is a deterministic algorithm that finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (2/3-\varepsilon)r$, using $O(\frac{n}{\varepsilon} + r \log r)$ independence oracle queries.*

[1] The $\tilde{O}$ notation omits factors polynomial in $\log n$.
[2] The $\tilde{O}_\varepsilon$ notation omits factors polynomial in ε and $\log n$.
[3] The results in this manuscript were obtained independently of the remarkable result by Blikstad–Tu [14].
[4] Blikstad–Tu [14] presented a deterministic $(1-\varepsilon)$-approximation algorithm using a strictly linear number of independence oracle queries only when $r = \Theta(n)$ (and ε is constant).

Our algorithm uses only a strictly linear number of independence oracle queries when $r = O(n/\log n)$ (and ε is constant). Prior to our work, Guruganesh–Singla [34] presented a randomized $O(n)$-independence-query algorithm that achieves $(1/2+\delta)$-approximation for some small constant $\delta > 0$, which was the best approximation ratio for (possibly randomized) algorithms using only a strictly linear number of independence oracle queries.[4]

We note that the time complexity of our algorithm is dominated by the independence oracle queries. That is, our algorithm in Theorem 1 has time complexity $O\left((\frac{n}{\varepsilon} + r\log r)\cdot \mathcal{T}_{\mathrm{ind}}\right)$, where $\mathcal{T}_{\mathrm{ind}}$ denotes the maximum running time of a single independence oracle query.

Our idea is very simple: we apply a recent $\tilde{O}(n\sqrt{r}/\varepsilon)$-independence-query $(1-\varepsilon)$-approximation algorithm of Blikstad [11], but terminate it before completion. We observe that a $(2/3-\varepsilon)$-approximate solution can be obtained by terminating Blikstad's algorithm early. As such, our algorithm does not introduce technical novelty, but we believe that it enhances the understanding of algorithms for the matroid intersection problem.

We also implement our algorithm using a *rank oracle*, which takes as input a subset $S \subseteq V$ and outputs the size of the maximum cardinality independent subset of S. Several recent studies have considered fast matroid intersection algorithms in the rank oracle model [18,49,60]. We note that the rank oracle is more powerful than the independence oracle, since a single query to the rank oracle can determine whether a given set is independent or not.

Chakrabarty–Lee–Sidford–Singla–Wong [18] presented an $\tilde{O}(n\sqrt{r})$-rank-query exact algorithm and an $O(\frac{n}{\varepsilon}\log n)$-rank-query $(1-\varepsilon)$-approximation algorithm. Their $(1-\varepsilon)$-approximation algorithm requires nearly linear rank oracle queries. Our second result is to obtain a $(2/3-\varepsilon)$-approximation algorithm using a strictly linear number of rank oracle queries.

▶ **Theorem 2.** *Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, for any $\varepsilon > 0$, there is a deterministic algorithm that finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (2/3-\varepsilon)r$, using $O(\frac{n}{\varepsilon})$ rank oracle queries.*

We note that the time complexity of our algorithm is dominated by the rank oracle queries. That is, our algorithm in Theorem 2 has time complexity $O\left(\frac{n}{\varepsilon}\cdot \mathcal{T}_{\mathrm{rank}}\right)$, where $\mathcal{T}_{\mathrm{rank}}$ denotes the maximum running time of a single rank oracle query.

Surprisingly, our algorithm can also be implemented in the semi-streaming model of computation. The streaming model for graph problems was initiated by Feigenbaum–Kannan–McGregor–Suri–Zhang [27], which is called the semi-streaming model. In this model, an algorithm computes over a graph stream using a limited space of $\tilde{O}(n)$ bits, where n is the number of vertices. The maximum matching problem is one of the most studied problems in the graph streaming setting [1–3, 5–8, 10, 24–27, 29–31, 33, 40–42, 44–46, 50, 59]. The first of these studies was a $(2/3-\varepsilon)$-approximation algorithm for the bipartite maximum matching problem by Feigenbaum et al. [27], which uses $\tilde{O}(n)$ bits of space and $O(\log(1/\varepsilon)/\varepsilon)$ passes. In particular, whether a $(1-\varepsilon)$ approximation can be achieved in $\mathrm{poly}(1/\varepsilon)$ passes in the semi-streaming model for general graphs was a significant open problem until it was solved by Fischer–Mitrović–Uitto [30].

Since the matroid intersection problem is a well-known generalization of the bipartite matching problem, there are also several studies on the matroid intersection problem (and its generalizations) in the streaming setting in the literature [17, 19, 28, 32, 37–39]. For the matroid intersection problem, in the semi-streaming model, we consider algorithms such that the memory usage is $O((r_1+r_2)\mathrm{polylog}(r_1+r_2))$, where r_1 and r_2 denote the ranks of matroids $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. We note that this memory requirement is natural when we formulate the bipartite matching problem as the matroid intersection problem.

In this work, we focus on constant-passes (poly$(1/\varepsilon)$-passes) semi-streaming algorithms for the matroid intersection problem, particularly given the significance of constant-passes semi-streaming algorithms for the matching problem.[5] Recently, Huang–Sellier [37] presented a $(2/3-\varepsilon)$-approximation semi-streaming algorithm for the matroid intersection problem in the random-order streaming model.[6]

Our third result is to obtain a $(2/3-\varepsilon)$-approximation semi-streaming algorithm with $O(\frac{1}{\varepsilon})$ passes in the adversary-order streaming model.

▶ **Theorem 3.** *Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, for any $\varepsilon > 0$, there is a deterministic semi-streaming algorithm that finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (2/3-\varepsilon)r$, using $O(\frac{r_1+r_2}{\varepsilon})$ memory and $O(\frac{1}{\varepsilon})$ passes.*

Our algorithm in Theorem 3 is a generalization of the bipartite matching semi-streaming algorithm of Feigenbaum et al. [27].

2 Preliminaries

2.1 Notation

Here, we provide the basic notations and conventions used throughout the paper.

Set Notation

For a set A and an element x, we will often write $A + x := A \cup \{x\}$ and $A - x := A \setminus \{x\}$. For two sets A and B, we will also write $A + B := A \cup B$ and $A - B := A \setminus B$ when co confusion can arise.

Matroid

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 $S \in \mathcal{I}$ and $S' \subseteq S$, then $S' \in \mathcal{I}$.

(Augmentation property) If $S, S' \in \mathcal{I}$ and $|S'| < |S|$, then there exists $v \in S \setminus S'$ such that $S' + v \in \mathcal{I}$.

A set $S \subseteq V$ is called *independent* if $S \in \mathcal{I}$ and *dependent* otherwise.

