

18th International Symposium on Parameterized and Exact Computation

IPEC 2023, September 6–8, 2023, Amsterdam, The Netherlands

Edited by

Neeldhara Misra

Magnus Wahlström



Editors

Neeldhara Misra 

IIT Gandhinagar, India
neeldhara.m@iitgn.ac.in

Magnus Wahlström 

Royal Holloway, University of London, UK
Magnus.Wahlstrom@rhul.ac.uk

ACM Classification 2012

Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Approximation algorithms analysis; Theory of computation → Graph algorithms analysis; Theory of computation → Mathematical optimization; Theory of computation → Algorithm design techniques

ISBN 978-3-95977-305-8

Published online and open access by

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <https://www.dagstuhl.de/dagpub/978-3-95977-305-8>.

Publication date

December, 2023

Bibliographic information published by the Deutsche Nationalbibliothek

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

License

This work is licensed under a Creative Commons Attribution 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.IPEC.2023.0

ISBN 978-3-95977-305-8

ISSN 1868-8969

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

LIPICs – Leibniz International Proceedings in Informatics

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

Editorial Board

- Luca Aceto (*Chair*, Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Roberto Di Cosmo (Inria and Université de Paris, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Daniel Král' (Masaryk University, Brno, CZ)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN)
- Anca Muscholl (University of Bordeaux, FR)
- Chih-Hao Luke Ong (University of Oxford, GB)
- Phillip Rogaway (University of California, Davis, US)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Raimund Seidel (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)
- Pierre Senellart (ENS, Université PSL, Paris, FR)

ISSN 1868-8969

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

■ Contents

Preface	
<i>Neeldhara Misra and Magnus Wahlström</i>	0:ix–0:x
Program Committees	
.....	0:xi
Reviewers	
.....	0:xiii
Authors	
.....	0:xv–0:xviii

Regular Papers

Kernelizing Temporal Exploration Problems	
<i>Emmanuel Arrighi, Fedor V. Fomin, Petr A. Golovach, and Petra Wolf</i>	1:1–1:18
Cluster Editing with Overlapping Communities	
<i>Emmanuel Arrighi, Matthias Bentert, Pål Grønås Drange, Blair D. Sullivan, and Petra Wolf</i>	2:1–2:12
Existential Second-Order Logic over Graphs: Parameterized Complexity	
<i>Max Bannach, Florian Chudigiewitsch, and Till Tantau</i>	3:1–3:15
On the Complexity of Finding a Sparse Connected Spanning Subgraph in a Non-Uniform Failure Model	
<i>Matthias Bentert, Jannik Schestag, and Frank Sommer</i>	4:1–4:12
Difference Determines the Degree: Structural Kernelizations of Component Order Connectivity	
<i>Sriram Bhyravarapu, Satyabrata Jana, Saket Saurabh, and Roohani Sharma</i>	5:1–5:14
The Parameterised Complexity Of Integer Multicommodity Flow	
<i>Hans L. Bodlaender, Isja Mannens, Jelle J. Oostveen, Sukanya Pandey, and Erik Jan van Leeuwen</i>	6:1–6:19
Treewidth Is NP-Complete on Cubic Graphs	
<i>Hans L. Bodlaender, Édouard Bonnet, Lars Jaffke, Dušan Knop, Paloma T. Lima, Martin Milanič, Sebastian Ordyniak, Sukanya Pandey, and Ondřej Suchý</i>	7:1–7:13
Stretch-Width	
<i>Édouard Bonnet and Julien Duron</i>	8:1–8:15
Minimum Separator Reconfiguration	
<i>Guilherme C. M. Gomes, Clément Legrand-Duchesne, Reem Mahmoud, Amer E. Mouawad, Yoshio Okamoto, Vinicius F. dos Santos, and Tom C. van der Zanden</i>	9:1–9:12
Kernels for the Disjoint Paths Problem on Subclasses of Chordal Graphs	
<i>Juhi Chaudhary, Harmender Gahlawat, Michal Włodarczyk, and Meirav Zehavi</i> ..	10:1–10:22

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström



Leibniz International Proceedings in Informatics

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

Parameterized Complexity Classification for Interval Constraints <i>Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, Marcin Pilipczuk, and Roohani Sharma</i>	11:1–11:19
An FPT Algorithm for Temporal Graph Untangling <i>Riccardo Dondi and Manuel Lafond</i>	12:1–12:16
Budgeted Matroid Maximization: a Parameterized Viewpoint <i>Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai</i>	13:1–13:17
Computing Complexity Measures of Degenerate Graphs <i>Pål Grønås Drange, Patrick Greaves, Irene Muzi, and Felix Reidl</i>	14:1–14:21
An Improved Kernelization Algorithm for TRIVIALY PERFECT EDITING <i>Maël Dumas and Anthony Perez</i>	15:1–15:17
From Data Completion to Problems on Hypercubes: A Parameterized Analysis of the Independent Set Problem <i>Eduard Eiben, Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider</i>	16:1–16:14
Approximate Monotone Local Search for Weighted Problems <i>Barış Can Esmer, Ariel Kulik, Dániel Marx, Daniel Neuen, and Roohani Sharma</i>	17:1–17:23
Consistency Checking Problems: A Gateway to Parameterized Sample Complexity <i>Robert Ganian, Liana Khazaliya, and Kirill Simonov</i>	18:1–18:17
Finding Degree-Constrained Acyclic Orientations <i>Jaroslav Garvardt, Malte Renken, Jannik Schestag, and Mathias Weller</i>	19:1–19:14
Graph Clustering Problems Under the Lens of Parameterized Local Search <i>Jaroslav Garvardt, Nils Morawietz, André Nichterlein, and Mathias Weller</i>	20:1–20:19
Bandwidth Parameterized by Cluster Vertex Deletion Number <i>Tatsuya Gima, Eun Jung Kim, Noleen Köhler, Nikolaos Melissinos, and Manolis Vasilakis</i>	21:1–21:15
Collective Graph Exploration Parameterized by Vertex Cover <i>Siddharth Gupta, Guy Sa’ar, and Meirav Zehavi</i>	22:1–22:18
Drawn Tree Decomposition: New Approach for Graph Drawing Problems <i>Siddharth Gupta, Guy Sa’ar, and Meirav Zehavi</i>	23:1–23:22
Single Machine Scheduling with Few Deadlines <i>Klaus Heeger, Danny Hermelin, and Dvir Shabtay</i>	24:1–24:15
Twin-Width of Graphs with Tree-Structured Decompositions <i>Irene Heinrich and Simon Raßmann</i>	25:1–25:17
Dynamic Programming on Bipartite Tree Decompositions <i>Lars Jaffke, Laure Morelle, Ignasi Sau, and Dimitrios M. Thilikos</i>	26:1–26:22
Kernelization for Counting Problems on Graphs: Preserving the Number of Minimum Solutions <i>Bart M. P. Jansen and Bart van der Steenhoven</i>	27:1–27:15
On the Parameterized Complexity of MULTIWAY NEAR-SEPARATOR <i>Bart M. P. Jansen and Shivesh K. Roy</i>	28:1–28:18

Sunflowers Meet Sparsity: A Linear-Vertex Kernel for Weighted Clique-Packing on Sparse Graphs <i>Bart M. P. Jansen and Shivesh K. Roy</i>	29:1–29:13
How Can We Maximize Phylogenetic Diversity? Parameterized Approaches for Networks <i>Mark Jones and Jannik Schestag</i>	30:1–30:12
Sidestepping Barriers for Dominating Set in Parameterized Complexity <i>Ioannis Koutis, Michał Włodarczyk, and Meirav Zehavi</i>	31:1–31:17
Approximate Turing Kernelization and Lower Bounds for Domination Problems <i>Stefan Kratsch and Pascal Kunz</i>	32:1–32:17
A Parameterized Approximation Scheme for the Geometric Knapsack Problem with Wide Items <i>Mathieu Mari, Timothé Picavet, and Michał Pilipczuk</i>	33:1–33:20
A Contraction-Recursive Algorithm for Treewidth <i>Hisao Tamaki</i>	34:1–34:15

PACE Solver Descriptions

PACE Solver Description: The PACE 2023 Parameterized Algorithms and Computational Experiments Challenge: Twinwidth <i>Max Bannach and Sebastian Berndt</i>	35:1–35:14
PACE Solver Description: Hydra Prime <i>Yosuke Mizutani, David Dursteler, and Blair D. Sullivan</i>	36:1–36:5
PACE Solver Description: Exact (GUTHMI) and Heuristic (GUTHM) <i>Alexander Leonhardt, Holger Dell, Anselm Haak, Frank Kammer, Johannes Meintrup, Ulrich Meyer, and Manuel Penschuck</i>	37:1–37:7
PACE Solver Description: Touiwidth <i>Gaétan Berthe, Yoann Coudert–Osmont, Alexander Dobler, Laure Morelle, Amadeus Reinald, and Mathis Rocton</i>	38:1–38:4
PACE Solver Description: Zygoty <i>Emmanuel Arrighi, Pål Grønås Drange, Kenneth Langedal, Farhad Vadiée, Martin Vatshelle, and Petra Wolf</i>	39:1–39:3
PACE Solver Description: RedAlert - Heuristic Track <i>Édouard Bonnet and Julien Duron</i>	40:1–40:5

■ Preface

The International Symposium on Parameterized and Exact Computation (IPEC, formerly IWPEC) is a series of international symposia covering research in all aspects of parameterized and exact algorithms and complexity. It started in 2004 as a biennial workshop and became an annual event in 2009. Previous iterations of the symposium were:

- 2004 Bergen, Norway
- 2006 Zürich, Switzerland
- 2008 Victoria, Canada
- 2009 Copenhagen, Denmark
- 2010 Chennai, India
- 2011 Saarbrücken, Germany
- 2012 Ljubljana, Slovenia
- 2013 Sophia Antipolis, France
- 2014 Wrocław, Poland
- 2015 Patras, Greece
- 2016 Aarhus, Denmark
- 2017 Vienna, Austria
- 2018 Helsinki, Finland
- 2019 Munich, Germany
- 2020 virtual / Hong Kong, China
- 2021 virtual / Lisbon, Portugal
- 2022 Potsdam, Germany

This volume contains the papers presented at IPEC 2023: the 18th International Symposium on Parameterized and Exact Computation. IPEC 2023 was held on September 6–8 (Wed to Fri) as part of ALGO 2023, and took place in Amsterdam, the Netherlands at Centrum Wiskunde & Informatica (CWI). In response to the call for papers, 85 extended abstracts were registered, of which 10 were withdrawn or otherwise failed to submit a full version. The resulting number of 75 full submissions represents a significant increase in interest in the conference compared to previous years. 34 papers were ultimately selected for presentation at the conference and inclusion in these proceedings. The reviews were performed in a double-blind fashion, and there were 106 external reviews out of a total of 223 reviews.

The **Best Paper Award** was given to Hans L. Bodlaender (Utrecht University), Isja Mannens (Utrecht University), Jelle Oostveen (Utrecht University), Sukanya Pandey (Utrecht University) and Erik Jan van Leeuwen (Utrecht University) for their paper “The Parameterised Complexity of Integer Multicommodity Flow”. The **Best Student Paper Award** was given to Stefan Kratsch (Humboldt-Universität zu Berlin) and Pascal Kunz (Humboldt-Universität zu Berlin) for their paper “Approximate Turing kernelization and lower bounds for domination problems”. The **EATCS-IPEC Nerode Prize** was given to Marek Cygan (University of Warsaw and Nomagic), Jesper Nederlof (Utrecht University), Marcin Pilipczuk (University of Warsaw), Michał Pilipczuk (University of Warsaw), Johan M. M. van Rooij (Utrecht University) and Jakub Onufry Wojtaszczyk (Google) for their paper “Solving Connectivity Problems Parameterized by Treewidth in Single Exponential Time”. IPEC 2023 hosted an award ceremony with a talk given jointly by Michał Pilipczuk and Johan M. M. van Rooij. The Nerode Prize committee consisted of Fedor Fomin (chair; University of Bergen), Thore Husfeldt (IT University of Copenhagen) and Sang-il Oum (Korea Advanced Institute of Science and Technology). Tuukka Korhonen (University of Bergen) presented an **invited tutorial** on “New methods in FPT algorithms for treewidth”. Finally, IPEC 2023 hosted the award ceremony of the eighth Parameterized Algorithms and Computational Experiments (PACE) challenge. These proceedings contain a report on the **PACE 2023** challenge and brief communications of the winners about their solvers.

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström



Leibniz International Proceedings in Informatics

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

0:x Preface

We thank the program committee and the external reviewers for their commitment in the paper selection process. We also thank all the authors who submitted their work. We are grateful to the local organizers of ALGO 2023 for the local arrangements.

Neeldhara Misra and Magnus Wahlström
Gandhinagar and London, October 2022

■ Program Committees

IPEC 2023 Program Committee

- Akanksha Agrawal (IIT Madras, India)
- Cristina Bazgan (Paris Dauphine University, France)
- Robert Brederick (TU Clausthal, Germany)
- Eduard Eiben (Royal Holloway, University of London, UK)
- Archontia Giannopoulou (University of Athens, Greece)
- Pallavi Jain (IIT Jodhpur, India)
- Bart M. P. Jansen (Eindhoven University of Technology, Netherlands)
- Mark Jones (TU Delft, Netherlands)
- Christian Knauer (Universität Bayreuth, Germany)
- Dusan Knop (Czech Technical University, Czech Republic)
- Bingkai Lin (Nanjing University, China)
- Neeldhara Misra (IIT Gandhinagar, India) (co-chair)
- André Nichterlein (TU Berlin, Germany)
- Sebastian Ordyniak (University of Leeds, UK)
- Fahad Panolan (IIT Hyderabad, India)
- Daniel Paulusma (Durham University, UK)
- R.B. Sandeep (IIT Dharwad, India)
- Magnus Wahlström (Royal Holloway, University of London, UK) (co-chair)

IPEC 2023 Steering Committee

- Holger Dell (2021-24, chair)
- Fedor Fomin (2021-24)
- Petr Golovach (2020-23)
- Łukasz Kowalik (2022-25)
- Daniel Marx (2020-23)
- Neeldhara Misra (2022-25)
- Jesper Nederlof (2021-24)
- Magnus Wahlström (2022-25)
- Meirav Zehavi (2020-23)

PACE 2023 Program Committee

- Max Bannach (Universität zu Lübeck)
- Sebastian Berndt (Universität zu Lübeck)





■ List of External Reviewers

- Duncan Adamson
- Junggho Ahn
- Dhanyamol Antony
- Emmanuel Arrighi
- Pradeesha Ashok
- Rémy Belmonte
- Matthias Bentert
- Steffen van Bergerem
- Magnus Bordewich
- Sergio Cabello
- Huairui Chu
- Alexis de Colnet
- Alex Crane
- Radu Curticapean
- Argyrios Deligkas
- Riccardo Dondi
- Pål Grønås Drange
- Foivos Fioravantes
- Till Fluschnik
- Vincent Froese
- Harmender Gahlawat
- Jaroslav Garvardt
- Serge Gaspers
- Panos Giannopoulos
- Tiger-Lily Goldsmith
- Petr Golovach
- Maximilian Gorsky
- Siddharth Gupta
- Sushmita Gupta
- Thekla Hamm
- Tesshu Hanaka
- Samuel Hand
- Klaus Heeger
- Leo van Iersel
- Tanmay Inamdar
- Yuni Iwamasa
- Lars Jaffke
- Satyabrata Jana
- Rasmus Ibsen-Jensen
- Andrzej Kaczmarczyk
- Lawqueen Kanesh
- Anjeneya Swami Kare
- Tomohiro Koana
- Christian Komusiewicz
- Stefan Kratsch
- Simon Krogmann
- Ariel Kulik
- O-Joung Kwon
- Noleen Köhler
- Matyáš Křišťan
- Junjie Luo
- Jayakrishnan Madathil
- Diptapriyo Majumdar
- Barnaby Martin
- Andrés López Martínez
- Rogers Mathew
- Filippos Mavropoulos
- Marcelo Garlet Milani
- Pranabendu Misra
- Matthias Mnich
- Hendrik Molter
- Nils Morawietz
- Amer Mouawad
- Anthony Perez
- Théo Pierron
- Evangelos Protopapas
- Lars Rohwedder
- Sanjukta Roy
- Abhishek Sahu
- Saket Saurabh
- Sanjay Seetharaman
- Sebastian Siebertz
- Kirill Simonov
- Fiona Skerman
- Ramanujan M. Sridharan
- Ramanujan Sridharan
- Giannos Stamoulis
- Fabian Stehn
- Céline Swennenhuis
- Prafullkumar Tale
- Ioan Todinca
- Vikash Tripathi
- Oxana Tsidulko
- Rao B V
- Ruben F.A. Verhaegh
- Christopher Weyand
- Hongxun Wu
- Karol Węgrzycki
- Jie Xue
- Yongjie Yang
- Chihao Zhang
- Dimitris Zoros




■ List of Authors

Emmanuel Arrighi  (1, 2, 39)
University of Bergen, Norway; University of
Trier, Germany


Max Bannach  (3, 35)
European Space Agency, Advanced Concepts
Team, Noordwijk, The Netherlands


Matthias Bentert (2, 4)
University of Bergen, Norway


Sebastian Berndt  (35)
Institute for Theoretical Computer Science,
University of Lübeck, Germany


Gaétan Berthe  (38)
LIRMM, CNRS, Université de Montpellier,
France


Sriram Bhyravarapu (5)
The Institute of Mathematical Sciences, HBNI,
Chennai, India


Hans L. Bodlaender  (6, 7)
Utrecht University, The Netherlands

Édouard Bonnet  (7, 8, 40)
LIP, ENS Lyon, France

Guilherme C. M. Gomes  (9)
Department of Computer Science, Federal,
University of Minas Gerais, Belo Horizonte,
Brazil

Tom C. van der Zanden  (9)
Department of Data Analytics and
Digitalisation, Maastricht University, The
Netherlands


Juhi Chaudhary  (10)
Ben-Gurion University of the Negev, Beersheba,
Israel


Florian Chudigiewitsch  (3)
Universität zu Lübeck, Germany

Yoann Coudert-Osmont (38)
Université de Lorraine, CNRS, Inria, LORIA,
France

Konrad K. Dabrowski  (11)
School of Computing, Newcastle University, UK

Holger Dell  (37)
Goethe University Frankfurt, Germany


Alexander Dobler  (38)
Algorithms and Complexity Group, TU Wien,
Austria

Riccardo Dondi  (12)
Università degli studi di Bergamo, Italy


Ilan Doron-Arad (13)
Computer Science Department, Technion, Haifa,
Israel


Pål Grønås Drange  (2, 14, 39)
University of Bergen, Norway

Maël Dumas (15)
Univ. Orléans, INSA Centre Val de Loire, LIFO
EA 4022, F-45067 Orléans, France

Julien Duron  (8, 40)
Univ Lyon, CNRS, ENS de Lyon, Université
Claude Bernard Lyon 1, LIP UMR5668, France


David Dursteler  (36)
University of Utah, Salt Lake City, UT, USA


Eduard Eiben  (16)
Department of Computer Science, Royal
Holloway, University of London, Egham, UK

Barış Can Esmer  (17)
CISPA Helmholtz Center for Information
Security, Saarbrücken, Germany; Saarbrücken
Graduate School of Computer Science, Saarland
Informatics Campus, Germany


Vinicius F. dos Santos  (9)
Department of Computer Science, Federal,
University of Minas Gerais, Belo Horizonte,
Brazil

Fedor V. Fomin  (1)
University of Bergen, Norway

Harmender Gahlawat  (10)
Ben-Gurion University of the Negev, Beersheba,
Israel

Robert Ganian  (16, 18)
Algorithms and Complexity Group, TU Wien,
Austria

Jaroslav Garvardt  (19, 20)
Philipps-Universität Marburg, Germany;
Friedrich-Schiller-Universität Jena, Germany

Tatsuya Gima  (21)
JSPS Research Fellow, Nagoya University, Japan

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström



Leibniz International Proceedings in Informatics

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

- Petr A. Golovach  (1)
University of Bergen, Norway
- Patrick Greaves (14)
Birkbeck, University of London, UK
- Siddharth Gupta  (22, 23)
BITS Pilani, Goa Campus, India
- Anselm Haak (37)
Goethe University Frankfurt, Germany
- Klaus Heeger  (24)
Department of Industrial Engineering and
Management, Ben-Gurion University of the
Negev, Beer-Sheva, Israel
- Irene Heinrich  (25)
Technische Universität Darmstadt, Germany
- Danny Hermelin  (24)
Department of Industrial Engineering and
Management, Ben-Gurion University of the
Negev, Beer-Sheva, Israel
- Lars Jaffke  (7, 26)
University of Bergen, Norway
- Satyabrata Jana  (5)
The Institute of Mathematical Sciences, HBNI,
Chennai, India
- Bart M. P. Jansen  (27, 28, 29)
Eindhoven University of Technology, The
Netherlands
- Mark Jones  (30)
TU Delft, The Netherlands
- Peter Jonsson  (11)
Department of Computer and Information
Science, Linköping University, Sweden
- Frank Kammer  (37)
THM, University of Applied Sciences ,
Mittelhessen, Gießen, Germany
- Iyad Kanj  (16)
School of Computing, DePaul University,
Chicago, IL, USA
- Liana Khazaliya  (18)
Technische Universität Wien, Austria
- Eun Jung Kim  (21)
Université Paris-Dauphine, PSL University,
CNRS UMR7243, LAMSADE, Paris, France
- Dušan Knop  (7)
Czech Technical University in Prague, Czech
Republic
- Ioannis Koutis  (31)
New Jersey Institute of Technology, NJ, USA
- Stefan Kratsch  (32)
Algorithm Engineering, Humboldt-Universität
zu Berlin, Germany
- Ariel Kulik  (13, 17)
CISPA Helmholtz Center for Information
Security, Saarbrücken, Germany
- Pascal Kunz  (32)
Algorithm Engineering, Humboldt-Universität
zu Berlin, Germany
- Noleen Köhler  (21)
Université Paris-Dauphine, PSL University,
CNRS UMR7243, LAMSADE, Paris, France
- Manuel Lafond  (12)
Université de Sherbrooke, Canada
- Kenneth Langedal  (39)
University of Bergen, Norway
- Clément Legrand-Duchesne  (9)
LaBRI, CNRS, Université de Bordeaux, France
- Alexander Leonhardt (37)
Goethe University Frankfurt, Germany
- Paloma T. Lima  (7)
IT University of Copenhagen, Denmark
- Reem Mahmoud (9)
Virginia Commonwealth University, Richmond,
VA, USA
- Isja Mannens  (6)
Utrecht University, The Netherlands
- Mathieu Mari (33)
Institute of Informatics, University of Warsaw,
Poland; IDEAS-NCBR, Warsaw, Poland
- Dániel Marx  (17)
CISPA Helmholtz Center for Information
Security, Saarbrücken, Germany
- Johannes Meintrup  (37)
THM, University of Applied Sciences ,
Mittelhessen, Gießen, Germany
- Nikolaos Melissinos  (21)
Department of Theoretical Computer Science,
Faculty of Information Technology, Czech
Technical University in Prague, Czech Republic
- Ulrich Meyer  (37)
Goethe University Frankfurt, Germany