Rank

For a matroid $\mathcal{M} = (V, \mathcal{I})$, we define the *rank* of $\mathcal{M}$ as $\mathrm{rank}(\mathcal{M}) = \max\{|S| \mid S \in \mathcal{I}\}$. In addition, for any $S \subseteq V$, we define the *rank* of S as $\mathrm{rank}_{\mathcal{M}}(S) = \max\{|T| \mid T \subseteq S, T \in \mathcal{I}\}$.

---

[5] Based on the results from Assadi [5] and Quanrud [53], we believe that it is able to develop a semi-streaming $(1-\varepsilon)$-approximation algorithm with similar space usage. However, such an algorithm would require $O(\log(n) \cdot \mathrm{poly}(1/\varepsilon))$ passes. In the context of research on streaming matching algorithms, the distinctions between $O(\mathrm{poly}(1/\varepsilon))$ passes and $O(\log(n) \cdot \mathrm{poly}(1/\varepsilon))$ are well recognized and crucial; see [5, Table 1 in the arXiv version]. We also believe that the result by Blikstad–Tu [14] is unlikely to immediately lead to constant-passes semi-streaming algorithm for the matroid intersection problem. This is because, while Blikstad–Tu used some idea from the constant-passes semi-streaming algorithm of Assadi-Liu-Tarjan [8] for the bipartite matching problem, Blikstad–Tu relies on the technique from Quanrud [53]. Quanrud's algorithm used some idea from the $O(\log(n) \cdot \mathrm{poly}(1/\varepsilon))$-passes semi-streaming algorithm by Assadi [5].

[6] We note that their algorithm requires too many queries to compute the density-based decomposition. Thus, their algorithm does not imply a nearly-linear-independence-query $(2/3-\varepsilon)$-approximation algorithm.

Matroid Intersection

Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ defined on the same ground set V, a *common independent set* S is a set in $\mathcal{I}_1 \cap \mathcal{I}_2$. In the *matroid intersection problem*, the aim is to find a largest common independent set, whose cardinality we denote by r.

2.2 Exchange Graph and Augmenting Path

We provide the standard definitions of *exchange graph* and *augmenting paths* and known lemmas used in recent fast matroid intersection algorithms. Many matroid intersection algorithms use an approach of iteratively finding augmenting paths in the exchange graph.

▶ **Definition 4** (Exchange Graph). *Consider a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of two matroids $\mathcal{M}_1$ and $\mathcal{M}_2$. The* exchange graph *is defined as a directed graph $G(S) = (V \cup \{s, t\}, E)$, with $s, t \notin V$ and $E = E' \cup E'' \cup E_s \cup E_t$, where*

$$\begin{aligned} E' &= \{(u, v) \mid u \in S, v \in V \setminus S, S - u + v \in \mathcal{I}_1\}, \\ E'' &= \{(v, u) \mid u \in S, v \in V \setminus S, S - u + v \in \mathcal{I}_2\}, \\ E_s &= \{(s, v) \mid v \in V \setminus S, S + v \in \mathcal{I}_1\}, \text{ and} \\ E_t &= \{(v, t) \mid v \in V \setminus S, S + v \in \mathcal{I}_2\}. \end{aligned}$$

▶ **Lemma 5** (Shortest Augmenting Path; see [56, Theorem 41.2]). *Let $s, v_1, v_2, \ldots, v_{\ell-1}, t$ be a shortest (s, t)-path in the exchange graph $G(S)$ relative to a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$. Then, $S' = S + v_1 - v_2 + \cdots - v_{\ell-2} + v_{\ell-1} \in \mathcal{I}_1 \cap \mathcal{I}_2$.*

Cunningham's [21] matroid intersection algorithm and recent fast matroid intersection algorithms [11, 15, 18, 20, 52] rely on the following lemma.

▶ **Lemma 6** (from [21, Corollary 2.2]). *Let S be a common independent set which is not maximum. Then, there exists an (s, t)-path of length at most $\frac{2|S|}{r-|S|} + 2$ in the exchange graph $G(S)$.*

Chakrabarty–Lee–Sidford–Singla–Wong [18] presented a new binary search technique that can efficiently find edges in the exchange graph. (This technique was developed independently by Nguyễn [52].)

▶ **Lemma 7** (Binary Search Technique from [18, 52]). *Given a matroid $\mathcal{M} = (V, \mathcal{I})$, an independent set $S \in \mathcal{I}$, an element $v \in V \setminus S$, and $T \subseteq S$, using $O(\log |T|)$ independence oracle queries, we can find an element $u \in T$ such that $S + v - u \in \mathcal{I}$ or otherwise determine that no such element exists. Furthermore, if there exists no such an element, then this procedure uses only one independence oracle query.*[7]

Let `FindExchange`$(\mathcal{M}, S, v, T)$ be the procedure that implements Lemma 7.

Let D_i be the set of elements $v \in V$ such that the distance from s to v is exactly i in the exchange graph $G(S)$. Chakrabarty et al. [18] showed that the distance layers D_i can be computed efficiently by using the binary search technique. In their algorithm, they compute odd and even layers separately because `FindExchange` cannot find both incoming and outgoing edges due to restrictions on the allowed queries; see [18, Algorithm 6 and Lemma 19]. Their algorithm requires $O(nr + r^2 \log r)$ independence oracle queries to compute all distance layers $D_1, D_2, \ldots, D_{2r+1}$; see [18, Remark just after Proof of Theorem 18]. In

[7] This is because there exists such an element if and only if $S + v - T \in \mathcal{I}$.

Algorithm 1 `GetDistance`.

Input: a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$
Output: distance layers $D_1 \subseteq V \setminus S, D_2 \subseteq S$, and $D_3 \subseteq V \setminus S$
1 $D_1 \leftarrow \emptyset$, $D_2 \leftarrow \emptyset$, $D_3 \leftarrow \emptyset$
2 **for** $v \in V \setminus S$ *with* $S + v \in \mathcal{I}_1$ **do**
3 $\quad D_1 \leftarrow D_1 + v$
4 $Q \leftarrow D_1$, $T \leftarrow S$
5 **while** Q *contains some* v **do**
6 $\quad$ **while** $u =$ `FindExchange`$(\mathcal{M}_2, S, v, T)$ *satisfies* $u \neq \emptyset$ **do**
7 $\quad\quad D_2 \leftarrow D_2 + u$, $T \leftarrow T - u$
8 $\quad Q \leftarrow Q - v$
9 **for** $v \in V \setminus (S \cup D_1)$ *with* $S + v - D_2 \in \mathcal{I}_1$ **do**
10 $\quad D_3 \leftarrow D_3 + v$

our matroid intersection algorithm, we only need to compute D_1, D_2, and D_3. Their algorithm can compute D_1, D_2, and D_3 using $O(n + r \log r)$ independence oracle queries. See `GetDistance` (Algorithm 1) for the pseudocode of the algorithm.