- Martin Milanič  (7)
FAMNIT and IAM, University of Primorska,
Koper, Slovenia
- Yosuke Mizutani  (36)
University of Utah, Salt Lake City, UT, USA
- Nils Morawietz  (20)
Institute of Computer Science, Friedrich Schiller
University Jena, Germany
- Laure Morelle  (26, 38)
LIRMM, Université de Montpellier, CNRS,
France
- Amer E. Mouawad  (9)
Department of Computer Science, American
University of Beirut, Beirut, Lebanon
- Irene Muzi  (14)
Birkbeck, University of London, UK
- Daniel Neuen  (17)
University of Bremen, Germany
- André Nichterlein  (20)
Technische Universität Berlin, Germany
- Yoshio Okamoto  (9)
Graduate School of Informatics and Engineering,
The University of Electro-Communications,
Chofu, Japan
- Jelle J. Oostveen  (6)
Utrecht University, The Netherlands
- Sebastian Ordyniak  (7, 11, 16)
University of Leeds, UK
- George Osipov  (11)
Department of Computer and Information
Science, Linköping University, Sweden
- Sukanya Pandey  (6, 7)
Utrecht University, The Netherlands
- Manuel Penschuck  (37)
Goethe University Frankfurt, Germany
- Anthony Perez (15)
Univ. Orléans, INSA Centre Val de Loire, LIFO
EA 4022, F-45067 Orléans, France
- Timothé Picavet  (33)
ENS de Lyon, France; Aalto University, Finland
- Marcin Pilipczuk  (11)
Faculty of Mathematics, Informatics and
Mechanics, University of Warsaw, Poland; IT
University Copenhagen, Denmark
- Michał Pilipczuk  (33)
Institute of Informatics, University of Warsaw,
Poland
- Simon Raßmann  (25)
Technische Universität Darmstadt, Germany
- Felix Reidl  (14)
Birkbeck, University of London, UK
- Amadeus Reinald  (38)
LIRMM, CNRS, Université de Montpellier,
France
- Malte Renken  (19)
Technische Universität Berlin, Germany
- Mathis Rocton  (38)
Algorithms and Complexity Group, TU Wien,
Austria
- Shivesh K. Roy  (28, 29)
Eindhoven University of Technology, The
Netherlands
- Guy Sa'ar (22, 23)
Ben Gurion University of the Negev, Beersheba,
Israel
- Ignasi Sau (26)
LIRMM, Université de Montpellier, CNRS,
France
- Saket Saurabh  (5)
The Institute of Mathematical Sciences, HBNI,
Chennai, India; University of Bergen, Norway
- Jannik Schestag  (4, 19, 30)
Faculteit Elektrotechniek, Wiskunde en
Informatica, TU Delft, The Netherlands;
Fakultät für Mathematik und Informatik,
Friedrich-Schiller-Universität Jena, Germany
- Dvir Shabtay  (24)
Department of Industrial Engineering and
Management, Ben-Gurion University of the
Negev, Beer-Sheva, Israel
- Hadas Shachnai (13)
Computer Science Department, Technion, Haifa,
Israel
- Roohani Sharma  (5, 11, 17)
Max Planck Institute for Informatics, Saarland
Informatics Campus, Saarbrücken, Germany
- Kirill Simonov  (18)
Hasso Plattner Institute, Universität Potsdam,
Germany

- Frank Sommer  (4)
Fakultät für Mathematik und Informatik,
Friedrich-Schiller-Universität Jena, Germany
- Ondřej Suchý  (7)
Czech Technical University in Prague, Czech
Republic
- Blair D. Sullivan  (2, 36)
University of Utah, Salt Lake City, UT, USA
- Stefan Szeider  (16)
Algorithms and Complexity Group, TU Wien,
Austria
- Hisao Tamaki  (34)
Meiji University, Kawasaki, Japan
- Till Tantau  (3)
Universität zu Lübeck, Germany
- Dimitrios M. Thilikos (26)
LIRMM, Université de Montpellier, CNRS,
France
- Farhad Vadiée  (39)
University of Bergen, Norway
- Bart van der Steenhoven  (27)
Eindhoven University of Technology, The
Netherlands
- Erik Jan van Leeuwen  (6)
Utrecht University, The Netherlands
- Manolis Vasilakis  (21)
Université Paris-Dauphine, PSL University,
CNRS UMR7243, LAMSADE, Paris, France
- Martin Vatshelle (39)
University of Bergen, Norway
- Mathias Weller  (19, 20)
Technische Universität Berlin, Germany
- Petra Wolf  (1, 2, 39)
University of Bergen, Norway
- Michał Włodarczyk  (10)
University of Warsaw, Poland
- Michał Włodarczyk  (31)
University of Warsaw, Poland
- Meirav Zehavi  (10, 22, 23, 31)
Ben-Gurion University of the Negev, Beersheba,
Israel

Kernelizing Temporal Exploration Problems

Emmanuel Arrighi   

University of Bergen, Norway
University of Trier, Germany

Fedor V. Fomin  

University of Bergen, Norway

Petr A. Golovach  

University of Bergen, Norway

Petra Wolf   

University of Bergen, Norway

Abstract

We study the kernelization of exploration problems on temporal graphs. A temporal graph consists of a finite sequence of snapshot graphs $\mathcal{G} = (G_1, G_2, \dots, G_L)$ that share a common vertex set but might have different edge sets. The non-strict temporal exploration problem (NS-TEXP for short) introduced by Erlebach and Spooner, asks if a single agent can visit all vertices of a given temporal graph where the edges traversed by the agent are present in non-strict monotonous time steps, i.e., the agent can move along the edges of a snapshot graph with infinite speed. The exploration must at the latest be completed in the last snapshot graph. The optimization variant of this problem is the k -ARB NS-TEXP problem, where the agent's task is to visit at least k vertices of the temporal graph. We show that under standard computational complexity assumptions, neither of the problems NS-TEXP nor k -ARB NS-TEXP allow for polynomial kernels in the standard parameters: number of vertices n , lifetime L , number of vertices to visit k , and maximal number of connected components per time step γ ; as well as in the combined parameters $L + k$, $L + \gamma$, and $k + \gamma$. On the way to establishing these lower bounds, we answer a couple of questions left open by Erlebach and Spooner.

We also initiate the study of structural kernelization by identifying a new parameter of a temporal graph $p(\mathcal{G}) = \sum_{i=1}^L (|E(G_i)|) - |V(G)| + 1$. Informally, this parameter measures how dynamic the temporal graph is. Our main algorithmic result is the construction of a polynomial (in $p(\mathcal{G})$) kernel for the more general WEIGHTED k -ARB NS-TEXP problem, where weights are assigned to the vertices and the task is to find a temporal walk of weight at least k .

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Problems, reductions and completeness

Keywords and phrases Temporal graph, temporal exploration, computational complexity, kernel

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.1

Related Version *Full Version:* <https://arxiv.org/abs/2302.10110>

Funding The research leading to these results has received funding from the Research Council of Norway via the project BWCA (grant no. 314528).

1 Introduction

We investigate the kernelization of exploration and connectivity tasks in temporal networks. While kernelization, and in particular, structural kernelization, appears to be a successful approach for addressing many optimization problems on *static* graphs, its applications in temporal graphs are less impressive, to say the least. A reasonable explanation for this (we will provide some evidence of that later) is that most of the structural parameters of static graphs, like treewidth, size of a feedback vertex set, or the vertex cover number, do not seem to be useful when it comes to dynamic settings. This brings us to the following question.



© Emmanuel Arrighi, Fedor V. Fomin, Petr A. Golovach, and Petra Wolf;
licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 1; pp. 1:1–1:18

Leibniz International Proceedings in Informatics



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

What structure of dynamic graphs could be “helpful” for kernelization algorithms?

We propose a new structural parameter of a temporal graph estimating how “dynamic” is the temporal graph. Our main result is an algorithm for the exploration problem on a temporal graph that produces a kernel whose size is polynomial in the new parameter. Before proceeding with the formal statement of the problem and our results, we provide a short overview of temporal graphs and kernelization.

Temporal exploration. Many networks considered nowadays show an inherently dynamic behavior, for instance connectivity in mobile ad-hoc networks, relationships in a social network, or accessibility of services in the internet. Classical graphs do not suffice to model those dynamic behaviors, which led to an intensive study of so-called *temporal graphs*. In general, temporal graphs are graphs that change over time. There are several models in the literature that consider different types of dynamic behaviors and encodings of the temporal graphs. Here, we consider a temporal graph \mathcal{G} to consist of a sequence of *snapshot* graphs G_1, G_2, \dots, G_L that share a common set of vertices V but might have different edge sets E_1, E_2, \dots, E_L . We call the graph $G = (V, \bigcup_{i=1}^L E_i)$ the *underlying graph* of \mathcal{G} . We can think of the temporal graph \mathcal{G} as the graph G where only a subset of edges are present at a certain time step. The number of snapshots L is commonly referred to as the *lifetime* of \mathcal{G} . Due to its relevance in modeling dynamic systems, a huge variety of classical graph problems have been generalized to temporal graphs. We refer to [10, 26, 29, 31] for an introduction to temporal graphs and their associated combinatorial problems.

Graph exploration is a fundamental problem in the realm of graph theory and has been extensively studied since its introduction by Shannon in 1951 [32]. The question of whether a given graph can be explored by a single agent was generalized to temporal graphs by Michail and Spirakis [30]. They showed that deciding whether a single agent can visit all vertices of a temporal graph within a given number of time steps while crossing only one edge per time step is an NP-hard problem. This hardness motivated the study of the parameterized complexity of the temporal exploration problem performed by Erlebach and Spooner in [19].

While the temporal exploration problem, TEXP for short, allows the agent to cross only one edge per time step, in some scenarios, the network only changes slowly, way slower than the speed of an agent. Then, it is natural to assume that the agent can travel with infinite speed and explore the whole connected component in which he is currently located in a single time step. Such scenarios arise for instance in delay-tolerant networks [9]. Erlebach and Spooner generalized TEXP to allow the agent to travel with infinite speed in [17] and called this problem the *non-strict temporal exploration problem*, NS-TEXP for short. In NS-TEXP the task is to identify whether an agent can visit all vertices of the graph starting from an initial vertex v according to the following procedure. At step 1, the agent visits all vertices of the connected component C_1 of the snapshot graph G_1 containing v . At step $i \in \{2, \dots, L\}$, the agent selects to explore the vertices of one of the components C_i of G_i that have a non-empty intersection with the component explored by the agent at step $(i - 1)$.

Shifting to an infinite exploration speed allows for exploration times that can be significantly shorter than the number of vertices. While for classical graphs, the question if a graph can be explored with infinite speed simply reduces to the question of connectivity for temporal graphs, NS-TEXP is NP-complete [17]. The parameterized complexity of NS-TEXP was studied by Erlebach and Spooner in [19] where FPT algorithms for the parameter *lifetime* L of the temporal graph (equivalently number of time steps during which the temporal exploration must be performed) was shown. They also studied an optimization

variant of NS-TEXP, where not all but at least k arbitrary vertices must be visited during the lifetime of the graph. This variant is called k -ARB NS-TEXP. An FPT algorithm with parameter k solving k -ARB NS-TEXP was obtained in [19]. In the long version of this study [20], it was stated as an open question whether NS-TEXP is in FPT or at least in XP for the parameter maximal number of connected components in a time step, γ , and, whether k -ARB NS-TEXP is in FPT for parameter L . These open problems were also stated at the 2022 ICALP satellite workshop *Algorithmic Aspects of Temporal Graphs V*. In this work, we will answer both of these questions negatively.

Kernelization. Informally, a kernelization algorithm is a preprocessing algorithm that consecutively applies various data reduction rules in order to shrink the instance size of a parameterized problem. Kernelization is one of the major research domains of parameterized complexity and many important advances in the area are on kernelization. These advances include general algorithmic findings on problems admitting kernels of polynomial size and frameworks for ruling out polynomial kernels under certain complexity-theoretic assumptions. We refer to the book [22] for an overview of the area. A fruitful approach in kernelization (for static graphs) is the study of the impact of various structural measurements (i.e., different than just the total input size or expected solution size) on the problem complexity. Such structural parameterizations, like the minimum size of a feedback vertex set, of a vertex cover, or the treedepth, measure the non-triviality of the instance [7, 24, 25, 34].

Our contribution. Our focus is on the possibility of kernelization for the problems NS-TEXP and k -ARB NS-TEXP. In their work on the parameterized complexity of these problems, Erlebach and Spooner in [19, 20] established fixed-parameter tractability for some combinations of these problems and “standard” parameterizations like number of vertices n , lifetime L , number of vertices to visit k , and $L + \gamma$, where γ is the maximal number of connected components per time step, see Table 1. As the first step, we rule out the existence of polynomial kernels for both problems when parameterized by each of these standard parameters. Moreover, our lower bounds hold even for “combined” parameters $L + k$, $L + \gamma$, and $k + \gamma$. On the way to establish our lower bounds for kernelization, we resolve two open problems from the parameterized study of NS-TEXP and k -ARB NS-TEXP posed by Erlebach and Spooner in [19, 20]. Namely, we show that NS-TEXP is NP-complete for constant values of $\gamma \geq 5$ and that k -ARB NS-TEXP is W[1]-hard parameterized by L .

This motivates the study of structural kernelization of the exploration problems. While at the first glance, the most natural direction would be to explore the structure of the underlying graph of the temporal graph \mathcal{G} , this direction does not seem to bring new algorithmic results. The reason is that our lower bounds on NS-TEXP and k -ARB NS-TEXP hold for temporal graphs with *very* restricted underlying graphs. In particular, we show that the problems remain NP-hard even when the underlying graph is a tree with vertex cover number at most 2. Thus a reasonable structural parameterization, in this case, should capture not only the “static” structure of the underlying graph but also the “dynamics” of the edges in \mathcal{G} . With this in mind, we introduce the new parameter of a temporal graph $\mathcal{G} = (G_1, G_2, \dots, G_L)$, $p(\mathcal{G}) = \sum_{i=1}^L (|E(G_i)|) - |V(\mathcal{G})| + 1$.

Note that if $\sum_{i=1}^L (|E(G_i)|) < |V(\mathcal{G})| - 1$, then the underlying graph is disconnected and hence the temporal graph \mathcal{G} cannot be explored. The parameter $p(\mathcal{G})$ bounds both the structure and the dynamic of \mathcal{G} . Indeed, consider the multigraph on $V(\mathcal{G})$ obtained by inserting a copy of the edge uv for each occurrence of uv in a snapshot. Then, $p(\mathcal{G})$ corresponds to the size of a minimum feedback edge set of this multigraph. As an example,

■ **Table 1** Overview of the parameterized complexity of NS-TEXP and k -ARB NS-TEXP. Our contribution is highlighted in red. The results stating that there is no polynomial kernel rely on the assumption that $\text{NP} \not\subseteq \text{coNP/poly}$. Here, for a temporal graph \mathcal{G} , n is the number of vertices, L the lifetime, γ is the maximal number of components per time step, and $p = p(\mathcal{G})$. For entries marked with $*$, we get a compression to the more general variant WEIGHTED k -ARB NS-TEXP.

Param.	NS-TEXP		k -ARB NS-TEXP	
	FPT	Kernel	FPT	Kernel
p	$2^{\mathcal{O}(p)}(nL)^{\mathcal{O}(1)}$	$\mathcal{O}(p^4)^*$	$2^{\mathcal{O}(p)}(nL)^{\mathcal{O}(1)}$	$\mathcal{O}(p^4)^*$
n	$\mathcal{O}^*((2e)^n n^{\log n})$ [20]	no poly kernel	FPT in k [20]	no poly kernel
L	$\mathcal{O}^*(L(L!)^2)$ [19]	no poly kernel	W[1]-hard	no poly kernel
k	-	-	$\mathcal{O}^*((2e)^k k^{\log k})$ [20]	no poly kernel
$L + k$	-	-	FPT in k [20]	no poly kernel
γ	in P for ≤ 2 [20], NP-hard for ≥ 5	-	in P for ≤ 2 [20], NP-hard for ≥ 5	-
$L + \gamma$	FPT in L [19]	no poly kernel for $\gamma \geq 6$	$\mathcal{O}(\gamma^L n^{\mathcal{O}(1)})$	no poly kernel for $\gamma \geq 6$
$k + \gamma$	-	-	FPT in k [20]	no poly kernel

$p(\mathcal{G}) = 0$ means that the underlying graph of \mathcal{G} is a tree and each edge of the underlying graph appears in exactly one snapshot. Hence, our parameter describes how far the graph is from such a tree. Our main result is a polynomial kernel for the more general problem WEIGHTED k -ARB NS-TEXP in the parameter $p = p(\mathcal{G})$. In WEIGHTED k -ARB NS-TEXP, the vertices contain weights and the task is to find a temporal walk that visits vertices with a total sum of weights of at least k for some given integer k . The obtained kernel is of size $\mathcal{O}(p^4)$ and contains a number of vertices that is linear in the parameter p .

Our results are summarized in Table 1. Due to space constraints, the proofs of some results are either sketched or omitted in this extended abstract. The full details could be found in the full arXiv version [3].

Further related work. Michail and Spirakis [30] introduced the TEXP problem and showed that the problem is NP-complete when no restrictions are placed on the input. They proposed considering the problem under the *always-connected* assumption that requires that the temporal graph is connected in every time step. Erlebach et al. [14] followed this proposition and showed that for always-connected temporal graphs, computing a foremost exploration schedule is NP-hard to approximate with ratio $\mathcal{O}(n^{1-\epsilon})$, for every $\epsilon > 0$. Bodlaender and van der Zanden [6] showed that the TEXP problem, when restricted to always-connected temporal graphs whose underlying graph has pathwidth at most 2, remains NP-complete. Bounds on the length of exploration schedules were given in [1, 14, 33] for temporal graphs where the underlying graph has a certain structure, and in [15, 16, 18] for temporal graphs where each snapshot graph has a certain structure.

Erlebach and Spooner [19] studied the TEXP problem from a parameterized perspective. Based on the color coding technique [2], they gave an FPT algorithm parameterized by k for the problems k -ARB TEXP and the non-strict variant k -ARB NS-TEXP. They also gave an FPT algorithm parameterized by the lifetime L of the temporal graph for the problems TEXP and NS-TEXP. In the respective long version [20], Erlebach and Spooner further studied the parameter maximal number of connected components per time step γ and showed

that TEXP is NP-hard for $\gamma = 1$, but NS-TEXP is solvable in polynomial time for $\gamma \leq 2$. The NS-TEXP problem was introduced and studied by Erlebach and Spooner [17]. Among other things, they showed NP-completeness of the general problem, as well as $\mathcal{O}(n^{1/2-\epsilon})$ and $\mathcal{O}(n^{1-\epsilon})$ -inapproximability for computing a foremost exploration schedule under the assumption that the number of time steps required to move between any pair of vertices is bounded by 2, resp. 3. Bumpus and Meeks [8] considered the parameterized complexity of a graph exploration problem that asks no longer to visit all vertices, but to traverse all *edges* of the underlying graph exactly once. They observed that for natural structural parameters of the underlying graph, the problem does not admit FPT algorithms. Similarly, Kunz et al. [28] obtained several hardness results for structural parameters of the underlying graph when studying the parameterized complexity of the problem of finding temporally disjoint paths and walks, a problem introduced by Klobas et al. [27]. Their obtained W[1]-hardness in the parameter number of vertices, that holds even for instances where the underlying graph is a star, indicates that simply considering structural parameters of the underlying graph is not sufficient to obtain FPT algorithms for temporal graph problems.

2 Preliminaries

Notations. We denote by \mathbb{Z} the set of integers, by \mathbb{N} the set of natural numbers including 0, by $\mathbb{N}_{>0}$ the set of positive integers, and by \mathbb{Q} the set of rational numbers. Let n be a positive integer, we denote with $[n]$ the set $\{1, 2, \dots, n\}$. Given a vector $w = (w_1, w_2, \dots, w_r) \in \mathbb{Q}^r$, we let $\|w\|_\infty = \max_{i \in [r]} |w_i|$ and $\|w\|_1 = \sum_{i \in [r]} |w_i|$. Given $x \in \mathbb{Z}$, we let $\text{sign}(x)$ be $+$ if $x \geq 0$ and $-$ otherwise.

Graphs. We consider a graph $G = (V, E)$ to be a static undirected graph. Given a graph G , we denote by $V(G)$ the set of vertices of G , by $E(G)$ the set of edges of G , by $N_G(x)$ the neighbors of a vertex x in G , and by $\gamma(G)$ the number of connected components of G . Let $G = (V, E)$ be a graph, given a subset of vertices $X \subseteq V(G)$ and a subset of edges $E' \subseteq E$, we define the following operation on G : $G[X] = (X, \{uv \in E \mid u, v \in X\})$, $G - X = G[V \setminus X]$ and $G - E' = (V, E \setminus E')$. We call a *walk* in a graph G an alternating sequence $W = v_0, e_1, v_1, \dots, e_r, v_r$ of vertices and edges, where $v_0, \dots, v_r \in V(G)$, $e_1, \dots, e_r \in E(G)$ and $e_i = v_{i-1}v_i$ for $i \in [r]$; note that W may visit the same vertices and edges several times. A walk without repeated vertices is called a *path*. We say that v_0 and v_r are *end-vertices* of W and W is a (v_0, v_r) -*walk*. We use $V(W) \subseteq V(G)$ to denote the set of vertices of G visited by W and denote by $E(W)$ the set of edges of G that are in W . Given a walk $W = v_0, e_1, v_1, \dots, e_r, v_r$, for $0 \leq i \leq j \leq r$, we call the sequence $W' = v_i, e_{i+1}, \dots, e_j, v_j$ a *subwalk* of W .

Temporal graphs. A *temporal graph* \mathcal{G} over a set of vertices V is a sequence $\mathcal{G} = (G_1, G_2, \dots, G_L)$ of graphs such that for all $t \in [L]$, $V(G_t) = V$. We call L the *lifetime* of \mathcal{G} and for $t \in [L]$, we call $G_t = (V, E_t)$ the *snapshot graph* of \mathcal{G} at *time step* t . We might refer to G_t as $\mathcal{G}(t)$. We call $G = (V, E)$ with $E = \bigcup_{t \in [L]} E_t$ the *underlying graph* of \mathcal{G} . We denote by $V(\mathcal{G})$ the set of vertices of \mathcal{G} and by $\gamma(\mathcal{G}) = \max_{t \in [L]} \gamma(G_t)$ the maximum number of connected components over all snapshot graphs of \mathcal{G} . We write V and γ if the graph or temporal graph is clear from the context. For a temporal graph $\mathcal{G} = (G_1, \dots, G_L)$, we define the *total number of edge appearances* as $\mathfrak{m}(\mathcal{G}) = \sum_{i=1}^L |E(G_i)|$; we use \mathfrak{m} to denote this value if \mathcal{G} is clear from the context.

In the following, we will be interested in temporal walks where the agent has infinite speed within a snapshot graph. Those temporal walks are called *non-strict temporal walks* in [19]. In [19], a non-strict temporal walk is defined as a sequence of connected components. However, it is more convenient for us to consider a *non-strict temporal walk* as a *monotone walk* in the underlying graph. For a walk $W = v_0, e_1, v_1, \dots, e_r, v_r$ in the underlying graph G of $\mathcal{G} = (G_1, \dots, G_L)$, we say that W is *monotone* if there are $t_1, \dots, t_r \in [L]$ with $1 \leq t_1 \leq \dots \leq t_r \leq L$ such that $e_i \in E(G_{t_i})$ for each $i \in [r]$. The definition of a non-strict temporal walk immediately implies the following observation.

► **Observation 1.** *Given a temporal graph \mathcal{G} and a vertex x , \mathcal{G} has a non-strict temporal walk starting in x that visits exactly the vertices of a set X if and only if the underlying graph G has a monotone (x, y) -walk W for some $y \in V(G)$ such that $X = V(W)$.*

For the remainder of this paper, we are mainly interested in the computational problem of finding monotone walks that visit all vertices of a temporal graph. We might also call such a walk an *exploration schedule* or simply an *exploration*.

► **Definition 2** (NON-STRICT TEMPORAL EXPLORATION (NS-TEXP)).

Input: Temporal graph $\mathcal{G} = (G_1, G_2, \dots, G_L)$, vertex $v \in V(\mathcal{G})$.

Question: Is there a monotone walk in \mathcal{G} that starts in v and visits all vertices in $V(\mathcal{G})$?

We further consider a more general variant of the NS-TEXP problem, called k -ARB NS-TEXP, where we ask for a monotone walk that visits at least k vertices (instead of $|V|$).

► **Definition 3** (k -ARBITRARY NON-STRICT TEMPORAL EXPLORATION (k -ARB NS-TEXP)).

Input: Temporal graph $\mathcal{G} = (G_1, G_2, \dots, G_L)$, vertex $v \in V(\mathcal{G})$, integer k .

Question: Is there a monotone walk in \mathcal{G} that starts in v and visits at least k vertices?

We will further generalize the problem by considering the weighted version of k -ARB NS-TEXP. We refer to this problem as WEIGHTED k -ARB NS-TEXP.

► **Definition 4** (WEIGHTED k -ARBITRARY NON-STRICT TEMPORAL EXPLORATION (WEIGHTED k -ARB NS-TEXP)).

Input: Temporal graph $\mathcal{G} = (G_1, G_2, \dots, G_L)$, positive-valued weight function $w: V(\mathcal{G}) \rightarrow \mathbb{N}_{>0}$, vertex $v \in V(\mathcal{G})$, integer k .

Question: Is there a monotone walk W in \mathcal{G} such that W starts in v and $w(V(W)) = \sum_{u \in V(W)} w(u) \geq k$?

Note that k -ARB NS-TEXP is a special case of WEIGHTED k -ARB NS-TEXP.

Parameterized complexity and kernelization. We refer to books [22] and [12] for an introduction to the field.

A *data reduction rule*, or simply, *reduction rule*, for a parameterized problem Q is a function $\phi: \Sigma^* \times \mathbb{N} \rightarrow \Sigma^* \times \mathbb{N}$ that maps an instance (I, k) of Q to an equivalent instance (I', k') of Q such that ϕ is computable in time polynomial in $|I|$ and k . We say that two instances of Q are *equivalent* if the following holds: $(I, k) \in Q$ if and only if $(I', k') \in Q$. We refer to this property of the reduction rule ϕ , that it translates an instance to an equivalent one, as to the *safeness* of the reduction rule.

Informally, *kernelization* is a preprocessing algorithm that consecutively applies various data reduction rules in order to shrink the instance size as much as possible. A preprocessing algorithm takes as input an instance $(I, k) \in \Sigma^* \times \mathbb{N}$ of Q , works in polynomial in $|I|$

and k time, and returns an equivalent instance (I', k') of Q . The quality of a preprocessing algorithm \mathcal{A} is measured by the size of the output. More precisely, the *output size* of a preprocessing algorithm \mathcal{A} is a function $\text{size}_{\mathcal{A}}: \mathbb{N} \rightarrow \mathbb{N} \cup \{\infty\}$ defined as follows:

$$\text{size}_{\mathcal{A}}(k) = \sup\{|I'| + k' : (I', k') = \mathcal{A}(I, k), I \in \Sigma^*\}.$$

A *kernelization algorithm*, or simply a *kernel*, for a parameterized problem Q is a preprocessing algorithm \mathcal{A} that, given an instance (I, k) of Q , works in polynomial in $|I|$ and k time and returns an equivalent instance (I', k') of Q such that $\text{size}_{\mathcal{A}}(k) \leq g(k)$ for some computable function $g: \mathbb{N} \rightarrow \mathbb{N}$. It is said that $g(\cdot)$ is the *size* of a kernel. If $g(\cdot)$ is a polynomial function, then we say that Q admits a *polynomial kernel*. It is well-known that a decidable parameterized problem is FPT if and only if it admits a kernel [13]. However, up to some reasonable complexity assumptions, there are FPT problems that have no polynomial kernels. In particular, we are using the cross-composition technique introduced in [4] and [5] to show that a parameterized problem does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

3 Lower Bounds

It was stated as an open problem in [19], whether NS-TEXP is in FPT with parameter γ (i.e. maximum number of components in any snapshot graph). We answer this question negatively by showing that NS-TEXP is NP-complete for $\gamma \geq 5$.

► **Theorem 5.** *NS-TEXP is NP-complete for $\gamma \geq 5$.*

Proof. We give a reduction from the satisfiability problem SAT which asks if a given Boolean formula has a satisfying variable assignment. Let $\varphi = \{c_1, c_2, \dots, c_m\}$ be a Boolean formula in conjunctive normal form over the variable set $X = \{x_1, x_2, \dots, x_n\}$. We construct from φ a temporal graph \mathcal{G} , where each snapshot graph has 4 or 5 connected components, such that \mathcal{G} has a monotone walk that visits all vertices in $V(\mathcal{G})$ if and only if φ is satisfiable.

The main idea of the construction is the following. In $V(\mathcal{G})$, we have a vertex for each clause, vertices $\hat{x}_i, x_i, \neg x_i$ for each variable x_i , and one single control vertex \hat{c} . The sequence of snapshot graphs alternates between having four and five connected components. In the case of four connected components, one component collects all clause vertices, one component collects all variable vertices x and $\neg x$, one component collects all not yet processed control vertices \hat{x} and \hat{c} , and one component collects all processed control vertices \hat{x} . In the case of five connected components, an additional component collects a negative literal $\neg x$ of a variable x together with all clauses containing $\neg x$. In this step, the clauses containing x are incorporated into the variable component, which still contains x . This will allow us to choose a variable assignment with the exploration schedule. In the next time step, and only in this time step, the control vertex \hat{x} is contained in the variable component. For all later time steps, \hat{x} is contained in the component collecting the processed control vertices. Thereby, the control vertices ensure that we return to the component containing the variables in each snapshot consisting of four connected components.

We now give a more formal construction. For this construction, we are interested in the connected components of each snapshot graph but not on the actual structure of the connected components. For simplicity, we define the connected components as set of vertices and assume that each connected component forms a clique. For a subset $S \subseteq V$, we denote by $K(S)$ the set of all possible edges between vertices in S . Thereby, $(S, K(S))$ is a clique. Let $\mathcal{G} = (G_1, G_2, \dots, G_L)$ with $L = 2n + 1$ be the temporal graph constructed

from $\varphi = \{c_1, c_2, \dots, c_m\}$. We define $V(\mathcal{G}) = \{c_1, c_2, \dots, c_m\} \cup \{x_i, \neg x_i, \widehat{x}_i \mid x_i \in X\} \cup \{\widehat{c}\}$. For $E(G_1) = K(C_{1,1}) \cup K(C_{1,2}) \cup K(C_{1,3})$ we set $C_{1,1} = \{\widehat{c}\} \cup \{\widehat{x}_i \mid x_i \in X\}$, $C_{1,2} = \{c_1, c_2, \dots, c_m\}$, and $C_{1,3} = \{x_i, \neg x_i \mid x_i \in X\}$. The start vertex is set to x_1 .

For the next time step, we define the edges of the snapshot graph as $E(G_2) = K(C_{2,1}) \cup K(C_{2,2}) \cup K(C_{2,3}) \cup K(C_{2,4})$ where $C_{2,1} = C_{1,1} = \{\widehat{c}\} \cup \{\widehat{x}_i \mid x_i \in X\}$, $C_{2,2} = \{c_1, c_2, \dots, c_m\} \setminus \{c \in \varphi \mid x_1 \in c \vee \neg x_1 \in c\}$, $C_{2,3} = (\{x_i, \neg x_i \mid x_i \in X\} \setminus \{\neg x_1\}) \cup \{c \in \varphi \mid x_1 \in c\}$ and $C_{2,4} = \{\neg x_1\} \cup \{c \in \varphi \mid \neg x_1 \in c\}$. The intuition is that the exploration schedule visits the vertices in $C_{2,3}$ after visiting the vertices in $C_{1,3}$ if the variable x_1 gets assigned with **true** and otherwise, visits $C_{2,4}$ after $C_{1,3}$ if x_1 gets assigned with **false**. Note that no other connected component is reachable from $C_{1,3}$ as only $C_{2,3}$ and $C_{2,4}$ have nonempty intersection with $C_{1,3}$.

In the next time step, we force the exploration schedule to return to the third connected component by passing the control vertex \widehat{x}_1 through this connected component. Therefore, we define $E(G_3) = K(C_{3,1}) \cup K(C_{3,2}) \cup K(C_{3,3})$ with $C_{3,1} = (\{\widehat{c}\} \cup \{\widehat{x}_i \mid x_i \in X\}) \setminus \{\widehat{x}_1\}$, $C_{3,2} = \{c_1, c_2, \dots, c_m\}$, and $C_{3,3} = \{x_i, \neg x_i \mid x_i \in X\} \cup \{\widehat{x}_1\}$.

For the remaining time steps, we alternate between the structure of the second and third time step with an additional connected component that collects the already processed control vertices \widehat{x}_i . As this connected component is monotone growing, it acts as a sink for the temporal walk, i.e., we could not leave this connected component if we ever enter it. Additionally, the first connected component containing the not yet processed control vertices is monotone shrinking, making it non-accessible from the start vertex of the temporal walk. The idea is now that the last control vertex \widehat{c} will only leave the first component into the variable component in the last time step enforcing us to not go into the sink component of processed control vertices and thereby enforcing the exploration schedule to return to the variable component for each odd time step $t \geq 3$. We formalize this by defining the snapshot graphs G_j for $3 < j < 2n + 1$ as follows.

First consider the case that j is even. We define $E(G_j) = K(C_{j,1}) \cup K(C_{j,2}) \cup K(C_{j,3}) \cup K(C_{j,4}) \cup K(C_{j,5})$ with $C_{j,1} = \{\widehat{c}\} \cup \{\widehat{x}_i \mid x_i \in X, i \geq \frac{j}{2}\}$, $C_{j,2} = \{c_1, c_2, \dots, c_m\} \setminus \{c \in \varphi \mid x_{\frac{j}{2}} \in c \vee \neg x_{\frac{j}{2}} \in c\}$, $C_{j,3} = (\{x_i, \neg x_i \mid x_i \in X\} \setminus \{\neg x_{\frac{j}{2}}\}) \cup \{c \in \varphi \mid x_{\frac{j}{2}} \in c\}$, $C_{j,4} = \{\neg x_{\frac{j}{2}}\} \cup \{c \in \varphi \mid \neg x_{\frac{j}{2}} \in c\}$, and $C_{j,5} = \{\widehat{x}_i \mid x_i \in X, i < \frac{j}{2}\}$.

For the case that j is odd, we define $E(G_j) = K(C_{j,1}) \cup K(C_{j,2}) \cup K(C_{j,3}) \cup K(C_{j,4}) \cup K(C_{j,5})$ with $C_{j,1} = \{\widehat{c}\} \cup \{\widehat{x}_i \mid x_i \in X, i > \frac{j}{2}\}$, $C_{j,2} = \{c_1, c_2, \dots, c_m\}$, $C_{j,3} = \{x_i, \neg x_i \mid x_i \in X\} \cup \{\widehat{x}_{\frac{j-1}{2}}\}$, $C_{j,4} = \emptyset$, and $C_{j,5} = \{\widehat{x}_i \mid x_i \in X, i < \frac{j-1}{2}\}$.

Finally, for the last time step $L = 2n + 1$, we define $E(G_L) = K(C_{L,1}) \cup K(C_{L,2}) \cup K(C_{L,3}) \cup K(C_{L,4}) \cup K(C_{L,5})$ with $C_{L,1} = \emptyset$, $C_{L,2} = \{c_1, c_2, \dots, c_m\}$, $C_{L,3} = \{x_i, \neg x_i \mid x_i \in X\} \cup \{\widehat{c}\}$, $C_{L,4} = \emptyset$, and $C_{L,5} = \{\widehat{x}_i \mid x_i \in X\}$.

Let us now analyze how a potential exploration schedule for \mathcal{G} can look like. Recall that for each time step t , the first connected component $C_{t,1}$ is monotonously shrinking, i.e., for $t, t' \in [L]$ with $t \leq t'$ it holds that $C_{t,1} \supseteq C_{t',1}$. As the start vertex of the exploration schedule is contained in the third connected component, this means that (i) any exploration schedule cannot visit a first connected component as the connected component $C_{t,1}$ never has a non empty intersection with any connected component $C_{t-1,\ell}$ for $1 < t \leq L$, and $\ell \neq 1$.

Further, recall that the fifth connected component $C_{t,5}$ is monotonously growing, i.e., for $t, t' \in [L]$ with $t \leq t'$ it holds that $C_{t,1} \subseteq C_{t',1}$. Therefore, (ii) an exploration schedule cannot leave any fifth connected component.

Now consider the vertex \widehat{c} . For all time steps $t \in [L - 1]$, this vertex is contained in the first connected component $C_{t,1}$ and therefore not reachable from the start vertex. The only time step in which \widehat{c} is in another connected component is the last time step $t = 2n + 1$

in which \widehat{c} is contained in the third connected component $C_{t,3}$ together with the variable vertices. As an exploration schedule need to reach \widehat{c} and \widehat{c} is never contained in a fifth connected component, observation (ii) implies, that we must never enter a fifth connected component. As all elements in $\{\widehat{x}_i \mid x_i \in X\} \cup \{\widehat{x}\}$ do only appear in a first, third or fifth connected component, observation (i) implies that (iii) we need to visit them in a third connected component.

We further need to visit all elements in $\{c_1, c_2, \dots, c_m\}$. Assume, we visit some of them for the first time in some second connected component in a time step t . By the construction of \mathcal{G} , the second connected component is only reachable from a third or fourth connected component in an odd time step. Note that the odd time step t is the only time step in which the control vertex $\widehat{x_{\frac{t-1}{2}}}$ is contained in a third connected component. As by assumption the exploration schedule visits the second connected component in time step t it will not visit $\widehat{x_{\frac{t-1}{2}}}$ in time step t . But observation (iii) implies that then, $\widehat{x_{\frac{t-1}{2}}}$ cannot be reached during the remaining walk. Hence, any exploration schedule cannot visit any second connected component and the vertices $\{c_1, c_2, \dots, c_m\}$ must be visited in a third or fourth connected component in an even time step.

We already observed that any exploration schedule must be in the third component in any odd time step. From this component, it is only possible to visit some vertex c_j by traversing to a component containing a literal that is contained in c_j . In each even time step t , it is possible to visit those clause vertices c_j that contain a literal from the variable $x_{\frac{t}{2}}$. Thereby, we can either visit those containing the positive literal, or those containing the negative literal. An exploration schedule visiting all clause vertices $\{c_1, c_2, \dots, c_m\}$ thereby corresponds to a satisfying variable assignment for φ . Conversely, a satisfying variable assignment for φ gives us an exploration schedule for \mathcal{G} by traversing to the component containing the satisfied literal of the variable x_i in time step x_{2i} . ◀

We now shift our focus to restricting the graph class of the underlying graph. We show that NS-TEXP is NP-hard restricted to temporal graphs where the underlying graph is a tree consisting of two stars connected with an edge, or in other words, trees of diameter three. The vertex cover number of such trees is two. With a simple adaptation of the construction, we obtain that NS-TEXP is NP-hard even if every edge appears at most once, or is non-present in at most one time step and the underlying graph is a tree.

► **Theorem 6** (\star).¹ *NS-TEXP is NP-complete for temporal graphs even if:*

- *the underlying graph consists of two stars connected with a bridge, or*
- *every edge of the input temporal graph appears at most once, or*
- *the underlying graph is a tree and every edge is not present in at most one time step*

For the combined parameter $\gamma + L$, there exists a trivial FPT-algorithm for the problem k -ARB NS-TEXP (and so for NS-TEXP) as this parameter bounds the size of a search tree, where γ is the branching factor and L is the depth of the tree. In this tree we consider for each time step all connected components that are reachable from the current connected component, starting with the component containing the start vertex v .

► **Observation 7.** *k -ARB NS-TEXP can be solved in $\mathcal{O}(\gamma^L n^{\mathcal{O}(1)})$ time.*

¹ The proofs of the claims marked with (\star) are omitted in this extended abstract and are available in the full arXiv version [3].

In contrast, using a reduction from HITTING SET, we obtain that NS-TEXP does not have a polynomial kernel in the same parameter $\gamma + L$ unless $\text{NP} \subseteq \text{coNP}/\text{poly}$. In fact, we show a stronger statement: if γ is constant, there does not exist a polynomial kernel in L .

► **Theorem 8** (\star). *Unless $\text{NP} \subseteq \text{coNP}/\text{poly}$, there does not exist a polynomial kernel for NS-TEXP in the parameter L for any constant $\gamma \geq 6$.*

The size of an instance of NS-TEXP can be bounded by $n^2 \times L$ and there exists FPT algorithms in the parameter L as well as in the parameter n [20]. We already showed that under standard complexity assumptions, there does not exist a polynomial kernel for the parameter L . The next result is obtained by a cross-composition from NS-TEXP into itself.

► **Theorem 9** (\star). *Unless $\text{coNP} \subseteq \text{NP}/\text{poly}$, NS-TEXP does not admit a polynomial kernel parameterized by n .*

In [20] an FPT-algorithm based on color coding is given for the problem k -ARB NS-TEXP with parameter k . We obtain in the following that k -ARB NS-TEXP parameterized by L is $W[1]$ -hard by a reduction from multicolored independent set. We thereby answer an open question from [20]. This hardness contrasts the FPT-algorithm in parameter L for the problem NS-TEXP. We further obtain that k -ARB NS-TEXP does not admit a polynomial kernel in $L + k$ unless $\text{coNP} \subseteq \text{NP}/\text{poly}$. This result is based on a cross-composition from the problem k -ARB NS-TEXP to itself.

► **Theorem 10** (\star). *k -ARB NS-TEXP is $W[1]$ -hard parameterized by L .*

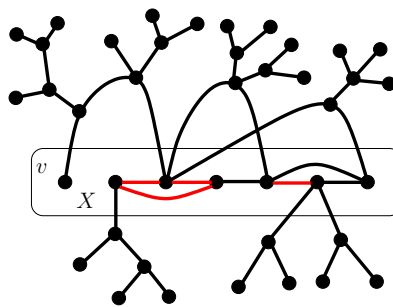
► **Theorem 11** (\star). *Unless $\text{coNP} \subseteq \text{NP}/\text{poly}$ k -ARB NS-TEXP does not admit a polynomial kernel parameterized by $k + L$.*

4 Polynomial Kernel for Bounded Number of Edge Appearances

In the field of static graphs, trees are nice structures that often lead to efficient algorithms. Building on this fact, structural parameters, like treewidth, defining a distance to trees were successfully introduced and used to design efficient algorithms. Unfortunately, we have seen in Theorem 6 that even when the underlying graph is a tree with vertex cover number 2 the problems we consider remain NP-hard. Therefore, in this section, we introduce a new structural parameter $p(\mathcal{G})$ which, intuitively, characterises how far \mathcal{G} is from being a tree in which each edge appears exactly once. Using this new parameter, we investigate the complexity of k -ARB NS-TEXP.

We restate the definition of $p(\mathcal{G})$. Recall that for a temporal graph $\mathcal{G} = (G_1, \dots, G_L)$, we write $\mathbf{m} = \sum_{i=1}^L |E(G_i)|$. Observe that if $\mathbf{m} < n - 1$, then the underlying graph G of \mathcal{G} is disconnected. Let X be the set of vertices of the connected component of G containing the source vertex v . Then, k -ARB NS-TEXP for \mathcal{G} is equivalent to k -ARB NS-TEXP for the temporal graph $\mathcal{G}[X] = (G_1[X], \dots, G_L[X])$. Hence, we can assume that the underlying graph is connected and $\mathbf{m} \geq n - 1$. This allows us to consider the above-guarantee parameter $p(\mathcal{G}) = \mathbf{m} - n + 1$. We show that k -ARB NS-TEXP admits a polynomial kernel in $p(\mathcal{G})$ when the underlying graph is connected. In fact, we give a kernel for a more general variant of the problem with vertex weights called WEIGHTED k -ARB NS-TEXP.