▶ **Lemma 8** (follows from [18, Lemma 19]). *Given a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of two matroids $\mathcal{M}_1$ and $\mathcal{M}_2$, using $O(n + r \log r)$ independence oracle queries, we can find the distance layers $D_1 \subseteq V \setminus S, D_2 \subseteq S$, and $D_3 \subseteq V \setminus S$.*

For completeness, we give a proof of Lemma 8.

Proof. The procedure `GetDistance` (Algorithm 1) simply performs a breadth first search in the exchange graph $G(S)$. Thus, the procedure `GetDistance` correctly computes the distance layers D_1, D_2, and D_3. Here, each element $u \in D_2$ is found by `FindExchange` only once. Thus, the number of `FindExchange` calls that do not output $\emptyset$ is $|D_2| \leq r$, and the number of `FindExchange` calls that output $\emptyset$ is $|D_1| \leq n$. Hence, by Lemma 7, the number of independence oracle queries used in Line 6 is $O(n + r \log r)$. In addition, the number of independence oracle queries used in Lines 2 and 9 is $O(n)$, which completes the proof. ◀

3 Augmenting Sets Technique

In this section, we recall the definition and properties of *augmenting sets*, which were introduced as a generalization of *augmenting paths* by Chakrabarty–Lee–Sidford–Singla–Wong [18]. The augmenting sets technique plays a crucial role in the $(1 - \varepsilon)$-approximation algorithms of Chakrabarty et al. [18] and Blikstad [11].

In our matroid intersection algorithm, we only need to find an augmenting set in the exchange graph whose shortest (s, t)-path length is 4. Thus, we introduce the definition and properties of augmenting sets when restricted to the case where the length of a shortest augmenting path is 4.

▶ **Definition 9** (Augmenting Sets from [18, Definition 24]). *Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ be such that shortest augmenting paths in $G(S)$ have length 4. We say that a collection of sets $\Pi := (B_1, A_1, B_2)$ form an* augmenting set *(of* width w*) in $G(S)$ if the following conditions are satisfied:*
(a) $B_1 \subseteq D_1$, $A_1 \subseteq D_2$, *and* $B_2 \subseteq D_3$.
(b) $|B_1| = |A_1| = |B_2| = w$
(c) $S + B_1 \in \mathcal{I}_1$

(d) $S + B_1 - A_1 \in \mathcal{I}_2$
(e) $S - A_1 + B_2 \in \mathcal{I}_1$
(f) $S + B_2 \in \mathcal{I}_2$

▶ **Theorem 10** (from [18, Theorem 25]). *Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ be such that shortest augmenting paths in $G(S)$ have length 4. Let $\Pi := (B_1, A_1, B_2)$ be an augmenting set in the exchange graph $G(S)$. Then, the set $S' := S \oplus \Pi := S + B_1 - A_1 + B_2$ is a common independent set.*

Here, we recall the definition and property of *maximal augmenting sets*, which correspond to a maximal collection of shortest augmenting paths such that augmentation along them must increase the (s,t)-distance. In the $(1-\varepsilon)$-approximation algorithms of Chakrabarty et al. and Blikstad, they repeatedly find a maximal augmenting set and augment along it.

▶ **Definition 11** (Maximal Augmenting Sets from [18, Definitions 31 and 35]). *Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ be such that shortest augmenting paths in $G(S)$ have length 4. Let $\Pi := (B_1, A_1, B_2)$ and $\tilde{\Pi} := (\tilde{B}_1, \tilde{A}_1, \tilde{B}_2)$ be two augmenting sets. We say* $\tilde{\Pi}$ contains Π *if $B_1 \subseteq \tilde{B}_1$, $A_1 \subseteq \tilde{A}_1$ and $B_2 \subseteq \tilde{B}_2$. We use the notation $\Pi \subseteq \tilde{\Pi}$ to denote this. An augmenting set Π is called* maximal *if there exists no other augmenting set $\tilde{\Pi}$ containing Π.*

▶ **Theorem 12** (from [18, Theorem 36]). *Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ be such that shortest augmenting paths in $G(S)$ have length 4. Let Π be a maximal augmenting set in the exchange graph $G(S)$. Then, there is no augmenting path of length at most 4 in $G(S \oplus \Pi)$.*

Here, we recall the definition of *partial augmenting sets*, which are the relaxed form of augmenting sets. In the algorithms of Chakrabarty et al. and Blikstad for finding a maximal augmenting set, they keep track of the partial augmenting set, which is iteratively made close to a maximal augmenting set through some refine procedures.

▶ **Definition 13** (Partial Augmenting Sets from [18, Definition 37]). *Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ be such that shortest augmenting paths in $G(S)$ have length 4. We say that $\Phi := (B_1, A_1, B_2)$ forms a* partial augmenting set *in $G(S)$ if it satisfies the conditions (a), (c), (e), (f) of an* augmenting set*, and the following two relaxed conditions:*
(b) $|B_1| \geq |A_1| \geq |B_2|$
(d) $\mathrm{rank}_2(S + B_1 - A_1) = \mathrm{rank}_2(S)$

Chakrabarty et al. presented an efficient algorithm to convert any partial augmenting set Φ into an augmenting set Π.

▶ **Lemma 14** (from [18, Lemma 38]). *Given a partial augmenting set $\Phi = (B_1, A_1, B_2)$, using $O(n)$ independence oracle queries, we can find an augmenting set $\Pi = (B_1', A_1', B_2')$ such that $B_1' \subseteq B_1$, $A_1' \subseteq A_1$ and $B_2' = B_2$.*

Chakrabarty et al. [18] presented an algorithm to find a maximal augmenting set in the exchange graph $G(S)$, where shortest augmenting paths have length $2(\ell+1)$, using $O(n^{1.5}\sqrt{\ell \log r})$ independence oracle queries; see [18, Proof of Theorem 48 and Proof of Theorem 21]. Blikstad [11] improved the query complexity and presented an algorithm to find a maximal augmenting set using $O(n\sqrt{r \log r})$ independence oracle queries; see [11, Lemma 35 and Proof of Theorem 1].

In Blikstad's algorithm for finding a maximal augmenting set, we keep track of a partial augmenting set and iteratively update it to closer to a maximal augmenting set. To obtain a fast algorithm, Blikstad combines two algorithms `Refine` [11, Algorithm 4] and `RefinePath` [11, Algorithm 5]; see [11, Algorithm 6 and Lemma 35]. Let $p \in [1, r]$ be the parameter that controls the trade-off of the two algorithms. He first applies `Refine` to find a

partial augmenting set that is close enough to a maximal augmenting set, which uses $O(nr/p)$ independence oracle queries. Then, he applies `RefinePath` until the partial augmenting set becomes a maximal augmenting set, which uses $O(np \log r)$ independence oracle queries.

In our algorithm, we apply only the first part `Refine` of Blikstad's algorithm, as formally stated as follows.

▶ **Lemma 15** (follows from [11, Lemma 35]). *For any $p \in [1, r]$, given a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ and the distance layers D_1, D_2, and D_3, using $O(\frac{nr}{p})$ independence oracle queries, we can find a partial augmenting set $\Phi = (B_1, A_1, B_2)$ such that the following properties hold.*

- **(i)** *There is an augmenting set $\Pi = (B_1', A_1', B_2')$ such that $B_1' \subseteq B_1$, $A_1' \subseteq A_1$ and $B_2' = B_2$.*
- **(ii)** *We have $|B_1| - |B_2| \leq p$.*
- **(iii)** *There is a maximal augmenting set $\tilde{\Pi}$ of width at most $|B_1|$ in $G(S)$.*

To obtain Lemma 15, we apply `Refine` [11, Algorithm 4] (see also Algorithm 5) until $|B_1| - |B_2| \leq p$, but at least once. The procedure `Refine` makes the partial augmenting set $\Phi = (B_1, A_1, B_2)$ close to a maximal augmenting set. To become $|B_1| - |B_2| \leq p$, we need to apply `Refine` $O(|S|/p + 1)$ times; see [11, Proof of Lemma 35]. Since each call of `Refine` uses $O(n)$ independence oracle queries (see [11, Lemma 29]), the total number of independence oracle queries is $O(nr/p)$.

▶ Remark 16. Lemma 15 is not explicitly stated in Blikstad's [11] paper. By the property of a partial augmenting set and the argument in [11, Proof of Lemma 35], it is clear that the conditions (i) and (ii) are satisfied. In particular, here we show that the condition (iii) is also satisfied. In the algorithm in [11, Algorithm 6 and Lemma 35] for finding a maximal augmenting set, after applying `Refine` until $|B_1| - |B_2| \leq p$, we apply `RefinePath` [11, Algorithm 5] until the partial augmenting set becomes a maximal augmenting set. While `RefinePath` modifies B_1, it does not increase $|B_1|$; see [11, the second paragraph of the proof of Lemma 35]. Thus, the partial augmenting set after applying `RefinePath` is a maximal augmenting set of width at most $|B_1|$. Here, let this maximal augmenting set be $\tilde{\Pi}$.

4 $(2/3 - \varepsilon)$-Approximation Algorithm Using Nearly-Linear Independence-Oracle Queries

In this section, by providing a nearly-linear-independence-query $(2/3 - \varepsilon)$-approximation algorithm for the matroid intersection problem, we prove Theorem 1, which we restate here.

▶ **Theorem 1.** *Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, for any $\varepsilon > 0$, there is a deterministic algorithm that finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (2/3 - \varepsilon)r$, using $O(\frac{n}{\varepsilon} + r \log r)$ independence oracle queries.*

We use the following lemma to prove Theorem 1.

▶ **Lemma 17.** *Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ be a common independent set such that shortest augmenting paths in $G(S)$ have length at least 6. Then, S is a 2/3-approximate solution of the matroid intersection problem.*

Proof. We argue by contraposition. Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ be a common independent set with $|S| < \frac{2}{3}r$. Then, we have $\frac{2|S|}{r-|S|} + 2 < 6$. Thus, by Lemma 6, there exists an augmenting path of length at most $\frac{2|S|}{r-|S|} + 2 < 6$ in $G(S)$. This completes the proof. ◀

Now, we give a proof of Theorem 1. See Algorithm 2 for the pseudocode of our algorithm.

Algorithm 2 $(2/3-\varepsilon)$-approximation algorithm for the matroid intersection problem.

Compute a maximal common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$. `// Write` $\bar{r} = |S|$.
Compute the distance layers D_1, D_2, and D_3 in the exchange graph $G(S)$.
if *the* (s,t)*-distance in* $G(S)$ *is more than* 4 **then**
  return S
Apply `Refine` (Algorithm 5) until $|B_1| - |B_2| \leq \varepsilon\bar{r}$.
Apply Lemma 14 for the obtained partial augmenting set $\Phi = (B_1, A_1, B_2)$ and then obtain an augmenting set $\Pi = (B_1', A_1', B_2')$ such that $B_1' \subseteq B_1$, $A_1' \subseteq A_1$ and $B_2' = B_2$.
return $S \oplus \Pi$

Proof of Theorem 1. We give an algorithm to find a $(2/3-\varepsilon)$-approximate solution for the matroid intersection problem. In our algorithm, we first compute a maximal common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, which uses $O(n)$ independence oracle queries. Here, let $\bar{r}$ be the size of this maximal common independent set S. We note that $r \leq 2\bar{r}$ since any maximal common independent set is at least half the size of a maximum common independent set (see e.g., [47, Proposition 13.26]).

Then, we compute the distance layers D_1, D_2, and D_3 in the exchange graph $G(S)$, which, by Lemma 8, uses $O(n + r\log r)$ independence oracle queries. Here, there is no augmenting path of length 2 in $G(S)$, since S is a maximal common independent set. Thus, shortest augmenting paths in $G(S)$ have length at least 4.

Next, by checking whether $S+v \in \mathcal{I}_2$ for each $v \in D_3$, we check whether the (s,t)-distance is 4 or not, which uses $O(n)$ independence oracle queries. If the (s,t)-distance is more than 4, then S is already a 2/3-approximate solution by Lemma 17, so we output S. Otherwise, we apply Lemma 15 in which we set the parameter $p = \varepsilon\bar{r}$, which uses $O(n/\varepsilon)$ independence oracle queries.[8] Then, we obtain a partial augmenting set $\Phi = (B_1, A_1, B_2)$. By applying Lemma 14 for the obtained partial augmenting set Φ, using $O(n)$ independence oracle queries, we can find an augmenting set $\Pi = (B_1', A_1', B_2')$ such that $B_1' \subseteq B_1$, $A_1' \subseteq A_1$ and $B_2' = B_2$. Then, we output the set $S \oplus \Pi$.

By Theorem 10, the set $S \oplus \Pi$ is a common independent set. Now, we show that the obtained solution $S \oplus \Pi$ is a $(2/3-\varepsilon)$-approximate solution.

By the condition (iii) in Lemma 15, there is a maximal augmenting set $\tilde{\Pi} = (\tilde{B}_1, \tilde{A}_1, \tilde{B}_2)$ in $G(S)$ such that $|\tilde{B}_1| \leq |B_1|$. By Theorems 10 and 12 and Lemma 17, $S \oplus \tilde{\Pi}$ is a 2/3-approximate solution, so we have $|S \oplus \tilde{\Pi}| \geq \frac{2}{3}r$. Moreover, by the condition (ii) in Lemma 15, we have $|\tilde{B}_1| \leq |B_1| \leq |B_2| + p = |B_2'| + p = |B_1'| + p$. Thus, we have $|S \oplus \tilde{\Pi}| - |S \oplus \Pi| = |\tilde{B}_1| - |B_1'| \leq p = \varepsilon\bar{r} \leq \varepsilon r$. Therefore, we have $|S \oplus \Pi| \geq (2/3-\varepsilon)r$, which completes the proof. ◀

5 $(2/3-\varepsilon)$-Approximation Algorithm Using Linear Rank-Oracle Queries

In this section, by providing a linear-rank-query $(2/3-\varepsilon)$-approximation algorithm for the matroid intersection problem, we prove Theorem 2. We show that Algorithm 2 can be implemented as a $(2/3-\varepsilon)$-approximation algorithm with linear rank oracle queries. We restate Theorem 2 here.