The starting point of our kernelization algorithm is a polynomial time algorithm for WEIGHTED k -ARB NS-TEXP on temporal graphs in which each edge appears exactly once and the underlying graph is a tree (Lemma 14). In addition to this algorithm, the crucial observation that leads to a kernel is that the underlying graph of the input temporal graph



■ **Figure 1** Structure of the underlying graph G of a temporal graph \mathcal{G} when the parameter $p = m - n + 1$ is bounded. Here $|X| \leq 4p$. (See Lemma 13.)

has a feedback edge set of bounded size (Lemma 12). By combining this observation and the fact that there is a bounded number of edges repetitions, we show that the input temporal graph \mathcal{G} has some specific structure. Namely, there exists a core set $X \subseteq V(\mathcal{G})$ of vertices of bounded size such that the underlying graph of the remaining graph $\mathcal{G} - X$ is a forest \mathcal{F} with the following property. In $\mathcal{G} - X$, each edge appears exactly once and each tree of \mathcal{F} is connected to X by at most two edges (see Figure 1 and Lemma 13). For each tree T in \mathcal{F} , depending on its structure and its interaction with X , we are able to describe all possible ways an exploration can visit some of the vertices of T . Using the polynomial time algorithm for trees in which edges appear only once, we can then compute the maximum weight contributed by the vertices in T for each of those cases and design a gadget of constant size that simulates the original tree T . Thereby, the gadget keeps the information of how many vertices, respectively vertex weights, were reachable in T and is the reason why we need to work with the weighted version of k -ARB-NS-TEXP. This gives us several reduction rules (Reduction Rule 1-4). To conclude the kernel, we show that after applying these reduction rules exhaustively, the obtained temporal graph has linear size in p (Claim 18). Lastly, by using an algorithm proposed by Frank and Tardos (Proposition 15), we can bound the size of the obtained weights on the vertices by $\mathcal{O}(p^4)$.

Throughout this section we assume that the temporal graphs under consideration have connected underlying graphs. Let $\mathcal{G} = (G_1, \dots, G_L)$ be a temporal graph and G its underlying graph. For $e \in E(G)$, we denote by $A(e) = \{t : e \in E(G_t) \text{ for } 1 \leq t \leq L\}$ and set $m(e) = |A(e)|$. Note that $m(\mathcal{G}) = \sum_{e \in E(G)} m(e)$.

By Observation 1, an instance (\mathcal{G}, w, v, k) of WEIGHTED k -ARB NS-TEXP is a yes-instance if and only if the underlying graph G has a monotone (v, x) -walk W for some $x \in V(G)$ such that $w(V(W)) \geq k$. Slightly abusing notation, we write $w(W)$ instead of $w(V(W))$ for the total weight of vertices visited by a walk.

We start by showing that the underlying graph has a feedback edge set of bounded size. Recall that a set of edges $S \subseteq E(G)$ is a *feedback edge set* of a graph G , if $G - S$ is a forest. Let $\mathcal{G} = (G_1, \dots, G_L)$ be a temporal graph and let $p = m - n + 1$. We denote by R the set of edges of the underlying graph G appearing at least twice in \mathcal{G} . We refer to the edges of R as *red*. We set $B = E(G) \setminus R$ and call the edges of B *blue*.

► **Lemma 12** (\star). *Let \mathcal{G} be a temporal graph and let $p = m - n + 1$. Then, the underlying graph G of \mathcal{G} has a feedback edge set S of size at most p such that every red edge is in S .*

Lemma 12 implies that G has a special structure that is used in our algorithm (see Figure 1).

► **Lemma 13.** *Let \mathcal{G} be a temporal graph with the underlying graph G and let $p = m - n + 1$. Let also $v \in V(G)$. Then, there is a set of vertices $X \subseteq V(G)$ of size at most $4p$ such that (i) $v \in X$, (ii) every red edge is in $G[X]$, (iii) $G - E(G[X])$ is a forest and (iv) each connected component T of $G - X$ is a tree with the properties that each vertex $x \in X$ has at most one neighbor in T and*

(a) *either X has a unique vertex x that has a neighbor in T ,*

(b) *or X contain exactly two vertices x and y having neighbors in T .*

Furthermore, if q is the number of connected components T of $G - X$ satisfying (b), then $q \leq 4p - 1$ and $G[X]$ has at most $5p - q - 1$ edges.

Proof. By Lemma 12, G has a feedback edge set S of size at most p containing all red edges. We define $Y = \{v\} \cup \{x \in V(G) \mid x \text{ is an endpoint of an edge of } S\}$. Note that $|Y| \leq 2p + 1$. Consider the graph G' obtained from G by the iterative deletion of vertices of degree one that are not in Y . Because S is a feedback edge set, we obtain that $H = G' - S$ is a forest such that every vertex of degree at most one is in Y . It is a folklore knowledge that any tree with $\ell \geq 2$ vertices of degree one (leaves) has at most $\ell - 2$ vertices of degree at least three. This implies that the set Z of vertices of H with degree at least three has size at most $2p - 1$. We define $X = Y \cup Z$. By the construction, $|X| \leq 4p$ and (i)–(iv) are fulfilled.

Let q be the number of connected components T of $G - X$ satisfying (b). Consider the multigraph G'' obtained from $G[X]$ by the following operation: for each connected component T of $G - X$ satisfying (b), where x and y are the vertices of X having neighbors in T , add the edge xy to $G[X]$ and make it a multi-edge if $xy \in E(G[X])$. Because of (iii), $G'' - E(G[X])$ is a tree on $|X|$ vertices, therefore, we have that $q \leq |X| - 1 \leq 4p - 1$. Because G has a feedback edge set of size at most p , G'' has the same property. Therefore, G'' has at most $|X| - 1 + p \leq 5p - 1$ edges. Hence, $G[X]$ has at most $5p - q - 1$ edges. ◀

We show that WEIGHTED k -ARB NS-TEXP can be solved in polynomial time if each edge appears exactly once and the underlying graph is a tree. If an edge e appears exactly once, then we use $t(e)$ to denote the unique element of $A(e)$.

► **Lemma 14.** *There is an algorithm running in $\mathcal{O}(n + L)$ time that, given a temporal graph $\mathcal{T} = (F_1, \dots, F_L)$ such that its underlying graph T is a tree with $m(e) = 1$ for each $e \in E(T)$, a weight function $w: V(T) \rightarrow \mathbb{N}_{>0}$ and two vertices $x, y \in V(T)$, either finds the maximum weight of a monotone (x, y) -walk W in T or reports that such a walk does not exist.*

Proof sketch. The algorithm works as follow. Consider the unique (x, y) -path P in T . Let $P = v_0, e_1, v_1, \dots, e_r, v_r$ in T with $x = v_0$ and $y = v_r$. As each edge e can only be crossed in the single time step $t(e)$, there exists a monotone (x, y) -walk W in T if and only if P is monotone. Suppose P is monotone. For $i \in [r]$, let $t_i = t(e_i)$, $t_0 = 0$ and $t_{r+1} = L$. Let $i \in \{0, \dots, r\}$. Because edges cannot be crossed in two different time steps, any (x, y) -walk needs to be on v_i between time step t_i and t_{i+1} . Therefore, for any time step $t_i \leq t \leq t_{i+1}$, an (x, y) -walk can only visit the vertices of $C_t(v_i)$, where $C_t(v_i)$ correspond to the connected components of F_t containing v_i , and the set $U(v_i) = \bigcup_{t_i \leq t \leq t_{i+1}} V(C_t(v_i))$ corresponds to all vertices that can be visited while waiting on v_i . Thereby, a (x, y) -walk of maximum weight visits exactly the vertices of $U = \bigcup_{i \in \{0, \dots, r\}} U(v_i)$. By the preconditions, the number of edges over all snapshots is $n - 1$. By using a BFS we can find the connected component containing a specific vertex in time linear in the size of the component. Therefore, we can construct U and compute its weight in time $\mathcal{O}(n + L)$. ◀

To reduce the vertex weights in our kernelization algorithm, we use the approach proposed by Etscheid et al. [21] that is based on the result of Frank and Tardos [23].

► **Proposition 15** ([23]). *There is an algorithm that, given a vector $\mathbf{w} \in \mathbb{Q}^r$ and an integer N , in polynomial time finds a vector $\bar{\mathbf{w}} \in \mathbb{Z}^r$ with $\|\bar{\mathbf{w}}\|_\infty \leq 2^{4r^3} N^{r(r+2)}$ such that $\text{sign}(\mathbf{w} \cdot b) = \text{sign}(\bar{\mathbf{w}} \cdot b)$ for all vectors $b \in \mathbb{Z}^r$ with $\|b\|_1 \leq N - 1$.*

Now we are ready to prove the main result of the section. We recall that a pair (P, Q) of subsets of $V(G)$ is called a *separation* of G if $P \cup Q = V(G)$ and there is no edge xy with $x \in P \setminus Q$ and $y \in Q \setminus P$. The set $P \cap Q$ is a *separator* and $|P \cap Q|$ is called the *order* of a separation (P, Q) . If a separation has order one, then the single vertex of $P \cap Q$ is called a *cut-vertex* and we say that this vertex is a separator. Notice that it may happen that $P \subseteq Q$ or $Q \subseteq P$ and we slightly abuse the standard notation, as in these cases necessarily $P \cap Q$ does not separate G . We also recall that an edge cut of G is a partition (P, Q) of $V(G)$ and the edge cut-set is the set of all edges xy for $x \in P$ and $y \in Q$.

► **Theorem 16.** *WEIGHTED k -ARB NS-TEXP parameterized by $p = \mathbf{m} - n + 1$ admits a kernel of size $\mathcal{O}(p^4)$ for connected underlying graphs such that for the output instance $(\mathcal{G} = (G_1, \dots, G_L), w, v, k)$, \mathcal{G} has $\mathcal{O}(p)$ vertices and edges, and $L \in \mathcal{O}(p)$.*

Proof sketch. Let $(\mathcal{G} = (G_1, \dots, G_L), w, v, k)$ be an instance of WEIGHTED k -ARB NS-TEXP and let $p = \mathbf{m} - n + 1$. Let also G be the underlying graph of \mathcal{G} and let R be the set of red edges. We describe our kernelization algorithm as a series of reduction rules that are applied exhaustively whenever it is possible to apply a rule. The rules modify $\mathcal{G} = (G_1, \dots, G_L)$ and w . Whenever we say that a rule deletes a vertex x , this means that x is deleted from G_1, \dots, G_L and G together with the incident edges. Similarly, whenever we create a vertex, this vertex is added to every graph G_i for $i \in [L]$ and G . When we either delete or add an edge, we specify G_i where this operation is performed and also assume that the corresponding operation is done for G .

To describe the first rule, we need some auxiliary notation. Let (P, Q) be a separation of G of order one with a cut-vertex x . We say that (P, Q) is *important* if

- (i) $G[P]$ is a tree with at least two vertices,
- (ii) the start vertex v is in Q and $R \subseteq E(G[Q])$, and
- (iii) P is an inclusion-maximal set satisfying (i) and (ii).

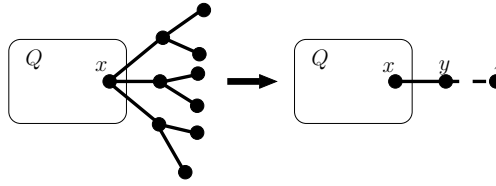
Suppose that (P, Q) is an important separation of G with a cut-vertex x . Let $a, b \in [L]$. Then, there is a separation (P_1, P_2) of $H = G[P]$ with x being the cut-vertex such that for any $y \in N_H(x)$, $y \in P_1$ if and only if $a \leq t(xy) \leq b$ and, moreover, (P_1, P_2) is unique. We use $\mathcal{T}_{[a,b]}$ to denote the temporal graph $(G_1[P_1], \dots, G_L[P_1])$ and $T_{[a,b]} = G[P_1]$. Observe that $\mathcal{T}_{[a,b]}$ may be a single-vertex temporal graph containing only x . We write $\mathcal{T}_{(a,b)}$ ($\mathcal{T}_{[a,b]}$ and $\mathcal{T}_{(a,b)}$, respectively) for $\mathcal{T}_{[a+1,b-1]}$ ($\mathcal{T}_{[a,b-1]}$ and $\mathcal{T}_{[a+1,b]}$, respectively) and we use the corresponding notation for the underlying trees. We also write \mathcal{T}_a and T_a instead of $\mathcal{T}_{[a,a]}$ and $T_{[a,a]}$. Given an important separation (P, Q) of G with a cut-vertex x , we denote $A(Q) = \bigcup_{y \in N_{G[Q]}(x)} A(xy)$. We say that $i, j \in A(Q)$ are *consecutive* if $i < j$ and there is no $t \in A(Q)$ such that $i < t < j$.

Observe that as cut-vertices and blocks of a graph can be found in linear time by standard algorithmic tools (see, e.g., [11]), all important separations can be listed in linear time.

Given a cut vertex x such that one side T of the important separation is a tree, then an optimal exploration can either enter and exit T by x or end in some vertex of T . In both cases, using the algorithm from Lemma 14 we can compute the optimal exploration of T and replace $T - x$ by 2 vertices simulating those two options giving us the first reduction rule.

► **Reduction Rule 1.** *If there is an important separation (P, Q) of G with a cut-vertex x such that either there is $i \in A(Q)$ such that $\mathcal{T} = \mathcal{T}_i$ has at least four vertices, or there are consecutive $i, j \in A(Q)$ such that $\mathcal{T} = \mathcal{T}_{(i,j)}$ has at least four vertices, or $\mathcal{T} = \mathcal{T}_{[1,i]}$ has at least four vertices for $i = \min A(Q)$, or $\mathcal{T} = \mathcal{T}_{[j,L]}$ has at least four vertices for $j = \max A(Q)$, then do the following for \mathcal{T} and the underlying tree T (see Figure 2):*

- Call the algorithm from Lemma 14 and compute the maximum weight w_1 of a monotone (x, x) -walk in T and the maximum weight w_2 of a monotone (x, y) -walk, where maximum is taken over all $y \in V(T)$. Set $t = \min_{u \in N_T(x)} t(xu)$.
- Delete the vertices of \mathcal{T} in \mathcal{G} except x , create a new vertex y , make it adjacent to x in G_t and set $w(y) = w_1 - w(x)$.
- If $w_2 > w_1$, then create a new vertex z , make it adjacent to y in G_L and set $w(z) = w_2 - w_1$.



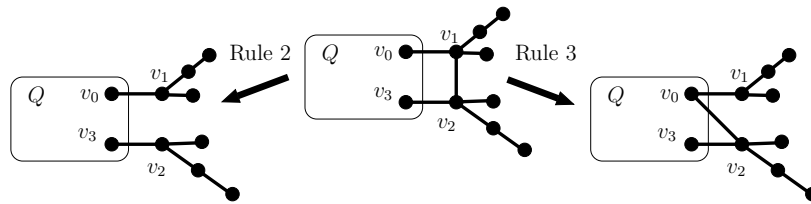
■ **Figure 2** The application of Reduction Rule 1.

To state the next rules, we define edge cuts that are important for our algorithm. We say that an edge cut (P, Q) of G is *important* if

- (i) the edge-cut set is of size two and consists of edges with distinct endpoints,
- (ii) $G[P]$ is a tree, and
- (iii) $v \in Q$ and $R \subseteq E(G[Q])$.

Given an important edge cut (P, Q) with an edge cut-set $\{x_1y_1, x_2y_2\}$, where $x_1, x_2 \in P$, there is the unique (x_1, x_2) -path S in the tree $G[P]$. We say that the path $y_1, y_1x_1, S, x_2y_2, y_2$ is the *backbone* of the edge cut. We also denote $\mathcal{T}_{P,Q} = (G_1[(P \cup \{y_1, y_2\})] - y_1y_2, \dots, G_L[P \cup \{y_1, y_2\}] - y_1y_2)$. Note that the underlying graph T of $\mathcal{T}_{P,Q}$ is a tree and the unique (y_1, y_2) -path in T is the backbone.

Observe that all important edge cuts can be found in polynomial time. The next three reduction rules reduce the length of backbones. We begin by deleting irrelevant edges.

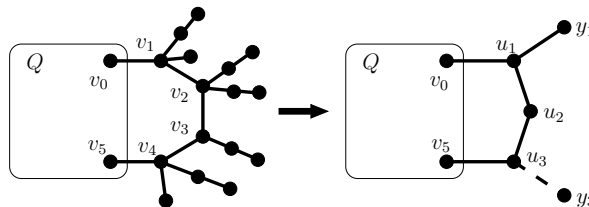


■ **Figure 3** The application of Reduction Rule 2 and Reduction Rule 3.

► **Reduction Rule 2.** *If there is an important edge cut (P, Q) of G with a backbone $v_0, e_1, v_1, e_2, v_2, e_3, v_3$ such that $t(e_1) > t(e_2)$ and $t(e_3) > t(e_2)$, then modify \mathcal{G} and G by deleting e_2 from $G_{t(e_2)}$ (see Figure 3). Furthermore, if the obtained underlying graph G is disconnected, then delete the vertices of the (unique) connected component that does not contain the source vertex v .*

The next rule aims at consecutive edges in backbones that occur in the same G_t .

► **Reduction Rule 3.** If there is an important edge cut (P, Q) of G with a backbone $v_0, e_1, v_1, e_2, v_2, e_3, v_3$ such that $t(e_1) = t(e_2)$, then modify \mathcal{G} and G by deleting e_2 from $G_{t(e_1)}$ and adding v_0v_2 in $G_{t(e_1)}$ (see Figure 3).



■ **Figure 4** The application of Reduction Rule 4.

It remains to shorten monotone backbones.

► **Reduction Rule 4.** If there is an important edge cut (P, Q) of G with a backbone $v_0, e_1, \dots, e_5, v_5$ such that $t(e_1) < \dots < t(e_5)$, then modify \mathcal{G} and G by doing the following for $\mathcal{T} = \mathcal{T}_{P, Q}$ with its underlying tree T (see Figure 4):

- Call the algorithm from Lemma 14 and compute the maximum weight w_1 of a monotone (v_0, v_0) -walk in T , the maximum weight w_2 of a monotone (v_0, v_5) -walk and the maximum weight w_3 of a monotone (v_5, v_5) -walk.
- Delete the vertices of $V(T) \setminus \{v_0, v_5\}$, create new vertices u_1, u_2, u_3 , make u_1 adjacent to v_0 in $G_{t(e_1)}$, make u_2 adjacent to u_1 and u_3 in $G_{t(e_2)}$ and $G_{t(e_4)}$, respectively, and make u_3 adjacent to v_5 in $G_{t(e_5)}$.
- Set $w(u_1) = w_1 - w(v_0)$, $w(u_3) = w_3 - w(v_5)$ and $w(u_2) = w_2 - w_1 - w_3$.
- Call the algorithm from Lemma 14 and compute the maximum weight w_0 of a monotone (v_0, u) -walk in $T - v_5$, where the maximum is taken over all $u \in V(T) \setminus \{v_5\}$. Create a vertex y_1 , make y_1 adjacent to u_1 in $G_{t(e_3)}$ and set $w(y_1) = w_0 - w(v_0) - w(u_1) - w(u_2)$.
- Call the algorithm from Lemma 14 and compute the maximum weight w_5 of a monotone (v_5, u) -walk in T , where the maximum is taken over all $u \in V(T)$. If $w_5 > w_3$, then create a vertex y_3 , make y_3 adjacent to u_3 in G_L and set $w(y_3) = w_5 - w(v_5) - w(u_3)$.

We observe that by the definitions of Rules 1–4, we immediately obtain the following claim.

▷ **Claim 17.** Rules 1–4 do not increase the parameter $p = \mathbf{m} - n + 1$.

After applying Rules 1–4, the graph has bounded size.

▷ **Claim 18.** If Rules 1–4 are not applicable, then $|V(\mathcal{G})| \leq 324p$ and $|E(\mathcal{G})| \leq 326p$.

The bound on the number of edges of G allows to reduce the value of L .

► **Reduction Rule 5.** If there is $t \in [L]$ such that G_t has no edge, then delete G_t from \mathcal{G} .

After applying this rule $L \leq 326p$ as $|E(\mathcal{G})| \leq 326p$.

Our last aim is to reduce the weights. For this, we use the algorithm from Proposition 15. Let $n = |V(G)|$ and let $V(G) = \{v_1, \dots, v_n\}$. We define $r = n + 1$ and consider the vector $\mathbf{w} = (w_0, w_1, \dots, w_n)^\top \in \mathbb{Z}^r$, where $w_0 = k$ and $w_i = w(v_i)$ for $i \in [n]$.

► **Reduction Rule 6.** Apply the algorithm from Proposition 15 for \mathbf{w} and $N = r + 1$ and find the vector $\bar{\mathbf{w}} = (\bar{w}_0, \dots, \bar{w}_n)$. Set $k := \bar{w}_0$ and set $w(v_i) := \bar{w}_i$ for $i \in [n]$.

To see that the rule is safe, let $k' = \bar{w}_0$ and let $w'(v_i) = \bar{w}_i$ for $i \in [n]$. Note that by the choice of N , for each vector $b \in \{-1, 0, 1\}^r$, we have that $\text{sign}(w \cdot b) = \text{sign}(\bar{w} \cdot b)$. This implies that the new weights $w'(x)$ and k' are positive integers and for every $U \subseteq V(G)$, $w(U) \geq k$ if and only if $w'(U) \geq k'$.

By Proposition 15, we obtain that $k \leq 2^{4n^3} (n+2)^{(n+1)(n+2)}$ and the same upper bound holds for $w(x)$ for every $x \in V(G)$. Because $|V(G)| = \mathcal{O}(p)$, we have that we need $\mathcal{O}(p^3)$ bits to encode k and the weight of each vertex. Then, the total bit-length of the encoding of the weights and k is $\mathcal{O}(p^4)$. Taking into account that $|V(G)| = \mathcal{O}(p)$, $|E(G)| = \mathcal{O}(p)$ and $L = \mathcal{O}(p)$, we conclude that we obtained a kernel of size $\mathcal{O}(p^4)$.

Because important separations and important edge cuts can be found in polynomial time and the algorithm from Lemma 14 is polynomial, we have that Rules 1–4 can be exhaustively applied in polynomial time. It is trivial that Reduction Rule 5 can be applied in polynomial time. Because the algorithm from Proposition 15 is polynomial, Reduction Rule 6 requires polynomial time. Therefore, the overall running time of our kernelization algorithm is polynomial. This concludes the proof. \blacktriangleleft

In [20], Erlebach and Spooner proved that NS-TEXP can be solved in $\mathcal{O}(2^n Ln^3)$ time. This fact together with Theorem 16 implies the following corollary.