[8] We apply `Refine` (Algorithm 5) only $O(1/\varepsilon)$ times.

▶ **Theorem 2.** *Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, for any $\varepsilon > 0$, there is a deterministic algorithm that finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (2/3 - \varepsilon)r$, using $O(\frac{n}{\varepsilon})$ rank oracle queries.*

Chakrabarty–Lee–Sidford–Singla–Wong [18] showed that a $(1-\varepsilon)$-approximate solution can be obtained with $O(\frac{n}{\varepsilon} \log n)$ rank oracle queries by using the binary search technique; see [18, Theorem 17]. In their $(1-\varepsilon)$-approximation algorithm, they find some edges using the binary search technique. For each edge, this requires $O(\log n)$ rank oracle queries. On the other hand, we show that a $(2/3-\varepsilon)$-approximate solution can be obtained with only $O(n/\varepsilon)$ rank oracle queries. In our $(2/3-\varepsilon)$-approximation algorithm, we find an almost maximal augmenting set without needing to identify any specific edges. Hence, the rank oracle query complexity of our algorithm is linear without any logarithmic factor.

We use the following lemma to prove Theorem 2.

▶ **Lemma 18.** *Given a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, using $O(n)$ rank oracle queries, we can find the distance layers $D_1 \subseteq V \setminus S, D_2 \subseteq S$, and $D_3 \subseteq V \setminus S$.*

Proof. Our algorithm for computing D_1, D_2, and D_3 is almost the same as Algorithm 1. Since a single rank oracle query can determine whether a given subset is independent or not, Algorithm 1 except for computing D_2 can be implemented with $O(n)$ rank oracle queries. Now, we show how D_2 can be computed with $O(n)$ rank oracle queries. For each $u \in S$, we check whether $\mathrm{rank}_{\mathcal{M}_2}(S + D_1 - u) \geq \mathrm{rank}_{\mathcal{M}_2}(S)$ holds or not. If $\mathrm{rank}_{\mathcal{M}_2}(S + D_1 - u) \geq \mathrm{rank}_{\mathcal{M}_2}(S)$ holds, then there exists an element $v \in D_1$ such that $S + v - u \in \mathcal{I}_2$, and consequently $u \in D_2$. Otherwise, there does not exist an element $v \in D_1$ such that $S + v - u \in \mathcal{I}_2$, and consequently $u \notin D_2$. This completes the proof. ◀

Proof of Theorem 2. Now, we show how Algorithm 2 can be implemented with $O(n/\varepsilon)$ rank oracle queries. As mentioned in the proof of Theorem 1, all parts of Algorithm 2, except for Line 2, can be implemented with $O(n/\varepsilon)$ independence oracle queries. Since a single rank oracle query can determine whether a given subset is independent or not, we can implement all parts of Algorithm 2, except for Line 2, with $O(n/\varepsilon)$ rank oracle queries. In addition, by Lemma 18, we can implement Line 2 with $O(n)$ rank oracle queries, which completes the proof. ◀

6 $(2/3 - \varepsilon)$-Approximation Algorithm in the Semi-Streaming Model

In this section, by providing a $(2/3-\varepsilon)$-approximation algorithm for the matroid intersection problem in the semi-streaming model, we prove Theorem 3. We show that Algorithm 2 can also be implemented in the semi-streaming model as a $(2/3-\varepsilon)$-approximation algorithm using a constant number of passes. We restate Theorem 3 here.

▶ **Theorem 3.** *Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, for any $\varepsilon > 0$, there is a deterministic semi-streaming algorithm that finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (2/3 - \varepsilon)r$, using $O(\frac{r_1 + r_2}{\varepsilon})$ memory and $O(\frac{1}{\varepsilon})$ passes.*

Before we present how Algorithm 2 can be implemented in the semi-streaming model, we describe the outline of Blikstad's [11] algorithm of Lemma 15. This is because some of his results will be used in our argument later. In this paper, we skip the proof of the correctness of the algorithm; see [11, Sections 3.1 and 3.3] for the proof. Recall that, in Lemma 15, we apply `Refine` only $O(1/\varepsilon)$ times. See Algorithm 5 for the pseudocode of `Refine`.

Algorithm 3 `RefineAB`(k) (from [11, Algorithm 1]); called `Refine1` in [18, Algorithm 9].

1 Find maximal $B \subseteq F_{2k+1}$ s.t. $S - A_k + B_{k+1} + B \in \mathcal{I}_1$
2 $B_{k+1} \leftarrow B_{k+1} + B$, $F_{2k+1} \leftarrow F_{2k+1} - B$
3 Find maximal $A \subseteq A_k$ s.t. $S - A_k + B_{k+1} + A \in \mathcal{I}_1$
4 $A_k \leftarrow A_k - A$, $R_{2k} \leftarrow R_{2k} + A$

Algorithm 4 `RefineBA`(k) (from [11, Algorithm 2]); called `Refine2` in [18, Algorithm 10].

1 Find maximal $B \subseteq B_k$ s.t. $S - (D_{2k} - R_{2k}) + B \in \mathcal{I}_2$
2 $R_{2k-1} \leftarrow R_{2k-1} + B_k \setminus B$, $B_k \leftarrow B$
3 Find maximal $A \subseteq F_{2k}$ s.t. $S - (D_{2k} - R_{2k}) + B_k + A \in \mathcal{I}_2$
4 $A_k \leftarrow A_k + F_{2k} \setminus A$, $F_{2k} \leftarrow A$

In Blikstad's algorithm, we maintain three types of elements in each layer (see [11, Section 3.1]):

- *fresh.* Denoted by $F_i \subseteq D_i$. These elements are candidates that could be added to the partial augmenting set.
- *selected.* Denoted by B_1, A_1, B_2. These elements form the current partial augmenting set $\Pi = (B_1, A_1, B_2)$.
- *removed.* Denoted by $R_i \subseteq D_i$. These elements are deemed useless, and then we can disregard them.

For convenience, we also define imaginary layers D_0 and D_4 with $A_0 = R_0 = F_0 = D_0 = A_2 = R_4 = F_4 = D_4 = \emptyset$.

In `Refine` (Algorithm 5), we iteratively apply `RefineAB` (Algorithm 3) and `RefineBA` (Algorithm 4), and then update the types of elements. Initially, we begin with all elements being *fresh*. Elements can change their types from *fresh* $\rightarrow$ *selected* $\rightarrow$ *removed*, but their types cannot be changed in the other direction. Note that an element of type *fresh* can change to type *removed* without first going through type *selected*.

Now, by providing how Algorithm 2 can be implemented in the semi-streaming model, we give a proof of Theorem 3.

Proof of Theorem 3. We show how Algorithm 2 can be implemented using $O(\frac{r_1+r_2}{\varepsilon})$ memory and $O(\frac{1}{\varepsilon})$ passes in the semi-streaming model.

In our algorithm, We first compute a maximal common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ using one pass of the stream. Note that we can easily compute a maximal set in a single pass. We store the set S explicitly using $O(r)$ memory.

Then, we compute D_2 in the exchange graph $G(S)$ using one pass of the stream. For each $v \in V \setminus S$, if $S + v \in \mathcal{I}_1$, then we find all elements $u \in S$ such that $S + v - u \in \mathcal{I}_2$ and add them to D_2.

We store D_2 explicitly using $O(r)$ memory. In our algorithm, we store the types of elements (i.e., F_2, A_1, R_2) in D_2 explicitly using $O(r)$ memory. Whenever an element $v \in V \setminus S$ arrives in the stream, we can determine whether $v \in D_1$, $v \in D_3$, or otherwise, by checking whether $S + v \in \mathcal{I}_1$ and whether there exists an element $u \in D_2$ such that $S + v - u \in \mathcal{I}_1$. Thus, we can maintain the distance layers D_1 and D_3 implicitly.

Next, we apply `Refine` (Algorithm 5) $O(1/\varepsilon)$ times. In our implementation of `Refine` in the semi-streaming model, we replace Lines 3–8 in `Refine` with `UpdateABA`(k) (Algorithm 6). By Claim 19, our implementation of `Refine` can also correctly find a desired partial augmenting set.

Algorithm 5 `Refine` (from [11, Algorithms 3 and 4]).

```
1  for k = 1, 0 do
2  |  RefineBA(k + 1)
3  |  for x ∈ F_{2k+1} do
4  |  |  if S − A_k + B_{k+1} + x ∈ I_1 then
5  |  |  |  if S − A_{k+1} − F_{2k+2} + B_{k+1} + x ∈ I_2 then
6  |  |  |  |  B_{k+1} ← B_{k+1} + x, F_{2k+1} ← F_{2k+1} − x
7  |  |  |  else
8  |  |  |  |  R_{2k+1} ← R_{2k+1} + x, F_{2k+1} ← F_{2k+1} − x
9  |  RefineBA(k + 1)
10 |  RefineAB(k)
```

Algorithm 6 `UpdateABA`(k) (implementation of Lines 3–8 in `Refine` (Algorithm 5) in the semi-streaming model).

```
1  for x ∈ F_{2k+1} do                                          // first pass
2  |  if S − A_k + B_{k+1} + x ∈ I_1 and S − A_{k+1} − F_{2k+2} + B_{k+1} + x ∈ I_2 then
3  |  |  B_{k+1} ← B_{k+1} + x, F_{2k+1} ← F_{2k+1} − x
4  Store the current A_k and B_{k+1} as A_k^(i) and B_{k+1}^(i), respectively.
   // Assume that this procedure is the i-th call of UpdateABA(k) in the
      entire matroid intersection algorithm.
5  for x ∈ F_{2k+1} do                                          // second pass
6  |  if S − A_k + B_{k+1} + x ∈ I_1 then
7  |  |  R_{2k+1} ← R_{2k+1} + x, F_{2k+1} ← F_{2k+1} − x
```

▷ Claim 19. Even if we replace Lines 3–8 in `Refine` with `UpdateABA`(k) (Algorithm 6), the procedure `Refine` can correctly find a partial augmenting set that satisfies the conditions (i)–(iii) in Lemma 15.

Proof. Consider the order of the elements in F_{2k+1} satisfying the following condition: an element added to B_{k+1} in the first for loop of `UpdateABA`(k) appears before any element that is not added to B_{k+1} in that loop. Suppose that, in the execution of Lines 3–8 in `Refine`, the elements in F_{2k+1} arrive in this order. Then, the changes of types of elements in D_{2k+1} by `UpdateABA`(k) call are exactly same as by the execution of Lines 3–8 in `Refine` (Algorithm 5). Regardless of the order of the elements in the for loop in Line 3, the implementation of `Refine` in Algorithm 5 can find a desired partial augmenting set. Therefore, `Refine` with Lines 3–8 replaced by `UpdateABA`(k) correctly finds a desired partial augmenting set, which completes the proof. ◁

Now, we show that each call of the modified `Refine` can be implemented with $O(1)$ passes. Since we can compute a maximal set in a single pass, both `RefineAB` (Algorithm 3) and `RefineBA` (Algorithm 4) can be implemented with 2 passes. In addition, `UpdateABA` (Algorithm 6) can also be implemented with 2 passes.

Next, we show that we can maintain the types of elements (i.e., $F_1, B_1, R_1, F_3, B_2, R_3$) in D_1 and D_3 implicitly. To do this, we explicitly maintain the following:

- We store the current partial augmenting set (B_1, A_1, B_2). Since the number of selected elements is always at most $r_1 + r_2$, we use $O(r_1 + r_2)$ memory to store it.

- We store all removed elements whose type has changed from *selected* to *removed*. Only `RefineAB` and `RefineBA` change the types of elements from *selected* to *removed*. In each call of `RefineAB` and `RefineBA`, this change occurs only for at most $r_1 + r_2$ elements. Since the number of `RefineAB` and `RefineBA` calls is $O(1/\varepsilon)$, we use $O((r_1 + r_2)/\varepsilon)$ memory to store them in the entire algorithm.
- We store $A_k^{(i)}$ and $B_{k+1}^{(i)}$ just after the first for loop in `UpdateABA` (Algorithm 6). Since the number of `UpdateABA` calls is $O(1/\varepsilon)$, we use $O((r_1 + r_2)/\varepsilon)$ memory to store them in the entire algorithm.

Here, we note that, for an element whose current type is *removed*, there are only the following two cases.

- The type has changed from *selected* to *removed* by `RefineAB` or `RefineBA`.
- The type has changed from *fresh* to *removed* by `UpdateABA`.

Whenever an element $v \in D_1 \cup D_3$ arrives, we can identify the current type of v in the following way:

- If the current type of v is *selected*, then we can easily identify it.
- If the current type of v is *removed*, then we identify it in the following way:
  - In the case where the type of v has changed from *selected* to *removed* by `RefineAB` or `RefineBA`, we can conclude that the current type of v is *removed*.
  - In the case where the type of v has changed from *fresh* to *removed* by Line 7 in `UpdateABA` (Algorithm 6), we identify it by simulating all previous executions of the second for loop in `UpdateABA`. More precisely, let C be the number of `UpdateABA` called so far. If $v \in D_{2k+1}$ and there is an index $i \leq C$ such that there are $A_k^{(i)}$ and $B_{k+1}^{(i)}$ such that $S - A_k^{(i)} + B_{k+1}^{(i)} + v \in \mathcal{I}_1$, then we conclude that the current type of v is *removed*.
- Otherwise, the current type of v is *fresh*.