► Corollary 19. *WEIGHTED k -ARB NS-TEXP parameterized by $p = m - n + 1$ can be solved in $2^{\mathcal{O}(p)}(nL)^{\mathcal{O}(1)}$ time on temporal graphs with connected underlying graphs.*

Conclusion

We initiated the study of polynomial kernels for exploration problems on temporal graphs. We showed that for the problems NS-TEXP, k -ARB NS-TEXP, and WEIGHTED k -ARB-NS-TEXP, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$, there does not exist a polynomial kernel for the parameters number of vertices n , lifetime L , and number of vertices to visit k ; and for the combined parameters $L+k$, $L+\gamma$, and $k+\gamma$, where γ is the maximal number of connected components per time step. In fact, by a straight forward reduction that repeats each snapshot sufficiently many times, all of our hardness results, that do not involve the parameter L , carry over to the strict settings TEXP and k -ARB TEXP where a temporal exploration traverses at most one edge per time step. We showed that the temporal exploration problems remain NP-hard restricted to temporal graphs where the underlying graph is a tree of diameter three. From a parameterized complexity point of view, this eliminates most of the common structural parameters considered on the underlying graph. Nonetheless, we were able to identify a new parameter of a temporal graph $p(\mathcal{G}) = \sum_{i=1}^L (|E(G_i)|) - |V(G)| + 1$ that captures how close the temporal graph is to a tree where each edge appears exactly once. Our parameter can also be seen as a notion of sparsity for temporal graphs. For this parameter $p(\mathcal{G})$, we were able to obtain a polynomial kernel for WEIGHTED k -ARB NS-TEXP. Using simplified reduction rules, we can obtain a kernel of linear size for NS-TEXP. While the reduction from NS-TEXP to TEXP blows up the parameter $p(\mathcal{G})$, our reduction rules can still be adapted to obtain polynomial kernels for the strict variants of the considered exploration problems. The natural next step would be to evaluate how useful our parameter is for other problems on temporal graphs.

References

- 1 Duncan Adamson, Vladimir V. Gusev, Dmitriy S. Malyshev, and Viktor Zamaraev. Faster exploration of some temporal graphs. In James Aspnes and Othon Michail, editors, *1st Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2022, March 28-30, 2022, Virtual Conference*, volume 221 of *LIPICs*, pages 5:1–5:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.SAND.2022.5.
- 2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.
- 3 Emmanuel Arrighi, Fedor V. Fomin, Petr A. Golovach, and Petra Wolf. Kernelizing temporal exploration problems. *CoRR*, abs/2302.10110, 2023. doi:10.48550/arXiv.2302.10110.
- 4 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.
- 5 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discret. Math.*, 28(1):277–305, 2014. doi:10.1137/120880240.
- 6 Hans L. Bodlaender and Tom C. van der Zanden. On exploring always-connected temporal graphs of small pathwidth. *Inf. Process. Lett.*, 142:68–71, 2019. doi:10.1016/j.ipl.2018.10.016.
- 7 Marin Bougeret and Ignasi Sau. How much does a treedepth modulator help to obtain polynomial kernels beyond sparse graphs? *Algorithmica*, 81(10):4043–4068, 2019. doi:10.1007/s00453-018-0468-8.
- 8 Benjamin Merlin Bumpus and Kitty Meeks. Edge exploration of temporal graphs. *Algorithmica*, pages 1–29, 2022.
- 9 Sobin C. C., Vaskar Raychoudhury, Gustavo Marfia, and Ankita Singla. A survey of routing and data dissemination in delay tolerant networks. *J. Netw. Comput. Appl.*, 67:128–146, 2016. doi:10.1016/j.jnca.2016.01.002.
- 10 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *Int. J. Parallel Emergent Distributed Syst.*, 27(5):387–408, 2012. doi:10.1080/17445760.2012.668546.
- 11 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.
- 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 Rodney G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer-Verlag, New York, 1999.
- 14 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *J. Comput. Syst. Sci.*, 119:1–18, 2021. doi:10.1016/j.jcss.2021.01.005.
- 15 Thomas Erlebach, Frank Kammer, Kelin Luo, Andrej Sajenko, and Jakob T. Spooner. Two moves per time step make a difference. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPICs*, pages 141:1–141:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ICALP.2019.141.
- 16 Thomas Erlebach and Jakob T. Spooner. Faster exploration of degree-bounded temporal graphs. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPICs*, pages 36:1–36:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.MFCS.2018.36.

- 17 Thomas Erlebach and Jakob T. Spooner. Non-strict temporal exploration. In Andrea Werneck Richa and Christian Scheideler, editors, *Structural Information and Communication Complexity - 27th International Colloquium, SIROCCO 2020, Paderborn, Germany, June 29 - July 1, 2020, Proceedings*, volume 12156 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2020. doi:10.1007/978-3-030-54921-3_8.
- 18 Thomas Erlebach and Jakob T. Spooner. Exploration of k-edge-deficient temporal graphs. *Acta Informatica*, 59(4):387–407, 2022. doi:10.1007/s00236-022-00421-5.
- 19 Thomas Erlebach and Jakob T. Spooner. Parameterized temporal exploration problems. In *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*, volume 221 of *LIPICs*, pages 15:1–15:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.SAND.2022.15.
- 20 Thomas Erlebach and Jakob T. Spooner. Parameterized temporal exploration problems. *CoRR*, abs/2212.01594, 2022. doi:10.48550/arXiv.2212.01594.
- 21 Michael Etscheid, Stefan Kratsch, Matthias Mnich, and Heiko Röglin. Polynomial kernels for weighted problems. *J. Comput. Syst. Sci.*, 84:1–10, 2017. doi:10.1016/j.jcss.2016.06.004.
- 22 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization. Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.
- 23 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Comb.*, 7(1):49–65, 1987. doi:10.1007/BF02579200.
- 24 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited - upper and lower bounds for a refined parameter. *Theory of Computing Systems*, 53(2):263–299, 2013. doi:10.1007/s00224-012-9393-4.
- 25 Bart M. P. Jansen, Jari J. H. de Kroon, and Michal Włodarczyk. Vertex deletion parameterized by elimination distance and even less. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1757–1769. ACM, 2021. doi:10.1145/3406325.3451068.
- 26 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.
- 27 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.
- 28 Pascal Kunz, Hendrik Molter, and Meirav Zehavi. In which graph structures can we efficiently find temporally disjoint paths and walks? *CoRR*, abs/2301.10503, 2023. doi:10.48550/arXiv.2301.10503.
- 29 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Math.*, 12(4):239–280, 2016. doi:10.1080/15427951.2016.1177801.
- 30 Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theor. Comput. Sci.*, 634:1–23, 2016. doi:10.1016/j.tcs.2016.04.006.
- 31 Polina Rozenshtein and Aristides Gionis. Mining temporal networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD*, pages 3225–3226. ACM, 2019. doi:10.1145/3292500.3332295.
- 32 Claude E Shannon. Presentation of a maze-solving machine. *Claude Elwood Shannon Collected Papers*, pages 681–687, 1993.
- 33 Shadi Taghian Alamouti. Exploring temporal cycles and grids. Master’s thesis, Concordia University, 2020.
- 34 Johannes Uhlmann and Mathias Weller. Two-layer planarization parameterized by feedback edge set. *Theor. Comput. Sci.*, 494:99–111, 2013. doi:10.1016/j.tcs.2013.01.029.

Cluster Editing with Overlapping Communities

Emmanuel Arrighi ✉ 🏠 

University of Trier, Germany

Matthias Bentert ✉

University of Bergen, Norway

Pål Grønås Drange¹ ✉ 

University of Bergen, Norway

Blair D. Sullivan ✉ 

University of Utah, Salt Lake City, UT, USA

Petra Wolf ✉ 🏠 

University of Bergen, Norway

Abstract

CLUSTER EDITING, also known as correlation clustering, is a well-studied graph modification problem. In this problem, one is given a graph and allowed to perform up to k edge additions and deletions to transform it into a cluster graph, i.e., a graph consisting of a disjoint union of cliques. However, in real-world networks, clusters are often overlapping. For example, in social networks, a person might belong to several communities – e.g. those corresponding to work, school, or neighborhood. Another strong motivation comes from language networks where trying to cluster words with similar usage can be confounded by homonyms, that is, words with multiple meanings like “bat”. The recently introduced operation of *vertex splitting* is one natural approach to incorporating such overlap into CLUSTER EDITING. First used in the context of graph drawing, this operation allows a vertex v to be replaced by two vertices whose combined neighborhood is the neighborhood of v (and thus v can belong to more than one cluster). The problem of transforming a graph into a cluster graph using at most k edge additions, edge deletions, or vertex splits is called CLUSTER EDITING WITH VERTEX SPLITTING and is known to admit a polynomial kernel with respect to k and an $O(9^{k^2} + n + m)$ -time (parameterized) algorithm. However, it was not known whether the problem is NP-hard, a question which was originally asked by Abu-Khazam et al. [Combinatorial Optimization, 2018]. We answer this in the affirmative. We further give an improved algorithm running in $O(2^{7k \log k} + n + m)$ time.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases graph modification, correlation clustering, vertex splitting, NP-hardness, parameterized algorithm

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.2

Funding *Matthias Bentert*: European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 819416).

Pål Grønås Drange: Research Council of Norway under grant Parameterized Complexity for Practical Computing (NFR, no. 274526).

Blair D. Sullivan: Gordon & Betty Moore Foundation under grant GBMF4560.

1 Introduction

Correlation clustering is a fundamental problem in data mining and machine learning that aims to identify groups of similar objects based on their pairwise similarity or dissimilarity. This problem arises in various domains, including social network analysis, image segmentation,

¹ corresponding author



document classification, gene expression analysis, and many more. Broadly speaking, the goal of correlation clustering is to partition the objects into clusters such that objects within the same cluster are more similar to each other than to objects in different clusters.

Correlation clustering has also been studied in algorithmic graph theory [34]. In this setting, the input is a graph and two vertices share an edge if they are similar. The goal is then to add or remove a minimum number of edges such that the resulting graph is a *cluster graph*, that is, a disjoint unions of cliques. This problem is typically known as CLUSTER EDITING and formally defined as follows.

CLUSTER EDITING

Input: A graph $G = (V, E)$ and an integer k .
Question: Does there exist a set $F \subseteq \binom{V}{2}$ of size at most k such that $G = (V, E \Delta F)$ is a cluster graph?

Here, Δ is the symmetric difference between two sets. Equivalently, one can ask whether there is a sequence $(\sigma_1, \sigma_2, \dots, \sigma_\ell)$ of length $\ell \leq k$ where each σ_i describes either an edge addition or an edge removal between two vertices such that performing all operations in the sequence yields a cluster graph.

CLUSTER EDITING has been extensively studied in terms of both classic and parameterized complexity. Cai [9] showed a general result implying an $O(3^k \cdot n^3)$ -time algorithm for CLUSTER EDITING. This running time has since been improved several times [23, 25, 29]. Moreover, it is known that CLUSTER EDITING admits a polynomial kernel with respect to k [21].

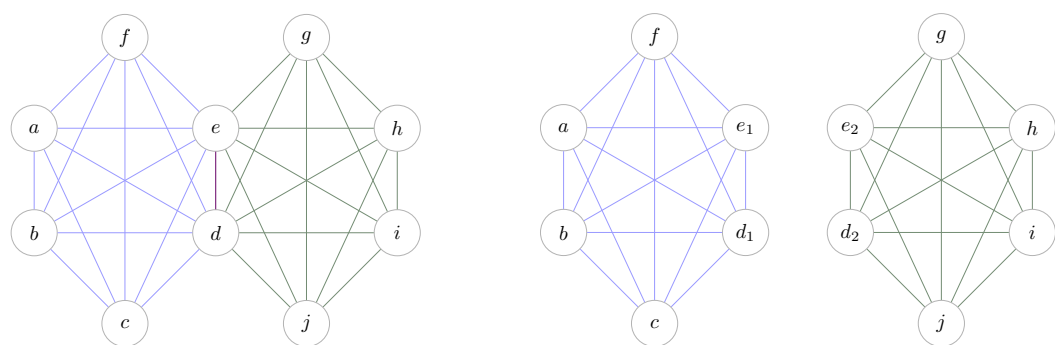
However, it has been observed that real-world data usually does not neatly conform to such an equivalence relation [27, 36, 40]. This led to the study of detecting so-called *overlapping communities* [5, 31], with applications in categorizing videos [35], classifying emotions in music [41] and sentiments in online posts [33], document classification [18, 32], and protein clustering based on amino-acid sequences [8].

The overlapping clusters model we study was introduced by Abu-Khzam et al. [2], who augmented CLUSTER EDITING with a *vertex-splitting* operation which enables vertices to belong to multiple clusters. Specifically, in addition to allowing edge modification, a vertex v can be *split* into two vertices v_1, v_2 such that $N(v_1) \cup N(v_2) = N(v)$, where N is the (open) neighborhood of a vertex (see Figure 1 for an example). In the same article [2], they show that this generalized problem admits a quadratic vertex kernel and an FPT algorithm. A slightly tighter analysis than the paper provides reveals that the algorithm runs in $O(9^{k^2} + n + m)$ time. Abu-Khzam et al. [1] later provided a greedy algorithm, but it was still not known whether the problem is actually NP-hard, a question posed by the former set of authors. We settle this question by showing that the following problem is indeed NP-complete.

CLUSTER EDITING WITH VERTEX SPLITTING (CEVS)

Input: A graph $G = (V, E)$ and an integer k .
Question: Does there exist a sequence $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_\ell)$ of length $\ell \leq k$, where each σ_i is an edge addition, an edge removal, or a vertex splitting, such that performing the operations in σ turns G into a cluster graph?

Related work. In addition to the aforementioned work [1, 2], overlapping clustering has been studied from the graph-editing point of view before, but under a less natural restriction. Fellows et al. [16] consider the problem where each vertex (and edge) is allowed to participate in a *fixed number* of clusters. In real-world applications, however, there is no fixed number s such that *every* vertex belongs to at most s cliques. Indeed, some vertices naturally belong to many clusters and some to few.



(a) A graph with two overlapping clusters that can be separated by two vertex splits.

(b) The graph after splitting d into d_1 and d_2 and e into e_1 and e_2 .

■ **Figure 1** An example of an instance where the budget needed for CLUSTER EDITING is 8, and where the resulting graph is quite different from the input graph (d has to be disconnected completely from (e.g.) g , h , i , and j). In CEVS, we only need a budget of 2, and the resulting graph is very similar to the input graph.

To the best of our knowledge, the first use of vertex splitting in graph modification is the PLANAR VERTEX SPLITTING used in graph drawing [15], a problem shown to be FPT quite recently by Nöllenburg et al. [30]. Even more recently, an FPT algorithm was given for obtaining a 2-layer planar drawing of a graph after at most k vertex splits [3]. In addition, vertex splitting has also been studied in the context of reducing a graph's pathwidth by Baumann, Pfretzschner, and Rutter [6], who also show that the problem Π -VERTEX SPLITTING is definable in MSO_2 (provided that Π is), making the problem FPT for graphs of bounded treewidth.

Finally, we would also like to highlight that finding ever faster algorithms for overlapping clustering and community detection is a major area of study in complex networks [4, 7, 11, 19, 20, 39, 43].

2 Preliminaries

For a positive integer n , we use $[n] = \{1, 2, \dots, n\}$ to denote the set of all positive integers up to n . All logarithms in this paper use 2 as their base.

We use standard graph-theoretic notation and refer the reader to the textbook by Diestel [12] for commonly used definitions. For an introduction to parameterized complexity, fixed-parameter tractability, and kernelization, we refer the reader to the textbooks by Flum and Grohe [17] and Cygan et al. [10]. The only non-standard graph concept used in this work are critical cliques as introduced by Lin et al. [28].

► **Definition 1.** A critical clique is a subset of vertices C that is maximal with the properties that

1. C is a clique
2. there exists $U \subseteq V(G)$ s.t. $N[v] = U$ for all $v \in C$.

► **Lemma 2** ([2, 28]). Every vertex appears in exactly one critical clique.

We denote the critical clique in which a vertex v appears by $CC(v)$. The critical clique quotient graph \mathcal{C} of G contains a vertex for each critical clique in G and two vertices are adjacent if and only if the two respective critical cliques C_1 and C_2 are adjacent, that is,

there is an edge between each vertex in C_1 and each vertex in C_2 . Note that by the definition of critical cliques this is equivalent to the condition that at least one edge $\{u, v\}$ with $u \in C_1$ and $v \in C_2$ exists. The main reason that critical cliques turn out to be useful when studying CEVS is captured by the following lemma.

► **Lemma 3** ([2]). *Let σ be an optimal solution to CEVS. Then for each critical clique C_i and each clique S_j in the cluster graph reached after performing the operations in σ , either $C_i \subseteq S_j$ or $C_i \cap S_j = \emptyset$.*

To show NP-hardness, we reduce from the well-known NP-complete problem 3-SAT.

3-SAT

Input: A CNF formula ϕ where each clause contains exactly three distinct literals.
Question: Is ϕ satisfiable?

Even stronger hardness results than NP-hardness can be achieved if one assumes the *exponential time hypothesis* (ETH) to be true. The ETH, formulated by Impagliazzo, Paturi, and Zane [24], states that there exists some positive real number s such that 3-SAT on N variables and M clauses cannot be solved in $2^{s(N+M)}$ time.

3 NP-hardness

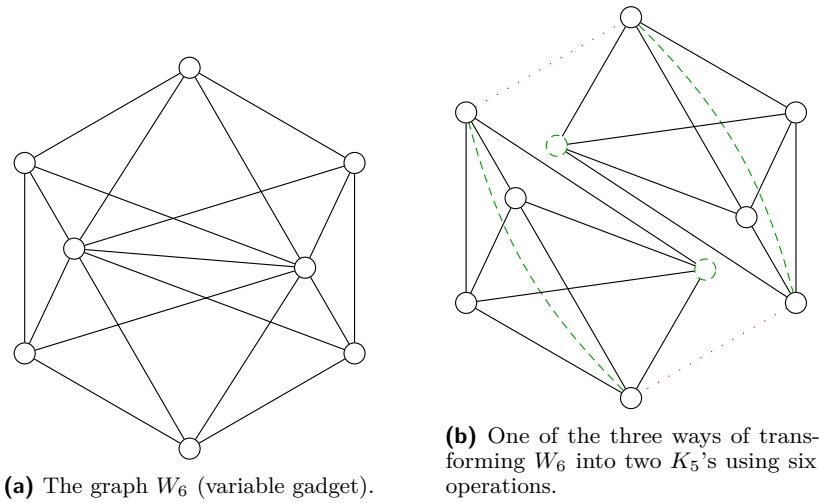
In this section, we show that CLUSTER EDITING WITH VERTEX SPLITTING is NP-hard, thereby resolving an open problem posed by Abu-Khzam et al. [2].

► **Theorem 4.** *CEVS is NP-complete. Moreover, assuming ETH, there is no $2^{o(n+m)}$ -time or $2^{o(k)} \cdot \text{poly}(n)$ -time algorithm for it.*

Proof. Since containment in NP is obvious (non-deterministically guess the sequence of operations and check that the resulting graph is indeed a cluster graph), we focus on the NP-hardness and present a reduction from 3-SAT. Therein, we will use two gadgets, a *variable gadget* and a *clause gadget*. The variable gadget is a wheel graph with two (connected) center vertices. An example of this graph is depicted on the left side of Figure 2. We call this graph with t vertices on the outside W_t and we will only consider instances with $t \bmod 6 = 0$, that is, $t = 6a$ for some positive integer a . The clause gadget is a “crown graph” as depicted in Figure 3(a).

More precisely, for each variable x_i , we construct a variable gadget G_i which is a W_{6a} where a is the number of clauses that contain either x_i or $\neg x_i$. For each clause C_j , we construct a clause gadget H_j , that is, a K_5 with the edges of a triangle removed. We arbitrarily assign each of the three vertices of degree two in H_j to one literal in C_j . Finally, we connect the variable and clause gadgets as follows. If a variable x_i appears in a clause C_j , then let u be the vertex in H_j assigned to x_i (or $\neg x_i$). Moreover, let b be the number such that C_j is the b^{th} clause containing either x_i or $\neg x_i$ and let $c = 6(b - 1)$. Let the vertices on the outer cycle of G_i be v_1, v_2, \dots, v_{6a} . If C_j contains the literal x_i , then we add the three edges $\{u, v_{c+1}\}, \{u, v_{c+2}\}, \{u, v_{c+3}\}$. If C_j contains the literal $\neg x_i$, then we add the three edges $\{u, v_{c+2}\}, \{u, v_{c+3}\}, \{u, v_{c+4}\}$. To complete the reduction, we set $k = 35M - 2N$, where M is the number of clauses and N is the number of variables.

We next show that the reduction is correct, that is, the constructed instance of CEVS is a yes-instance if and only if the original formula ϕ of 3-SAT is satisfiable. To this end, first assume that ϕ is satisfiable and let β be a satisfying assignment. For each variable x_i we will

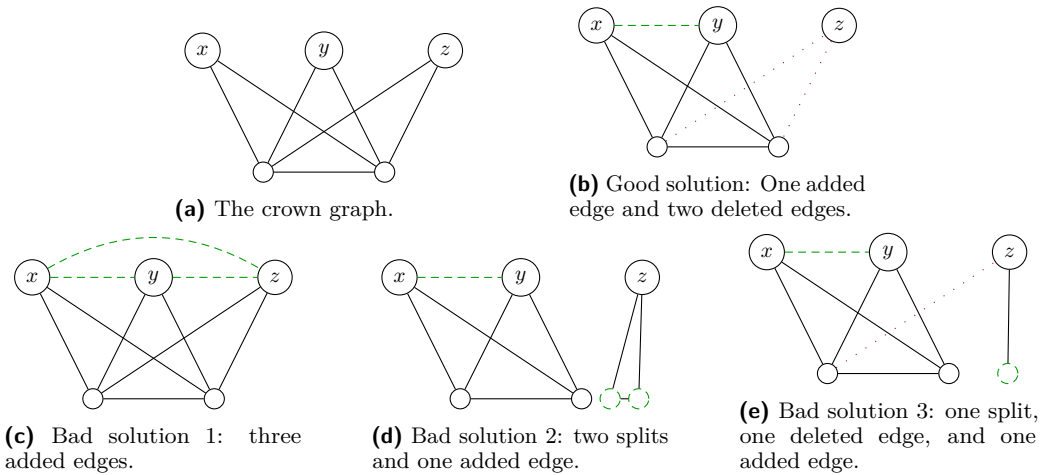


■ **Figure 2** The graph W_{6t} requires $8t - 2$ edits and any solution with exactly $8t - 2$ edits results in a disjoint union of K_5 s.

partition G_i into K_5 's as follows. Let a be the value such that G_i is isomorphic to W_{6a} . If β sets x_i to true, then we remove the edge $\{v_{3j}, v_{3j+1}\}$ and add the edge $\{v_{3j+1}, v_{3j+3}\}$ for each integer $1 \leq j \leq 2a$ (where values larger than $6a$ are taken modulo $6a$). If β sets x_i to false, then we remove the edge $\{v_{3j+1}, v_{3j+2}\}$ and add the edge $\{v_{3j+2}, v_{3j+4}\}$ for each $1 \leq j \leq 2a$. Moreover, we split the two center vertices $2a - 1$ times. In total, we use $8a - 2$ modifications to transform G_i into a collection of K_5 's. Since each clause contains exactly three literals and we add six vertices for each variable appearance, the sum of lengths of cycles in all variable gadgets combined is $18M$. Hence, in all variable gadgets combined, we perform $24M - 2N$ modifications.