Therefore, each call of the modified `Refine` can be implemented with $O(1)$ passes.

After applying `Refine` $O(1/\varepsilon)$ times, we obtain a partial augmenting set $\Phi = (B_1, A_1, B_2)$ such that $|B_1| - |B_2| \leq \varepsilon \bar{r}$. Then, by applying Lemma 14 for the obtained partial augmenting set $\Phi = (B_1, A_1, B_2)$, we obtain an augmenting set $\Pi = (B_1', A_1', B_2')$ such that $B_1' \subseteq B_1$, $A_1' \subseteq A_1$ and $B_2 = B_2'$. The algorithm in Lemma 14 can be implemented without an additional pass of the stream, because it only accesses the set S and the partial augmenting set $\Phi = (B_1, A_1, B_2)$; see [18, Proof of Lemma 38].

Finally, we output the set $S \oplus \Pi$. By the same argument as Proof of Theorem 1, the set $S \oplus \Pi$ is a $(2/3 - \varepsilon)$-approximate solution, which completes the proof. ◀

7 Concluding Remarks

We have observed that a $(2/3 - \varepsilon)$-approximate solution can be obtained by terminating Blikstad's [11] algorithm early. Then, we obtained a deterministic nearly-linear-independence-query $(2/3 - \varepsilon)$-approximation algorithm for the matroid intersection problem.

We were also able to use this observation in the streaming model of computation. Then, we obtain a $(2/3 - \varepsilon)$-approximation constant-pass semi-streaming algorithm for the matroid intersection problem. This result is a generalization of the $(2/3 - \varepsilon)$-approximation bipartite matching semi-streaming algorithm of Feigenbaum–Kannan–McGregor–Suri–Zhang [27], who initiated the study of matching algorithms in the semi-streaming model. Soon after, McGregor [50] first obtained a $(1 - \varepsilon)$-approximation constant-pass semi-streaming algorithm

for the maximum matching problem. Since then, there have been many studies for $(1-\varepsilon)$-approximation constant-pass semi-streaming algorithms for the maximum matching problem in the literature [1,8,24,25,30,31,40,42,59]. Then, it is natural to ask whether we can obtain a $(1-\varepsilon)$-approximation constant-pass semi-streaming algorithm for the matroid intersection problem.

In the current Blikstad's algorithm for finding an augmenting set, it is necessary to perfectly find an augmenting set of length $2k$ before finding an augmenting set of length $2k+2$. By finding a maximal common independent set, we can perfectly find an augmenting set of length 2. This means that we can start from a state where shortest augmenting paths have length 4 or greater. In our current algorithm, since we do not perfectly find an augmenting set of length 4, we are unable to find an augmenting set of length 6 or greater. Thus, our current algorithm achieves only a $(2/3-\varepsilon)$-approximation rather than a $(1-1/k-\varepsilon)$-approximation. However, we believe our study could be an important step to obtain a deterministic nearly-linear-independence-query $(1-\varepsilon)$-approximation algorithm.

References

1 Kook Jin Ahn and Sudipto Guha. Laminar families and metric embeddings: Non-bipartite maximum matching problem in the semi-streaming model. *arXiv preprint arXiv:1104.4058*, 2011. doi:10.48550/arXiv.1104.4058.

2 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Information and Computation*, 222:59–79, 2013. Announced at ICALP 2011. doi:10.1016/j.ic.2012.10.006.

3 Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. *ACM Transactions on Parallel Computing (TOPC)*, 4(4):1–40, 2018. Announced at SPAA 2015. doi:10.1145/3154855.

4 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.

5 Sepehr Assadi. A simple $(1-\varepsilon)$-approximation semi-streaming algorithm for maximum (weighted) matching. In *Proceedings of the 7th Symposium on Simplicity in Algorithms (SOSA 2024)*, pages 337–354. SIAM, 2024. doi:10.1137/1.9781611977936.31.

6 Sepehr Assadi and Soheil Behnezhad. Beating two-thirds for random-order streaming matching. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198, pages 19:1–19:13, 2021. doi:10.4230/LIPIcs.ICALP.2021.19.

7 Sepehr Assadi, Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Semi-streaming bipartite matching in fewer passes and optimal space. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2022)*, pages 627–669. SIAM, 2022. doi:10.1137/1.9781611977073.29.

8 Sepehr Assadi, S Cliff Liu, and Robert E Tarjan. An auction algorithm for bipartite matching in streaming and massively parallel computation models. In *Proceedings of the 4th Symposium on Simplicity in Algorithms (SOSA 2021)*, pages 165–171. SIAM, 2021. doi:10.1137/1.9781611976496.18.

9 Kristóf Bérczi, Tamás Király, Yutaro Yamaguchi, and Yu Yokoi. Matroid intersection under restricted oracles. *SIAM Journal on Discrete Mathematics*, 37(2):1311–1330, 2023. doi:10.1137/22m152579x.

10 Aaron Bernstein. Improved bounds for matching in random-order streams. *Theory of Computing Systems*, pages 1–15, 2023. Announced at ICALP 2020. doi:10.1007/s00224-023-10155-7.

11 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.

12 Joakim Blikstad. Sublinear-round parallel matroid intersection. In *Proceedings of the 49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229, pages 25:1–25:17, 2022. doi:10.4230/LIPIcs.ICALP.2022.25.

13 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.

14 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.

15 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.

16 Niv Buchbinder and Moran Feldman. Deterministic algorithm and faster algorithm for submodular maximization subject to a matroid constraint. In *Proceedings of the 65th Annual Symposium on Foundations of Computer Science (FOCS 2024)*, pages 700–712. IEEE, 2024. doi:10.1109/FOCS61266.2024.00050.

17 Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Mathematical Programming*, 154:225–247, 2015. Announced at IPCO 2014. doi:10.1007/s10107-015-0900-7.

18 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.

19 Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015)*, pages 318–330. Springer, 2015. doi:10.1007/978-3-662-47672-7_26.

20 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.

21 William H Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM Journal on Computing*, 15(4):948–957, 1986. doi:10.1137/0215066.

22 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Structures and Their Applications*, pages 69–87, 1970.

23 Jack Edmonds. Matroid intersection. In *Annals of Discrete Mathematics*, volume 4, pages 39–49. Elsevier, 1979. doi:10.1016/S0167-5060(08)70817-3.

24 Sebastian Eggert, Lasse Kliemann, Peter Munstermann, and Anand Srivastav. Bipartite matching in the semi-streaming model. *Algorithmica*, 63(1):490–508, 2012. doi:10.1007/s00453-011-9556-8.

25 Sebastian Eggert, Lasse Kliemann, and Anand Srivastav. Bipartite graph matchings in the semi-streaming model. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA 2009)*, pages 492–503. Springer, 2009. doi:10.1007/978-3-642-04128-0_44.

26 Hossein Esfandiari, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. Finding large matchings in semi-streaming. In *Proceedings of the 16th International Conference on Data Mining Workshops (ICDMW 2016)*, pages 608–614. IEEE, 2016. doi:10.1109/ICDMW.2016.0092.

27 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005. Announced at ICALP 2004. doi:10.1016/j.tcs.2005.09.013.

28 Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. *Advances in Neural Information Processing Systems 31: Proceedings of the 32th Annual Conference on Neural Information Processing Systems (Neurips 2018)*, 31, 2018.

29 Moran Feldman and Ariel Szarf. Maximum matching sans maximal matching: A new approach for finding maximum matchings in the data stream model. *Algorithmica*, 86(4):1173–1209, 2024. Announced at APPROX/RANDOM 2022. doi:10.1007/s00453-023-01190-4.

30 Manuela Fischer, Slobodan Mitrović, and Jara Uitto. Deterministic $(1+\varepsilon)$-approximate maximum matching with poly $(1/\varepsilon)$ passes in the semi-streaming model and beyond. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022)*, pages 248–260, 2022. doi:10.1145/3519935.3520039.

31 Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In *Proceedings of the 38th Symposium on Principles of Distributed Computing (PODC 2019)*, pages 491–500, 2019. doi:10.1145/3293611.3331603.

32 Paritosh Garg, Linus Jordan, and Ola Svensson. Semi-streaming algorithms for submodular matroid intersection. *Mathematical Programming*, 197(2):967–990, 2023. Announced at IPCO 2021. doi:10.1007/s10107-022-01858-9.

33 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 468–485. SIAM, 2012. doi:10.1137/1.9781611973099.41.

34 Guru Prashanth Guruganesh and Sahil Singla. Online matroid intersection: Beating half for random arrival. In *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization (IPCO 2017)*, pages 241–253. Springer, 2017. doi:10.1007/978-3-319-59250-3_20.

35 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.

36 Chien-Chung Huang, Naonori Kakimura, and Naoyuki Kamiyama. Exact and approximation algorithms for weighted matroid intersection. *Mathematical Programming*, 177(1-2):85–112, 2019. Announced at SODA 2016. doi:10.1007/s10107-018-1260-x.

37 Chien-Chung Huang and François Sellier. Robust sparsification for matroid intersection with applications. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2024)*, pages 2916–2940. SIAM, 2024. doi:10.1137/1.9781611977912.104.

38 Chien-Chung Huang, Theophile Thiery, and Justin Ward. Improved multi-pass streaming algorithms for submodular maximization with matroid constraints. In *Proceedings of the 23th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2020) and the 24th International Workshop on Randomized and Computation (RANDOM 2020)*, volume 176, pages 62:1–62:19, 2020. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.62.

39 Chien-Chung Huang and Justin Ward. FPT-algorithms for the ℓ-matchoid problem with a coverage objective. *SIAM Journal on Discrete Mathematics*, 37(2):1053–1078, 2023. doi:10.1137/21M1442267.

40 Shang-En Huang and Hsin-Hao Su. $(1-\varepsilon)$-approximate maximum weighted matching in $\text{poly}(1/\varepsilon, \log n)$ time in the distributed and parallel settings. In *Proceedings of the 42nd Symposium on Principles of Distributed Computing (PODC 2023)*, pages 44–54, 2023. doi:10.1145/3583668.3594570.

41 Sagar Kale and Sumedh Tirodkar. Maximum matching in two, three, and a few more passes over graph streams. In *Proceedings of the 20th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2017) and the 21st International Workshop on Randomized and Computation (RANDOM 2017)*, volume 81, pages 15:1–15:21, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.15.

42 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*, pages 1679–1697. SIAM, 2013. doi:10.1137/1.9781611973105.121.

43 Yusuke Kobayashi and Tatsuya Terao. Subquadratic submodular maximization with a general matroid constraint. In *Proceedings of the 51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, volume 297, pages 100:1–100:19, 2024. doi:10.4230/LIPIcs.ICALP.2024.100.

44 Christian Konrad. A simple augmentation method for matchings with applications to streaming algorithms. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, pages 74–1, 2018. doi:10.4230/LIPIcs.MFCS.2018.74.

45 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Proceedings of the 15th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2012) and the 16th International Workshop on Randomized and Computation (RANDOM 2012)*, pages 231–242. Springer, 2012. doi:10.1007/978-3-642-32512-0_20.

46 Christian Konrad, Kheeran K. Naidu, and Arun Steward. Maximum matching via maximal matching queries. In *Proceedings of the 40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, volume 254, pages 41:1–41:22, 2023. doi:10.4230/LIPIcs.STACS.2023.41.

47 Bernhard H Korte and Jens Vygen. *Combinatorial optimization*. Springer, third edition, 2006.

48 Eugene L Lawler. Matroid intersection algorithms. *Mathematical Programming*, 9(1):31–56, 1975. doi:10.1007/BF01681329.

49 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.

50 Andrew McGregor. Finding graph matchings in data streams. In *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2005) and the 9th International Workshop on Randomized and Computation (RANDOM 2005)*, pages 170–181. Springer, 2005. doi:10.1007/11538462_15.

51 Kazuo Murota. *Matrices and matroids for systems analysis*. Springer, 2010.

52 Huy L Nguyen. A note on Cunningham's algorithm for matroid intersection. arXiv preprint arXiv:1904.04129, 2019. doi:10.48550/arXiv.1904.04129.

53 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.

54 Kent Quanrud. Faster exact and approximation algorithms for packing and covering matroids via push-relabel. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2024)*, pages 2305–2336. SIAM, 2024. doi:10.1137/1.9781611977912.82.

55 András Recski. *Matroid theory and its applications in electric network theory and in statics*, volume 6. Springer Science & Business Media, 2013. doi:10.1007/978-3-662-22143-3.

56 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

57 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.

58 Tatsuya Terao. Faster matroid partition algorithms. *ACM Transactions on Algorithms (TALG)*, 21(2):1–26, 2025. Announced at ICALP 2023. doi:10.1145/3707208.

59 Sumedh Tirodkar. Deterministic algorithms for maximum matching on general graphs in the semi-streaming model. In *Proceedings of the 38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.39.

60 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.

61 Vignesh Viswanathan and Yair Zick. A general framework for fair allocation under matroid rank valuations. In *Proceedings of the 24th ACM Conference on Economics and Computation (EC 2023)*, pages 1129–1152, 2023. doi:10.1145/3580507.3597675.