Next, we modify the crown graphs. To this end, let C_j be a clause and let H_j be the constructed clause gadget. Since β is a satisfying assignment, at least one variable appearing in C_j satisfies it. If multiple such variables exist, then we pick any one. Let x_i be the selected variable and let u be the vertex in H_j assigned to x_i . We first turn H_j into a K_4 and an isolated vertex by removing the two edges incident to u in H_j and add the missing edge between the two vertices assigned to different variables. Finally, we look at the edges between variable gadgets and clause gadgets. For the vertex u , note that by construction the three vertices that u is adjacent to in G_i already belong to a K_5 and hence we can add two edges to the two (split) centers of the variable gadget to get to a K_6 . For the two other vertices in H_j that have edges to vertices in variable gadgets, we remove all three such edges, that is, six edges per clause. Hence, we use $3 + 2 + 6 = 11$ modifications for each clause. Since the total number of modifications is $35M - 2N$ and the resulting graph is a collection of K_4 's, K_5 's, and K_6 's, the constructed instance of CEVS is a yes-instance.

For the other direction, suppose the constructed instance of CEVS is a yes-instance. We first show that $24M - 2N$ modifications are necessary to transform all variable gadgets into cluster graphs and that this bound can only be achieved if each time exactly three consecutive vertices on the cycles are contained in the same K_5 . To this end, consider any variable gadget G_i . By construction, G_i is isomorphic to W_{6a} for some integer a . By the counting argument from above, we show that at least $8a - 2$ modifications are necessary. Note first that some edge in the cycle has to be removed or some vertex on the cycle has to be split as otherwise any solution would contain a clique with all vertices in the cycle and



■ **Figure 3** The crown graph with its four solutions of size 3. The *good solution* is the only solution with three operations that creates at least one isolated vertex.

this would require at least $18a^2 - 9a > 8a - 2$ edge additions (since the degree of each of the $6a$ vertices in the cycle would need to increase from 2 to $6a - 1$). We next analyze how many modifications are necessary to separate b vertices from the outer cycle into a clique. We require at least two modifications for the center vertices (either splitting them or removing the edges between them and the first vertex that we want to separate) and one operation to separate the cycle on the other end (either splitting a vertex or removing an edge of the cycle). For $b \in \{1, 2\}$ these operations are enough. For $b \geq 3$, we need to add $\binom{b}{2} - (b - 1)$ edges (all edges in a clique of size b minus the already existing edges of a path on b vertices). Note that the “average cost” per separated vertex (number of operations divided by b) is minimized (only) with $b = 3$ with a cost of 4 for three vertices. Hence, to separate all but c vertices from the cycle, we require at least $4(6a - c)/3$ operations. The cost for making the remaining c vertices into a clique requires again $\binom{c}{2} - (c - 1)$ edge additions. Analogously, the optimal solution is to have $c = 3$ with just a single edge addition. Thus, the minimum number of required operations is at least $4(6a - 3)/3 + 2 = 8a - 2$ (where the $+2$ comes from the initial edge removal and the final edge addition between the last $c = 3$ vertices) and this value can only be reached by partitioning the cycle into triples which each form a K_5 with the two center vertices. Note that it is always preferable to delete an edge on the outer cycle and not split one of the two incident edges as splitting a vertex increases the number of vertices on the cycle and thus invokes a higher average cost. Next, we analyze the clause gadget and the edges between the different gadgets. We start with the latter. Let u be a vertex in a clause gadget H_j with (three) incident edges to some variable gadget. The only way to not use at least three operations to deal with the three edges is if u is an isolated vertex or if the three neighbors do not have two more neighbors in the current solution. In the former case, we can (possibly) add the two edges between u and the two centers of the respective variable gadgets to build a K_6 . In the latter case, we have used at least three operations more in the variable gadget than intended (either by removing edges between neighbors of u and the center vertices or by splitting all neighbors of u). Since each vertex in a variable gadget is only adjacent to at most one vertex in a clause gadget, this cannot lead to an overall reduction in the number of operations and we can therefore ignore this latter case.

We are now in a position to argue that at least eleven modifications are necessary for each clause gadget. First, note that at least three operations are required to transform a crown into a cluster graph. Possible ways of achieving this are depicted in Figure 3. In each of these possibilities at most one vertex becomes an isolated vertex. To make two vertices independent, at least four operations are required and for three isolated vertices at least five operations are required. As shown above, at least two operations are required for each isolated vertex with edges to variable gadgets and at least three operations are required for non-isolated vertices with edges to variable gadgets. Thus, at least eleven operations are required for each clause gadget and eleven operations are sufficient if and only if the three vertices incident to one of the vertices in H_j belong to the same K_5 in the variable gadget.

By the argument above, at least $24M - 2N + 11M = k$ operations are necessary and since the constructed instance is a yes-instance, there is a way to cover all variable gadgets with K_5 's such that for each clause there is at least one vertex whose three neighbors in a variable gadget belong to the same K_5 . Let C_j be a clause, let u be a vertex with all three neighbors in the same K_5 , and let x_i be the variable corresponding to this variable gadget. If x_i appears positively in C_j , then v_{3i+1}, v_{3i+2} , and v_{3i+3} belong to the same K_5 for each i and we set x_i to true. If x_i appears negatively in C_j , then v_{3i+2}, v_{3i+3} , and v_{3i+4} belong to the same K_5 for each i and we set x_i to false. Note that we never set a variable to both true and false in this way. We set all remaining variables arbitrarily to true or false. By construction, the variable x_i satisfies C_j and since we do the same for all clauses, all clauses are satisfied, that is, the original formula ϕ is satisfiable. Thus, the constructed instance is equivalent to the original 3-SAT instance.

Since the reduction can clearly be computed in polynomial time, this concludes the proof for the NP-hardness. For the ETH-hardness, observe that $k, n, m \in O(N + M)$. This implies that there are no $2^{o(n+m)}$ -time or $2^{o(k)} \cdot \text{poly } n$ -time algorithm for CEVS unless the ETH fails [24]. ◀

In contrast to the reduction for CLUSTER EDITING [26], our reduction does not produce instances with constant maximum degree. We instead observe that in our reduction, the maximum degree of the produced instances depends only on the maximum number of times a variable appears in a clause. Combining this with the fact that 3-SAT remains NP-hard when restricted to instances where each variable appears in at most four clauses [37], we obtain the following corollary.

► **Corollary 5.** *CEVS remains NP-hard on bounded-degree graphs.*

4 A faster algorithm

In this section, we improve upon the known $O(9^{k^2} + n + m)$ -time algorithm and present an algorithm running in $O(2^{7k \log k} + n + m)$ time. The general outline of the two algorithms is fairly similar. We first compute a kernel with $O(k^2)$ vertices as well as all critical cliques in linear time [2]. We then guess² which critical cliques belong to the same clique (cluster) in the solution. The main difference between the two algorithms is how these guesses are made.

► **Theorem 6.** *CEVS can be solved in $O(2^{7k \log k} + n + m)$ time.*

² Whenever we pretend to guess something, we iterate over all possibilities and assume in the presentation that we are currently in an iteration that yields a solution.

Proof. First, we compute the critical clique of each vertex and the critical clique quotient graph \mathcal{C} of G in linear time [28]. As shown by Abu-Khzam et al. [2], we may assume that \mathcal{C} contains at most $4k$ vertices. By Lemma 3, we can also assume that all vertices in a critical clique belong to the same clique in the graph G' reached after performing an optimal solution σ . Let $\mathcal{X} = (S_1, S_2, \dots, S_\ell)$ be the set of cliques in G' . Note that \mathcal{X} contains $\ell \leq 2k$ cliques as each operation can complete at most two cliques of the solution (removing an edge between two cliques or splitting a vertex contained in both cliques). Hence, if there are more than $2k$ cliques in the solution, then we cannot reach the solution with k operations. To streamline the following argumentation, we will cover the vertices in \mathcal{C} by cliques S_1, S_2, \dots, S_ℓ and assume that an optimal solution contains exactly $2k$ cliques by allowing some of the cliques to be empty. Next, we iterate over all possible colorings of the vertices in \mathcal{C} using $\ell + 1$ colors $0, 1, 2, \dots, \ell$. Note that there are at most $(\ell + 1)^{4k} \in O((2k + 1)^{4k})$ such colorings.

The idea behind the coloring is the following. All colors $1, 2, \dots, \ell$ will correspond to the cliques S_1, S_2, \dots, S_ℓ , that is, we try to find a solution where all (critical cliques corresponding to) vertices of the same color (except for color 0) belong to the same clique in the solution. The color 0 indicates that the vertex will belong to multiple cliques in the solution, that is, that all vertices in the respective critical clique will be split. Since each such split operation reduces k by one, we can reject any coloring in which the number of vertices in critical cliques corresponding to vertices with color 0 is more than k . In particular, we can reject any coloring in which more than k vertices have color 0.

Next, we guess two indices $i \in [k], j \in [\ell]$ and assume that the i^{th} vertex of color 0 belongs to S_j or we guess that all vertices of color 0 have been assigned to all cliques they belong to. Note that in each iteration there are $k\ell + 1$ possibilities and since each guess (except for the last one) reduces k by at least one, we can make at most k guesses (after k guesses we know that the next guess has to be that all vertices have been fully assigned). Hence, there are at most $(k\ell + 1)^k = (2k^2 + 1)^k \in O((2k + 1)^{2k})$ such guesses.

It remains to compute the best solution corresponding to each possible sequence of guesses. To this end, we first iterate over each pair of vertices and remove an existing edge between them if we guessed that the two vertices do not appear in a common clique. Moreover, we add an edge between them if such an edge does not already exist and we guessed that there is a clique S_i which contains both vertices. Finally, we perform all split operations. Therein, we iteratively split one vertex v into two vertices u_1 and u_2 where u_1 will be the vertex in some clique S_i and u_2 might be split further in the future. The vertex u_1 is adjacent to all vertices that are guessed to belong to S_i . The vertex u_2 is adjacent to all vertices that u was adjacent to, except for vertices that are adjacent to u_1 and not guessed to also belong to some other clique S_j which (some part of) u_2 belongs to.

Since our algorithm basically performs an exhaustive search, it will find an optimal solution. It only remains to analyze the running time. We first compute the kernel in $O(n + m)$ time. We then try $O((2k + 1)^{4k})$ possible colorings of \mathcal{C} and for each coloring $O((2k + 1)^{2k})$ sequences of guesses. Afterwards, we compute the solution in $O(k^3)$ time. Thus, the overall running time is in $O((2k + 1)^{6k} \cdot k^3 + n + m) \subseteq O(2^{7k \log k} + n + m)$. ◀

We mention in passing that while the constants in the running time of our algorithm can probably be improved, a completely new approach is required if one wants a single-exponential-time algorithm. This is due to the fact that the number of possible partitions of $O(k)$ critical cliques into clusters grows super-exponentially (roughly as fast as $k!$) even if no vertex-splitting operations are allowed.

5 Conclusion

On the one hand, we resolve an open question from the literature by showing that CEVS is NP-complete. We also show that, assuming the ETH, there are no $2^{o(n+m)}$ -time or $2^{o(k)} \cdot \text{poly}(n)$ -time algorithms for CEVS. On the other hand, we give an $O(2^{7k \log k} + n + m)$ -time algorithm, beating the previously best $O(9^{k^2} + n + m)$ algorithm. This leaves the following gap.

► **Open problem.** *Does there exist a $2^{O(k)} \cdot \text{poly}(n)$ -time algorithm for CEVS?*

However, even resolving this question should only be seen as a starting point for a much larger undertaking. While we do understand the parameterized complexity of CEVS with respect to k reasonably well, there are still a lot of open questions regarding structural parameters of the input graph. Moreover, one might also study the approximability of CEVS as the trivial constant-factor approximation of CLUSTER EDITING does not carry over if we allow vertex splitting. In case CEVS turns out to be hard to approximate, one might then continue with studying FPT-approximation (schemes) and approximation kernels (also known as lossy kernels).

Regarding real-world applications, it has been observed that requiring communities to be cliques is too restrictive in some settings. To circumvent this issue, many relaxations of cliques such as s -cliques, s -clubs, s -plexes, k -cores, and γ -quasi-cliques have been proposed. It would also be interesting to study vertex-splitting operations in the respective graph editing problems for these relaxations.

Finally, a related area of study is clustering of bipartite data, which is modelled by BICLUSTER EDITING and which has received significant attention recently [13, 22, 38, 42]. Overlapping structures are also relevant in the bipartite case [14]. To the best of our knowledge, nothing is known about BICLUSTER EDITING WITH VERTEX SPLITTING. We mention that there are two natural versions in the bipartite case and both of them seem worth studying. The two versions differ in whether or not one requires that all copies of a split vertex lie on the same side of a bipartition in a solution. On the one hand, the additional requirement makes sense if the data is inherently bipartite. This happens for example if each vertex represents either a researcher or a paper. On the other hand, if edges reflect something like a seller-buyer relationship, then it is plausible that a person both sells and buys.

References

- 1 Faisal N. Abu-Khzam, Joseph R. Barr, Amin Fakhhereldine, and Peter Shaw. A greedy heuristic for cluster editing with vertex splitting. In *Proceedings of the 4th International Conference on Artificial Intelligence for Industries (AI4I '21)*, pages 38–41. IEEE, 2021.
- 2 Faisal N. Abu-Khzam, Judith Egan, Serge Gaspers, Alexis Shaw, and Peter Shaw. Cluster editing with vertex splitting. In *Combinatorial optimization*, pages 1–13. Springer, 2018.
- 3 Reyan Ahmed, Stephen Kobourov, and Myroslav Kryven. An FPT algorithm for bipartite vertex splitting. In *Proceedings of the 30th International Symposium on Graph Drawing and Network Visualization (GD '22)*, pages 261–268. Springer International Publishing, 2022.
- 4 Sanjeev Arora, Rong Ge, Sushant Sachdeva, and Grant Schoenebeck. Finding overlapping communities in social networks: toward a rigorous approach. In *Proceedings of the 13th ACM Conference on Electronic Commerce (EC '12)*, pages 37–54. Association for Computing Machinery, 2012.
- 5 Sanghamitra Bandyopadhyay, Garisha Chowdhary, and Debarka Sengupta. FOCS: Fast overlapped community search. *IEEE Transactions on Knowledge and Data Engineering*, 27(11):2974–2985, 2015.

- 6 Jakob Baumann, Matthias Pfretzschner, and Ignaz Rutter. Parameterized complexity of vertex splitting to pathwidth at most 1. *CoRR*, abs/2302.14725, 2023.
- 7 Jeffrey Baumes, Mark Goldberg, and Malik Magdon-Ismael. Efficient identification of overlapping communities. In *Proceedings of the 2005 IEEE International Conference on Intelligence and Security Informatics (ISI '05)*, pages 27–36. Springer, 2005.
- 8 Francesco Bonchi, Aristides Gionis, and Antti Ukkonen. Overlapping correlation clustering. *Knowledge and Information Systems*, 35(1):1–32, 2013.
- 9 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.
- 10 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.
- 11 George B. Davis and Kathleen M. Carley. Clearing the FOG: Fuzzy, overlapping groups for social networks. *Social Networks*, 30(3):201–212, 2008. doi:10.1016/j.socnet.2008.03.001.
- 12 Reinhard Diestel. *Graph Theory*. Springer, 2005.
- 13 Pål Grønås Drange, Felix Reidl, Fernando S. Villaamil, and Somnath Sikdar. Fast biclustering by dual parameterization. In *Proceedings of the 10th International Symposium on Parameterized and Exact Computation (IPEC '15)*, pages 402–413. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2015.
- 14 Nan Du, Bai Wang, Bin Wu, and Yi Wang. Overlapping community detection in bipartite networks. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT '08)*, pages 176–179, 2008.
- 15 Peter Eades and Candido Ferreira Xavier de Mendonça Neto. Vertex splitting and tension-free layout. In *Proceedings of the 3rd International Symposium on Graph Drawing and Network Visualization (GD '95)*, pages 202–211. Springer, 1995.
- 16 Michael R. Fellows, Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. Graph-based data clustering with overlaps. In *Proceedings of the 15th Annual International Conference on Computing and Combinatorics (COCOON '09)*, pages 516–526. Springer, 2009.
- 17 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.
- 18 Reynaldo Gil-García and Aurora Pons-Porrata. Dynamic hierarchical algorithms for document clustering. *Pattern Recognition Letters*, 31(6):469–477, 2010. doi:10.1016/j.patrec.2009.11.011.
- 19 Mark Goldberg, Stephen Kelley, Malik Magdon-Ismael, Konstantin Mertsalov, and Al Wallace. Finding overlapping communities in social networks. In *Proceedings of the 2nd IEEE International Conference on Social Computing (SC '10)*, pages 104–113, 2010. doi:10.1109/SocialCom.2010.24.
- 20 Steve Gregory. An algorithm to find overlapping community structure in networks. In *Proceedings of 2007 Knowledge Discovery in Databases (PKDD '07)*, pages 91–102. Springer, 2007.
- 21 Jiong Guo. A more effective linear kernelization for cluster editing. *Theoretical Computer Science*, 410(8-10):718–726, 2009. doi:10.1016/j.tcs.2008.10.021.
- 22 Jiong Guo, Falk Hüffner, Christian Komusiewicz, and Yong Zhang. Improved algorithms for bicluster editing. In *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation (TAMC '08)*, pages 445–456. Springer, 2008.
- 23 Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory of Computing Systems*, 47(1):196–217, 2010. doi:10.1007/s00224-008-9150-x.
- 24 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.

- 25 Christian Komusiewicz and Johannes Uhlmann. Alternative parameterizations for cluster editing. In *Proceedings of the 2011 Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '11)*, pages 344–355. Springer, 2011.
- 26 Christian Komusiewicz and Johannes Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012. doi:10.1016/j.dam.2012.05.019.
- 27 Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th International Conference on World Wide Web (WWW '08)*, pages 695–704. Association for Computing Machinery, 2008.
- 28 Guo-Hui Lin, Tao Jiang, and Paul E. Kearney. Phylogenetic k -root and Steiner k -root. In *Proceedings of the 11th International Symposium on Algorithms and Computation (ISAAC '00)*, pages 539–551. Springer, 2000.
- 29 Neeldhara Misra, Fahad Panolan, and Saket Saurabh. Subexponential algorithm for d -cluster edge deletion: Exception or rule? *Journal of Computer and System Sciences*, 113:150–162, 2020.
- 30 Martin Nöllenburg, Manuel Sorge, Soeren Terziadis, Anaïs Villedieu, Hsiang-Yun Wu, and Jules Wulms. Planarizing graphs and their drawings by vertex splitting. In *Proceedings of the 30th International Symposium on Graph Drawing and Network Visualization (GD '22)*, pages 232–246. Springer, 2022.
- 31 Lorenzo Orecchia, Konstantinos Ameranis, Charalampos Tsourakakis, and Kunal Talwar. Practical almost-linear-time approximation algorithms for hybrid and overlapping graph clustering. In *Proceedings of the 39th International Conference on Machine Learning (ICML '22)*, pages 17071–17093. PMLR, 2022.
- 32 Airel Pérez-Suárez, José Fco. Martínez-Trinidad, Jesús A. Carrasco-Ochoa, and José E. Medina-Pagola. An algorithm based on density and compactness for dynamic overlapping clustering. *Pattern Recognition*, 46(11):3040–3055, 2013. doi:10.1016/j.patcog.2013.03.022.
- 33 Hafiz Hassaan Saeed, Khurram Shahzad, and Faisal Kamiran. Overlapping toxic sentiment classification using deep neural architectures. In *Proceedings of the 2018 IEEE International Conference on Data Mining Workshops (ICDMW '18)*, pages 1361–1366. IEEE Computer Society, 2018. doi:10.1109/ICDMW.2018.00193.
- 34 Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007. doi:10.1016/j.cosrev.2007.05.001.
- 35 Cees G. M. Snoek, Marcel Worring, Jan C. van Gemert, Jan-Mark Geusebroek, and Arnold W. M. Smeulders. The challenge problem for automated detection of 101 semantic concepts in multimedia. In *Proceedings of the 14th ACM International Conference on Multimedia (MM '06)*, pages 421–430. Association for Computing Machinery, 2006.
- 36 Lei Tang and Huan Liu. Scalable learning of collective behavior based on sparse social dimensions. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM '09)*, pages 1107–1116. Association for Computing Machinery, 2009.
- 37 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.
- 38 Dekel Tsur. Faster parameterized algorithm for bicluster editing. *Information Processing Letters*, 168, 2021. doi:10.1016/j.ipl.2021.106095.
- 39 Qinna Wang and Eric Fleury. Uncovering overlapping community structure. In *Proceedings of the 2nd International Workshop on Complex Networks (COMPLEX '10)*, pages 176–186. Springer, 2010.
- 40 Xufei Wang, Lei Tang, Huiji Gao, and Huan Liu. Discovering overlapping groups in social media. In *Proceedings of the 2010 IEEE International Conference on Data Mining (ICDM '10)*, pages 569–578, 2010.

2:12 Cluster Editing with Overlapping Communities

- 41 Alicja Wieczorkowska, Piotr Synak, and Zbigniew W. Raś. Multi-label classification of emotions in music. In *Proceedings of the 2006 Intelligent Information Processing and Web Mining (IIPWM '04)*, pages 307–315. Springer, 2006.
- 42 Mingyu Xiao and Shaowei Kou. A simple and improved parameterized algorithm for bicluster editing. *Information Processing Letters*, 2022.
- 43 Shihua Zhang, Rui-Sheng Wang, and Xiang-Sun Zhang. Identification of overlapping community structure in complex networks using fuzzy c -means clustering. *Physica A: Statistical Mechanics and its Applications*, 374(1):483–490, 2007.



Existential Second-Order Logic over Graphs: Parameterized Complexity

Max Bannach  

European Space Agency, Advanced Concepts Team, Noordwijk, The Netherlands

Florian Chudigiewitsch  

Universität zu Lübeck, Germany

Till Tantau  

Universität zu Lübeck, Germany

Abstract

By Fagin’s Theorem, NP contains precisely those problems that can be described by formulas starting with an existential second-order quantifier, followed by only first-order quantifiers (ESO formulas). Subsequent research refined this result, culminating in powerful theorems that characterize for each possible sequence of first-order quantifiers how difficult the described problem can be. We transfer this line of inquiry to the *parameterized* setting, where the size of the set quantified by the second-order quantifier is the parameter. Many natural parameterized problems can be described in this way using simple sequences of first-order quantifiers: For the clique or vertex cover problems, two universal first-order quantifiers suffice (“for all pairs of vertices . . . must hold”); for the dominating set problem, a universal followed by an existential quantifier suffice (“for all vertices, there is a vertex such that . . .”); and so on. We present a complete characterization that states for each possible sequence of first-order quantifiers how high the parameterized complexity of the described problems can be. The uncovered dividing line between quantifier sequences that lead to tractable versus intractable problems is distinct from that known from the classical setting, and it depends on whether the parameter is a lower bound on, an upper bound on, or equal to the size of the quantified set.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Theory of computation → Complexity theory and logic; Theory of computation → Fixed parameter tractability; Theory of computation → W hierarchy

Keywords and phrases existential second-order logic, graph problems, parallel algorithms, fixed-parameter tractability, descriptive complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.3

Related Version *Full Version*: <https://arxiv.org/abs/2310.01134> [2]

Acknowledgements We thank Marcel Wienöbst for fruitful discussions and helpful comments on an earlier draft.

1 Introduction

The 3-coloring problem is to decide, given an undirected simple graph, whether there exist three sets R , G , and B (the red, green, and blue vertices) such that any two vertices x and y connected by an edge have different colors; or in logical terms:

$$\exists R \exists G \exists B \forall x \forall y \left((Rx \vee Gx \vee Bx) \wedge (x \sim y \rightarrow \neg((Rx \wedge Ry) \vee (Gx \wedge Gy) \vee (Bx \wedge By))) \right). \quad (1)$$

This formula is an *existential second-order* formula, meaning that it starts with existential second-order quantifiers ($\exists R \exists G \exists B$) followed by first-order quantifiers ($\forall x \forall y$) followed by a quantifier-free part. We can succinctly describe which quantifiers are used in such a prefix



© Max Bannach, Florian Chudigiewitsch, and Till Tantau;
licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 3; pp. 3:1–3:15

Leibniz International Proceedings in Informatics



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

by using “ E_1 ” for a (monadic, hence the “ $_1$ ”) existential second-order quantifier and “ e ” and “ a ” for existential and universal first-order quantifiers, respectively. The resulting *quantifier pattern* of the above formula is then $E_1E_1E_1aa$; and (monadic) existential second-order formulas are formulas with a prefix in $E_1^*(ae)^*$. It is no coincidence that an NP-complete problem can be described using the quantifier pattern $E_1E_1E_1aa$: Fagin’s Theorem [12] states that a problem lies in NP iff it can be described by a formula with a pattern in $E_i^*(ae)^*$ for some arity i . However, the example shows that the pattern $E_1E_1E_1aa$ already suffices to describe an NP-complete problem and a closer look reveals that so does E_1E_1aa . In contrast, formulas with the pattern E_iaa can only describe problems decidable in NL (regardless of the arity i of the quantified relation variables). Such observations have sparked an interest in different quantification patterns’ power. The question was answered by Gottlob, Kolaitis, and Schwentick [15] in the form of a dichotomy (“can only describe problems in P” versus “can describe an NP-complete problem”) and later in a refined form by Tantau [17], where the described problems in P are further classified into “in AC^0 ” or “L-complete” or “NL-complete”.

While in the formula for 3-colorability it was only necessary that three sets of colors *exist*, for many problems the *size* of these sets is important. Consider:

$$\phi_{\text{clique}} = \exists^{\geq} C \forall x \forall y ((Cx \wedge Cy) \rightarrow x \sim y), \quad (2)$$

$$\phi_{\text{vertex-cover}} = \exists^{\leq} C \forall x \forall y (x \sim y \rightarrow (Cx \vee Cy)), \quad (3)$$

$$\phi_{\text{dominating-set}} = \exists^{\leq} D \forall x \exists y (Dy \wedge (x = y \vee x \sim y)). \quad (4)$$

where the second-order quantifiers \exists^{\geq} and \exists^{\leq} ask whether there exists a set of size at least or at most some parameter value k such that the rest of the formula holds. These formulas show that we can describe the clique problem using a formula with the succinctly written pattern $E_1^{\geq}aa$ (and also $E_1^=aa$); the vertex cover problem using $E_1^{\leq}aa$ (and again also $E_1^=aa$); and the dominating set using $E_1^{\geq}ae$ (and yet again also $E_1^=ae$). Readers will notice that the problems are some of the most fundamental problems studied in that theory and lie in different levels of the W-hierarchy. The main message of the present paper is that it is once more *no coincidence that the quantifier patterns needed to describe these problems differ* ($E_1^{\geq}aa$ versus $E_1^{\leq}aa$ versus $E_1^{\geq}ae$): As done in [15, 17], we will give a complete characterization of the complexities of the problems that can be described using a specific quantifier pattern. The well-known results that the (parameterized) clique and dominating set problems are $W[1]$ -hard while the (parameterized) vertex cover problem lies in FPT (in fact, in para-AC^0) can now all be derived just from the syntactic structure of the formulas used to describe these problems.

Our Contributions. In this paper, we classify the complexity of the following classes (formal definitions are given in Section 2): Given a pattern $p \in \{a, e\}^*$ of first-order quantifiers, the classes $\text{p-FD}(E_1^=p)$, $\text{p-FD}(E_1^{\leq}p)$, and $\text{p-FD}(E_1^{\geq}p)$ contain all parameterized problems that can be described by formulas with quantifier pattern $E_1^=p$, or $E_1^{\geq}p$, or $E_1^{\leq}p$, respectively. The restriction to study just a *single, monadic, parameterized* ESO quantifier is motivated by our earlier observation that important and interesting problems of parameterized complexity can be described in this way. Our classification is complete in the sense that for every p we either show that all problems in the class are fixed-parameter tractable (in these cases, we derive more fine-grained results by placing the problems in para-AC^0 or $\text{para-AC}^{0\uparrow}$) or there is a $W[1]$ -hard problem that can be described using the pattern. Table 1 lists the obtained bounds. In the table, the classes with the subscript “basic” refer to the restriction to undirected graphs without self-loops. As can be seen, for these graphs we get slightly

■ **Table 1** Complete complexity classification of the weighted ESO logic for a single weighted monadic second-order quantification followed by first-order quantifiers with some pattern $p \in \{a, e\}^*$ (where $p \preceq q$ means that p is a subsequence of q). The upper part (arbitrary structures) and lower part (basic graphs) are identical except for the patterns $E_1^{\geq}ae$ and $E_1^{\leq}aa$, where they differ. Note that $\text{para-AC}^0 \subsetneq \text{para-AC}^{0\uparrow} \subseteq \text{para-P} = \text{FPT}$ and $\text{FPT} \cap \text{W}[1]\text{-hard} = \emptyset$ is a standard assumption.

$\text{p-FD}(E_1^-p)$	$\subseteq \text{para-AC}^0$, when $\cap \text{W}[1]\text{-hard} \neq \emptyset$, when	$p \preceq e^*a$. ae or $aa \preceq p$.
$\text{p-FD}(E_1^{\geq}p)$	$\subseteq \text{para-AC}^0$, when $\cap \text{W}[1]\text{-hard} \neq \emptyset$, when	$p \preceq e^*a$. ae or $aa \preceq p$.
$\text{p-FD}(E_1^{\leq}p)$	$\subseteq \text{para-AC}^0$, when $\not\subseteq \text{para-AC}^0$ but $\subseteq \text{para-AC}^{0\uparrow}$, when $\cap \text{W}[1]\text{-hard} \neq \emptyset$, when	$p \preceq e^*a$. $aa \preceq p \preceq e^*a^*$. $ae \preceq p$.
$\text{p-FD}_{\text{basic}}(E_1^-p)$	$\subseteq \text{para-AC}^0$, when $\cap \text{W}[1]\text{-hard} \neq \emptyset$, when	$p \preceq e^*a$. ae or $aa \preceq p$.
$\text{p-FD}_{\text{basic}}(E_1^{\geq}p)$	$\subseteq \text{para-AC}^0$, when $\cap \text{W}[1]\text{-hard} \neq \emptyset$, when	$p \preceq e^*a$ or ae . $aae, eae, \text{ or } aa \preceq p$.
$\text{p-FD}_{\text{basic}}(E_1^{\leq}p)$	$\subseteq \text{para-AC}^0$, when $\not\subseteq \text{para-AC}^0$ but $\subseteq \text{para-AC}^{0\uparrow}$, when $\cap \text{W}[1]\text{-hard} \neq \emptyset$, when	$p \preceq e^*a$ or aa . $aaa \preceq p \preceq e^*a^*$. $ae \preceq p$.

different complexity results. This is in keeping with the classical, non-parameterized setting studied by Gottlob et al. [15], where results for basic graphs are often *considerably* harder to obtain. However, the complexity landscape we uncover in the present paper is different from the one presented in [15] and [17]: Although certain patterns (like $p = ae$) feature prominently in the parameterized and non-parameterized analysis, the dividing lines are different. To establish these lines, we combine ideas used in the classical setting with different methods from parameterized complexity theory, tailored to the specific problems we study. The notoriously difficult cases from the classical setting (Gottlob et al. [15] spend 34 pages to address the case E_1^*ae , Tantau [17] spends several pages on E_1aa) are also technically highly challenging in the parameterized setting.

Our research sheds new light on what difference it makes whether we want solutions to have size *exactly* k or *at most* k or *at least* k . To begin, equations (2) and (4) already show that for individual problems (like clique) the maximization problem can be hard while minimization is trivial (a single vertex is always a clique) and for some problems (like dominating set) the opposite is true (the whole vertex set itself is always a dominating set). Furthermore, from the perspective of descriptive complexity, there is a qualitative difference between $\exists=C$ and $\exists^{\leq}C$ on the one hand and $\exists^{\geq}C$ on the other: For any k , the first two can easily be expressed in normal ESO logic using k first-order quantifiers binding the elements of C , while \exists^{\geq} translates to $\exists C \exists x_1 \cdots \exists x_k$ where the x_i bind the elements not in C . Thus, $\exists=$ and \exists^{\leq} only allow us to express problems that are “slicewise first-order” and hence in $\text{XAC}^0 \subseteq \text{XP}$, while already the slice for $k = 0$ of \exists^{\geq} formulas can express NP-complete problems for many patterns. *However*, we also prove a result for basic graphs for $p = ae$ that runs counter this “tendency” of \exists^{\geq} to be harder than \exists^{\leq} : While $\text{p-FD}_{\text{basic}}(E_1^{\leq}ae)$ contains the $\text{W}[2]\text{-hard}$ dominating set problem, $\text{p-FD}_{\text{basic}}(E_1^{\geq}ae) \subseteq \text{para-AC}^0$.

Related Work. Using logic to describe languages dates back all the way to Büchi’s pioneering work [6] on the expressive power of monadic second-order logic (which, over strings, describes exactly the regular languages). Switching from monadic second-order logic to existential second-order logic yields Fagin’s Theorem [12]. Since then, the expressive power of fragments of this logic was the subject of intensive research: Eiter et al. [11] studied the expressiveness of ESO-patterns over strings; Gottlob et al. did so over graphs [15]; Tantau [17] refined the latter results for subclasses of P. Taken together, these results give us a complete complexity-theoretic classification of the problems resulting from any ESO quantifier pattern over strings, basic graphs, directed graphs, undirected graphs, and arbitrary structures.

Using logical fragments to characterize complexity classes is also standard practice in parameterized complexity theory [13], especially the power of MSO logic plays a prominent role, see for instance [8]. In particular, characterizations of the levels of the W-hierarchy in terms of the number of quantifier alternations are known [9, 13], but – to the best of our knowledge – a complete and exact analysis of the parameterized complexity of problems in terms of the quantifier patterns describing them is new.

Organization of this Paper. Following a review of basic concepts and terminology in Section 2, we present our results on the power of quantifier patterns of the forms $E_1^{\leq} p$, $E_1^{\geq} p$, and $E_1^= p$ for $p \in \{a, e\}^*$ in the subsections of Section 3 (arbitrary structures) and Section 4 (basic graphs). Due to space constraints, we provide only proof ideas for some of the results. The full proofs can be found in the technical report version [2] of the text.

2 Background in Descriptive and Parameterized Complexity

Terminology for Graphs and Logic. A *directed graph* (“digraph”) is a pair $G = (V, E)$ where V is a *vertex set* and $E \subseteq V \times V$ an *edge set*. An *undirected graph* is a pair $G = (V, E)$ such that $E \subseteq \{\{u, v\} \mid u, v \in V\}$. A *basic graph* is an undirected graph that has no self-loops, that is, where all edges have size 2. In this paper, graphs are always finite.

We use standard terminology from logic and finite model theory, see for instance [10]. Let us point out some perhaps not-quite-standard notation choices: Our vocabularies τ (also known as *signatures*) contain *only relation symbols* and we write $\text{STRUC}[\tau]$ to denote the set of all finite τ -structures. For a first-order or second-order τ -sentence ϕ (a formula without free variables), let $\text{MODELS}(\phi)$ denote the subset of $\text{STRUC}[\tau]$ of all τ -structures that are models of ϕ . As an example, we can represent digraphs using the vocabulary $\tau_{\text{digraph}} = \{\sim^2\}$, containing a single binary relation symbol, and the class of digraphs is exactly $\text{STRUC}[\tau_{\text{digraphs}}]$. The formula $\phi = \forall x \forall y (x \sim y \rightarrow x \neq y)$ expresses that there are no loops in a graph, that is, $\text{MODELS}(\phi) = \{G \mid G \text{ is a digraph that has no self-loops}\}$. While an undirected graph $G = (V, E)$ is not immediately a τ_{digraph} -structure, we can trivially “turn it” into a structure \mathcal{G} by setting the universe to be V and setting $\sim^{\mathcal{G}} = \{(x, y) \mid \{x, y\} \in E\}$ and this structure is a model of $\phi_{\text{undirected}} = \forall x \forall y (x \sim y \rightarrow y \sim x)$. The structures representing basic graphs are then models of $\phi_{\text{basic}} = \forall x \forall y (x \sim y \rightarrow (x \neq y \wedge y \sim x))$. As another example, the class of all bipartite graphs equals $\text{MODELS}(\phi_{\text{bipartite}})$ where $\phi_{\text{bipartite}}$ is the second-order formula $\exists X \forall u \forall v (u \sim v \rightarrow (Xu \leftrightarrow \neg Xv))$ and Xu is our shorthand for the less concise $X(u)$.

As already sketched in the introduction, we can associate a *quantifier prefix pattern* (a word over the infinite alphabet $\{E_1, E_2, E_3, \dots\} \cup \{e, a\}$), or just a *pattern*, to formulas of ESO logic by first writing them in prenex normal form (quantifiers first, in a block) and

then replacing each (existential) second-order quantifier by E_i , where i is the arity of the quantifier, each universal first-order quantifier by a , and each existential first-order quantifier by e . For instance, the pattern of $\phi_{\text{bipartite}}$ is E_1aa .

Describing Problems and Classes. In the context of descriptive complexity a *decision problem* P is a subset of $\text{STRUC}[\tau]$ that is closed under isomorphisms. We say that ϕ *describes* P if $\text{MODELS}(\phi) = P$. Moving on to classes, for a set Φ of τ -formulas, let $\text{FD}(\Phi) := \{\text{MODELS}(\phi) \mid \phi \in \Phi\}$ denote the class of problems “Fagin-defined” by Φ . For a quantifier prefix pattern p let $\text{FD}(p) := \{\text{MODELS}(\phi) \mid \phi \text{ has pattern } p\}$, so (1) shows that $3\text{COLORABLE} \in \text{FD}(E_1E_1E_1aa)$, and for a set S of patterns let $\text{FD}(S) = \bigcup_{p \in S} \text{FD}(p)$. In slight abuse of notation, we usually write down regular expressions to denote sets S of quantifier patterns: For instance, Fagin’s Theorem [12] can now be written as “ $\text{NP} = \text{FD}(E_2^*(ae)^*$.” Trivially, more quantifiers potentially allow us to express more problems. Formally, let $p \preceq q$ denote that p is a subsequence of q (so p can be obtained from q by, possibly, deleting some letters). Then $\text{FD}(p) \subseteq \text{FD}(q)$. Also in slight abuse of notation, we also write things like “ $p \preceq e^*a$ ” to indicate that $p \preceq q$ holds for some q of the form e^*a .

Our analysis will show that restricting attention to basic graphs yields particularly interesting results. For this reason, it will be convenient to consider the introduced complexity classes restricted to basic graphs by adding a subscript “basic”: For τ_{digraph} -formulas ϕ , let $\text{MODELS}_{\text{basic}}(\phi) = \text{MODELS}(\phi) \cap \text{MODELS}(\phi_{\text{basic}})$ and define $\text{FD}_{\text{basic}}(\Phi)$ and $\text{FD}_{\text{basic}}(p)$ in the obvious ways – and similarly for the classes with the subscript “undirected.”

When we move from classical complexity theory to parameterized complexity, we assign to every *instance* a *parameter* that measures an aspect of interest of that instance and that is hopefully small for practical instances. A *parameterized problem* is a set $Q \subseteq \text{STRUC}[\tau] \times \mathbb{N}$ such that for every k the *slice* $\{x \mid (x, k) \in Q\}$ is closed under isomorphisms. In a pair (x, k) we call x the *input* and k the *parameter*. The usual goal in the field is to prove that a problem is *fixed-parameter tractable* (in FPT) by deciding $(x, k) \in Q$ in time $f(k) \cdot |x|^{O(1)}$ for some computable function f . In the context of problems described by ESO formulas, a natural parameter to consider is the *size of the relations that we can assign to the existential second-order quantifiers* and this size is commonly called the *solution weight*. As mentioned earlier, problems like the vertex cover problem can be described naturally in this manner: Consider the formula $\phi(X) = \forall u \forall v (u \sim v \rightarrow (Xu \vee Xv))$, where X is a free monadic second-order variable. Then for a graph $G = (V, E)$, viewed as a logical structure \mathcal{G} , and a set $C \subseteq V$ we have $\mathcal{G} \models \phi(C)$ iff C is a vertex cover of G . Thus, $(\mathcal{G}, k) \in \text{p-VERTEX-COVER} = \{(\mathcal{G}, k) \mid \mathcal{G} \text{ has a vertex cover of size } k\}$ iff there exists a set $C \subseteq V$ of size k such that $\mathcal{G} \models \phi(C)$.

Formally, the second-order quantifiers \exists^{\leq} , $\exists^=$, and \exists^{\geq} have the following semantics: For a structure \mathcal{S} with a universe U , a non-negative integer k , an i -ary second-order variable X , and a formula $\phi(X)$, we say that \mathcal{S} is a *model of* $\exists^{\leq} X \phi(X)$ for parameter k and write $(\mathcal{S}, k) \models \exists^{\leq} X \phi(X)$, if there is a set $C \subseteq U^i$ with $|C| \leq k$ such that $\mathcal{S} \models \phi(C)$. A formula starting with a \exists^{\leq} quantifier then gives rise to a parameterized problem: Let $\text{p-MODELS}(\exists^{\leq} X \phi(X)) = \{(\mathcal{S}, k) \mid (\mathcal{S}, k) \models \exists^{\leq} X \phi(X)\}$ and let $\text{p-FD}(\Phi) := \{\text{p-MODELS}(\phi) \mid \phi \in \Phi\}$. The at-least and equal cases are, of course, defined analogously. As an example, we have $\text{p-VERTEX-COVER} \in \text{p-FD}(E_1^{\leq} aa)$ since $\text{p-VERTEX-COVER} = \text{p-MODELS}(\exists^{\leq} X \forall u \forall v (u \sim v \rightarrow (Xu \vee Xv)))$.

Standard and Parameterized Complexity Classes. Concerning standard complexity classes, we use standard definitions, see for instance [1, 16]. In the context of descriptive complexity theory, it is often necessary to address coding issues (meaning the question of how words are encoded as logical structures and *vice versa*) – but fortunately this will not be important

for the present paper. Concerning parameterized complexity classes like $\text{FPT} = \text{para-P}$ or $\text{W}[1]$, we also use standard definitions, which can be adapted to the descriptive setting in exactly the same way as for classical complexity classes (see for instance [3, 4] for details) and encoding details will once more be unimportant. The classes para-AC^0 and $\text{para-AC}^{0\uparrow}$ are likely less well-known: We have $Q \in \text{para-AC}^0$ if there is a family $(C_{n,k})_{n,k \in \mathbb{N}}$ of unbounded fan-in circuits of constant depth and size $f(k) \cdot n^{O(1)}$ for some computable function f , such that for every $(\mathcal{S}, k) \in \text{STRUC}[\tau] \times \mathbb{N}$ we have $(\mathcal{S}, k) \in Q$ iff the circuit $C_{\text{length}(\mathcal{S}), k}$ outputs 1 on input of (a suitably encoded) \mathcal{S} , where $\text{length}(\mathcal{S})$ is the length of the encoding of \mathcal{S} . For the class $\text{para-AC}^{0\uparrow}$, the circuits may have depth $f(k)$. We have $\text{para-AC}^0 \subsetneq \text{para-AC}^{0\uparrow} \subseteq \text{para-P} = \text{FPT}$ [3]. In our proofs, two properties of the classes will be important: First, all of them are (quite trivially) closed under para-AC^0 -reductions. Second, for $\tau = (I^1)$, the signature with a single unary relation symbol, we have $\text{p-THRESHOLD} = \{(\mathcal{S}, k) \mid \mathcal{S} = (U, I^{\mathcal{S}}), |I^{\mathcal{S}}| \geq k\} \in \text{para-AC}^0$, that is, we can “count up to the parameter in para-AC^0 .” For more details on these classes, including discussions of uniformity, see [3, 4, 7].

3 Classifying Parameterized ESO Classes: Arbitrary Structures

We now begin tracing the tractability frontier for the classes from the upper part of Table 1: $\text{p-FD}(E_1^=p)$, $\text{p-FD}(E_1^{\geq}p)$, and $\text{p-FD}(E_1^{\leq}p)$. Recall that for these classes we are given a formula ϕ starting with one of the monadic second-order quantifiers $\exists^=$, \exists^{\leq} , or \exists^{\geq} , followed by first-order quantifiers with the pattern p ; and the objective is to show *upper bounds* of the form “for all ϕ with pattern p all $\text{p-MODELS}(\phi)$ lie in a certain class” and *lower bounds* of the form “there is a ϕ with pattern p such that $\text{p-MODELS}(\phi)$ contains a problem that is hard for a certain class”. We dedicate one subsection to each of $\exists^=$, \exists^{\leq} , and \exists^{\geq} , each starting with the main theorem and covering more technical parts of the proofs later.

In this section, we allow arbitrary (finite, relational) structures, meaning that the signature τ can contain arbitrary relation symbols (but neither constant nor function symbols), and our upper bounds will hold for all such structures. However, for our *lower bounds* it will suffice to consider only *undirected graphs*. That is, the lower bounds for a pattern p will be of the form “there is a ϕ with pattern p such that $\text{p-MODELS}_{\text{undirected}}(\phi)$ contains a hard problem”. Interestingly, we can typically (*but not always*, by the results of Section 4) replace *undirected graphs* by *basic graphs* (undirected graphs without self-loops) here.

3.1 Solution Weight Equals the Parameter for Arbitrary Structures

We start with the classification of $\text{p-FD}(E_1^=p)$, the first two lines of Table 1:

- **Theorem 3.1** (Complexity Dichotomy for $\text{p-FD}(E_1^=p)$). *Let $p \in \{a, e\}^*$ be a pattern.*
1. $\text{p-FD}(E_1^=p) \subseteq \text{para-AC}^0$, if $p \preceq e^*a$.
 2. $\text{p-FD}(E_1^=p)$ contains a $\text{W}[1]$ -hard problem, if $aa \preceq p$ or $ae \preceq p$.
- Both items also hold for $\text{p-FD}_{\text{undirected}}(E_1^=p)$ and even $\text{p-FD}_{\text{basic}}(E_1^=p)$.*

The cases in the above theorem are exhaustive (so for every p we either have $p \preceq e^*a$ or we have $aa \preceq p$ or $ae \preceq p$). The theorem follows directly from the following lemma:

- **Lemma 3.2** (Detailed Bounds for $\text{p-FD}(E_1^=p)$).
1. $\text{p-FD}(E_1^=e^*a) \subseteq \text{para-AC}^0$.
 2. $\text{p-FD}_{\text{basic}}(E_1^=aa)$ contains a $\text{W}[1]$ -hard problem.
 3. $\text{p-FD}_{\text{basic}}(E_1^=ae)$ contains a $\text{W}[2]$ -hard problem.

Proof. Item 1 is shown in Corollary 3.5, which we prove later in this section. For item 2, we already saw in equation (2) that we can describe the $W[1]$ -hard clique problem p-CLIQUE using a formula ϕ_{clique} with pattern $E_1^{\geq}aa$. It was also already mentioned that replacing \exists^{\geq} by $\exists^=$ yields the same problem and, thus, $\text{p-FD}_{\text{basic}}(E_1^=aa)$ contains a $W[1]$ -hard problem. Similarly, for item 3, replacing \exists^{\leq} by $\exists^=$ in equation (4) shows we can describe the $W[2]$ -hard dominating set problem using an $E_1^=ae$ formula. ◀

To establish the upper bound (item 1 of the theorem), we make use of a well-known connection between weighted satisfiability in predicate logic (problems in $\text{p-FD}(E_1^=e^*a)$ in our case) and weighted satisfiability in propositional logic (the problem $\text{p-1WSAT}^=$ below). We present this connection in more generality than strictly necessary to prove the upper bound since we will rely on variants of it later on. For propositional formulas ψ in $d\text{CNF}$ (meaning *at most* d literals per clause), let $\text{vars}(\psi)$ and $\text{clauses}(\psi)$ denote the sets of variables and clauses, respectively. For an *assignment* $\beta: \text{vars}(\psi) \rightarrow \{0, 1\}$, with the model relation $\beta \models \psi$ defined as usual, the *weight* is $\text{weight}(\beta) = |\{v \in \text{vars}(\psi) \mid \beta(v) = 1\}|$. The following problem is the weighted version of the satisfiability problem for $d\text{CNF}$ formulas:

► **Problem 3.3** ($\text{p-}d\text{WSAT}^=$ for fixed d).

Instance: A $d\text{CNF}$ formula ψ and a non-negative integer $k \in \mathbb{N}$.

Parameter: k

Question: Is there an assignment β with $\beta \models \psi$ and $\text{weight}(\beta) = k$?

The problem is also known as $\text{p-}d\text{WSAT}$ in the literature, but we keep the “=” superscript since we also consider $\text{p-}d\text{WSAT}^{\leq}$ and $\text{p-}d\text{WSAT}^{\geq}$, where we ask whether there is a satisfying assignment with $\text{weight}(\beta) \leq k$ and $\text{weight}(\beta) \geq k$, respectively. For us, the importance of these problems lies in the following lemma, where “ $\leq_{\text{para-AC}^0}$ ” refers to the earlier-mentioned para-AC^0 -reductions. Recall that these reductions are extremely weak and that all classes considered in this paper, including para-AC^0 , are closed under them.

► **Lemma 3.4.** *Let $d \geq 1$. Then:*

1. *For every $Q \in \text{p-FD}(E_1^=e^*a^d)$ we have $Q \leq_{\text{para-AC}^0} \text{p-}d\text{WSAT}^=$.*
2. *For every $Q \in \text{p-FD}(E_1^{\geq}e^*a^d)$ we have $Q \leq_{\text{para-AC}^0} \text{p-}d\text{WSAT}^{\geq}$.*
3. *For every $Q \in \text{p-FD}(E_1^{\leq}e^*a^d)$ we have $Q \leq_{\text{para-AC}^0} \text{p-}d\text{WSAT}^{\leq}$.*

Proof. In all three cases, Q is the set of models of a weighted ESO formula of the form $\exists^=X \phi(X)$ or $\exists^{\leq}X \phi(X)$ or $\exists^{\geq}X \phi(X)$ where $\phi(X)$ has the quantifier pattern e^*a^d . In [13, Lemma 7.2] it is shown that given a formula $\phi(X)$ with such a pattern, we can map any structure \mathcal{S} with some universe S to a $d\text{CNF}$ formula ψ such that there is a one-to-one correspondence between the sets $C \subseteq S$ with $\mathcal{S} \models \phi(C)$ and the satisfying assignments β of ψ . Furthermore, when C corresponds to β , we have $|C| = \text{weight}(\beta)$. While in [13] it is only argued that the mapping from $\phi(X)$ to ψ can be done in polynomial time, a closer look reveals that a para-AC^0 reduction suffices. This means that in all three items we can use this mapping as the reduction whose existence is claimed. ◀

► **Corollary 3.5.** $\text{p-FD}(E_1^=e^*a)$, $\text{p-FD}(E_1^{\leq}e^*a)$, $\text{p-FD}(E_1^{\geq}e^*a)$ are subsets of para-AC^0 .

Proof. Let us start with some $Q \in \text{p-FD}(E_1^=e^*a)$. By item 1 of Lemma 3.4, $Q \leq_{\text{para-AC}^0} \text{p-1WSAT}^=$. Thus, showing $\text{p-1WSAT}^= \in \text{para-AC}^0$ yields the claim for $\text{p-FD}(E_1^=e^*a)$ as para-AC^0 is closed under para-AC^0 reductions. However, a 1CNF formula ψ is just a conjunction of literals. It is trivial to check in plain AC^0 (independently of the parameter) whether ψ is satisfiable (it may not contain a literal and its negation) and, if so, it is trivial

to determine the single satisfying assignment $\beta: \text{vars}(\psi) \rightarrow \{0, 1\}$. We are left with having to check whether $\text{weight}(\beta) = k$ holds. It is well known [3] that the problem of checking whether the number of 1 bits in a bitstring is at least, at most, or equal to a parameter value lies in para-AC^0 , yielding the claim. However, this also yields the other two items. ◀

3.2 Solution Weight Is At Least the Parameter for Arbitrary Structures

For $\text{p-FD}(E_1^{\geq} p)$, we get the exact same dichotomy as for $\text{p-FD}(E_1^= p)$. However, a look at the detailed bounds in the lemma shows that for $\text{p-FD}(E_1^{\geq} p)$ we only get a lower bound for *undirected* graphs and not for *basic* graphs (and, indeed, we will show in Section 4 that the complexity is different for basic graphs). Furthermore, while we always have $\text{p-FD}(E_1^= p) \subseteq W[t]$ for some t (see [13, Definition 5.1]), we show that the patterns $E_1^{\geq} eae$ or $E_1^{\geq} aee$ suffice to describe even para-NP -complete problems even on basic graphs. Thus, although the tractability frontier (“in FPT” versus “contains $W[1]$ -hard problems”) is the same for $\text{p-FD}(E_1^= p)$ and $\text{p-FD}(E_1^{\geq} p)$, the detailed structure is more complex.

► **Theorem 3.6** (Complexity Dichotomy for $\text{p-FD}(E_1^{\geq} p)$). *Let p be a pattern.*

1. $\text{p-FD}(E_1^{\geq} p) \subseteq \text{para-AC}^0$, if $p \preceq e^* a$.
 2. $\text{p-FD}(E_1^{\geq} p)$ contains a $W[1]$ -hard problem, if $aa \preceq p$ or $ae \preceq p$.
- Both items also hold for $\text{p-FD}_{\text{undirected}}(E_1^{\geq} p)$.

The theorem follows directly from the following lemma (whose last two items are not actually needed here, but shed more light on the detailed structure and *will* be needed in Section 4).

► **Lemma 3.7** (Detailed Bounds for $\text{p-FD}(E_1^{\geq} p)$).

1. $\text{p-FD}(E_1^{\geq} e^* a) \subseteq \text{para-AC}^0$.
2. $\text{p-FD}_{\text{basic}}(E_1^{\geq} aa)$ contains a $W[1]$ -hard problem.
3. $\text{p-FD}_{\text{undirected}}(E_1^{\geq} ae)$ contains a para-NP -hard problem.
4. $\text{p-FD}_{\text{basic}}(E_1^{\geq} eae)$ contains a para-NP -hard problem.
5. $\text{p-FD}_{\text{basic}}(E_1^{\geq} aee)$ contains a para-NP -hard problem.

Proof. Item 1 is already stated in Corollary 3.5. For item 2, equation 2 shows that the $W[1]$ -complete problem p-CLIQUE can be expressed with a weighted ESO formula with the pattern $E_1^{\geq} aa$ and, thus, lies in $\text{p-FD}_{\text{basic}}(E_1^{\geq} aa)$.

For the other items, a claim is useful:

▷ **Claim 3.8.** If there is an NP-hard problem in $\text{FD}(E_1 p)$, there is a para-NP -hard problem in $\text{p-FD}(E_1^{\geq} p)$; and this holds also for the restrictions to undirected, basic, or directed graphs.

To see that this claim holds, just note that the non-parameterized problem is the special case of the parameterized maximization problem where $k = 0$.

To prove item 3, observe that Gottlob et al. have shown [15, Theorem 2.1] that there are NP-complete problems in $\text{FD}_{\text{undirected}}(E_1 ae)$. By the claim, there must be para-NP -hard problems in $\text{p-FD}_{\text{undirected}}(E_1^{\geq} ae)$. Next, for item 4, in [15, Theorem 2.5] it is shown that there is an NP-hard problem in $\text{FD}_{\text{basic}}(E_1 eae)$. Finally, for item 5, by [15, Theorem 2.6] there is also an NP-hard problem in $\text{FD}_{\text{basic}}(E_1 aee)$. ◀

Note that compared to Lemma 3.2, in the third item of Lemma 3.7 we have shown the lower bound for the restriction of structures to undirected graphs rather than basic graphs. This is no coincidence: the self-loops which are allowed for undirected graphs have in some cases an impact on the complexity of the problems we can express. Later, when we cover basic graphs, we will see that, indeed, sometimes problems become easier compared to their counterparts where undirected graphs are admissible as structures.

3.3 Solution Weight Is At Most the Parameter for Arbitrary Structures

When the parameter is an upper bound on the weight of solutions, the tractability landscape changes quite a bit: The pattern $E_1^{\leq}ae$ becomes intractable, while $E_1^{\leq}aa$ no longer lies in para-AC^0 , but stays tractable:

► **Theorem 3.9** (Complexity Trichotomy for $\text{p-FD}(E_1^{\leq}p)$). *Let p be a pattern.*

1. $\text{p-FD}(E_1^{\leq}p) \subseteq \text{para-AC}^0$, if $p \preceq e^*a$.
2. $\text{p-FD}(E_1^{\leq}p) \subseteq \text{para-AC}^{0\uparrow}$ but $\text{p-FD}(E_1^{\leq}p) \not\subseteq \text{para-AC}^0$, if $aa \preceq p \preceq e^*a^*$.
3. $\text{p-FD}(E_1^{\leq}p)$ contains a $\text{W}[1]$ -hard problem, if $ae \preceq p$.

All items also hold for $\text{p-FD}_{\text{undirected}}(E_1^{\leq}p)$.

As before, the theorem covers all possible p and follows from a lemma that is a bit more general than strictly necessary: We will need items 4 and 5 only in Section 4.2, where we show that item 3 does *not* hold for basic graphs.

► **Lemma 3.10** (Detailed Bounds for $\text{p-FD}(E_1^{\leq}p)$).

1. $\text{p-FD}(E_1^{\leq}e^*a) \subseteq \text{para-AC}^0$.
2. $\text{p-FD}(E_1^{\leq}e^*a^*) \subseteq \text{para-AC}^{0\uparrow}$.
3. $\text{p-FD}_{\text{undirected}}(E_1^{\leq}aa)$ contains a problem not in para-AC^0 .
4. $\text{p-FD}_{\text{basic}}(E_1^{\leq}aaa)$ contains a problem not in para-AC^0 .
5. $\text{p-FD}_{\text{basic}}(E_1^{\leq}eaa)$ contains a problem not in para-AC^0 .
6. $\text{p-FD}_{\text{basic}}(E_1^{\leq}ae)$ contains a $\text{W}[2]$ -hard problem.

Proof. Item 1 is already stated in Corollary 3.5. Item 2 is shown in Lemma 3.11 below. Items 3, 4, and 5 are shown in Lemma 3.14, Lemma 3.15, and Lemma 3.16, respectively. Item 6 follows, once more, from $\text{p-DOMINATING-SET} \in \text{p-FD}_{\text{basic}}(E_1^{\leq}ae)$ by equation (4). ◀

► **Lemma 3.11.** $\text{p-FD}(E_1^{\leq}e^*a^*) \subseteq \text{para-AC}^{0\uparrow}$.

Proof. Let $Q \in \text{p-FD}(E_1^{\leq}e^*a^d)$ for some fixed d . By Lemma 3.4, $Q \leq_{\text{para-AC}^0} \text{p-dWSAT}^{\leq}$. We now show that $\text{p-dWSAT}^{\leq} \in \text{para-AC}^{0\uparrow}$, which implies the claim as $\text{para-AC}^{0\uparrow}$ is closed under para-AC^0 reductions.

We have to construct a circuit family of depth $f(k)$ and size $f(k) \cdot n^{O(1)}$ for $n = |\text{vars}(\psi)|$ for some computable function f . The circuit implements a bounded search tree such that every layer evaluates one level of the tree. To that end, each layer gets a set Ψ_i of formulas as input and outputs a new set Ψ_{i+1} of formulas. We start with $\Psi_0 = \{\psi\}$. The invariant will be that ψ has a satisfying assignment of (exact) weight w iff some formula in Ψ_i has a satisfying assignment of (exact) weight $w - i$.

To compute the next Ψ_{i+1} for $i \in \{0, \dots, k\}$, we perform the following operations in parallel for every $\rho \in \Psi$:

1. If every clause in ρ contains a negative literal (meaning that ρ is satisfied by the all-0 assignment), accept the original input. (Doing so is correct by the invariant.)
2. Take a clause $c \in \text{clauses}(\rho)$ that only contains positive literals x_1, \dots, x_e . For each x_i , generate a new formula ρ^i from ρ by “setting one of these variables to 1” or, formally, by removing all clauses that contain it positively and removing the variable from all clauses that contain it negatively, respectively. Add ρ^1 to ρ^e to Ψ_{i+1} . (This maintains the invariant as we *must* set one of the x_i to 1 in any assignment that satisfies ρ .)

If we have not accepted the input after having computed Ψ_{k+1} , we reject. This is correct since all satisfying assignments of the $\rho \in \Psi_{k+1}$ have weight at least 0 and, thus, by the invariant all satisfying assignments of ψ have weight at least $k + 1 - 0 > k$.

3:10 ESO-Logic over Graphs: Parameterized Complexity

To see that the circuit can be implemented with the claimed depth and size, note that since $e \leq d$, the list grows by a factor of at most d in every layer and we can implement each layer in constant depth. As there are only $k+1$ layers, we have $|\Psi_{k+1}| \leq (k+1)^d =: f(k)$. ◀

For the three remaining still-to-be-proved lower bounds in Lemma 3.10, the claim is always that a class is (unconditionally) not contained in para-AC^0 . To prove this, we will show that the following problem is (provably) not in para-AC^0 but can be para-AC^0 -reduced to problems that lie in the three classes:

► **Problem 3.12** (p-MATCHED-REACH).

Instance: A directed layered graph G with vertex set $\{1, \dots, n\} \times \{1, \dots, k\}$, where the i th layer is $V_i := \{1, \dots, n\} \times \{i\}$, such that for each $i \in \{1, \dots, k-1\}$ the edges point to the next layer and they form a perfect matching between V_i and V_{i+1} ; two designated vertices $s \in V_1$ and $t \in V_k$.

Parameter: k .

Question: Is t reachable from s ?

► **Lemma 3.13.** $\text{p-MATCHED-REACH} \notin \text{para-AC}^0$ and consequently, for any problem Q with $\text{p-MATCHED-REACH} \leq_{\text{para-AC}^0} Q$ we have $Q \notin \text{para-AC}^0$.

Proof. Beame et al. [5] have shown that any depth- c circuit that solves p-MATCHED-REACH requires size $n^{\Omega(k^{\rho^{-2c}/3})}$, where ρ is the golden ratio. However, $\text{p-MATCHED-REACH} \in \text{para-AC}^0$ would imply that for some constant c there is a depth- c circuit family that decides the problem in size $f(k) \cdot n^{O(1)}$; contradicting the Beame et al. bound of $n^{k^{\Theta(1)}}$. For the claim concerning Q , just note that para-AC^0 is closed under para-AC^0 reductions. ◀

► **Lemma 3.14.** $\text{p-FD}_{\text{undirected}}(E_1^{\leq} aa) \not\subseteq \text{para-AC}^0$.

Proof. Consider the following formula with quantifier pattern $E_1^{\leq} aa$:

$$\phi_{\text{reach}} := \exists^{\leq} S \forall x \forall y ((x \sim x) \rightarrow Sx) \wedge ((Sx \wedge x \sim y) \rightarrow Sy).$$

We claim that we can reduce p-MATCHED-REACH to $\text{p-MODELS}_{\text{undirected}}(\phi_{\text{reach}})$ as follows (and the claim then follows from Lemma 3.13): On input (G, s, t) , the reduction first checks that the graph is, indeed, a layered graph with perfect matchings between consecutive levels. Then, we forget about the direction of the edges (making the graph undirected). Next, we add an additional layer V_{k+1} and match each vertex of V_k to the corresponding new vertex V_{k+1} . Next, we *remove* the just-added edge from t in layer V_k to its counterpart in layer V_{k+1} . Finally, add a self-loop at s . To see that this reduction is correct, note that the self-loop at s forces it (but does not *force* any other vertex), to be part of the solution set S by the first part of the formula. The second part then forces the solution set to be closed under reachability. Thus, if t lies on the same path as s , there is a solution of size k , and if not, the smallest solution has size $k+1$. ◀

► **Lemma 3.15.** $\text{p-FD}_{\text{basic}}(E_1^{\leq} aaa) \not\subseteq \text{para-AC}^0$.

Proof. We reduce p-MATCHED-REACH to $\text{p-MODELS}_{\text{basic}}(\phi_{\text{reach-aaa}})$ for

$$\phi_{\text{reach-aaa}} := \exists^{\leq} S \forall x \forall y \forall z (((x \sim y \wedge y \sim z \wedge x \sim z) \rightarrow Sx) \wedge ((Sx \wedge x \sim y) \rightarrow Sy)).$$

On input (G, s, t) , once more we start by forgetting about the direction of the edges. This time, add two vertices and connect them to s so that these three vertices form a triangle. Do the same for t by adding another two vertices. Output $k+4$ as the new parameter. This

reduction is correct, because the first part of the formula forces every vertex which is part of a triangle to be part of S , which are exactly the triangles at s and t . The latter part of the formula forces the solution set to be closed under reachability. Thus, if t lies on the same path as s , there is a solution of size $k + 4$, and if not, the smallest solution has size at least $k + 5$. \blacktriangleleft

► **Lemma 3.16.** $\text{p-FD}_{\text{basic}}(E_1^{\leq} eaa) \not\subseteq \text{para-AC}^0$.

Proof. We reduce p-MATCHED-REACH to $\text{p-MODELS}_{\text{basic}}(\phi_{\text{reach-aaa}})$ for

$$\phi_{\text{reach-aaa}} := \exists^{\leq} S \exists z \forall x \forall y (Sz \wedge ((Sx \wedge x \sim y) \rightarrow Sy)).$$

On input (G, s, t) we forget the direction of edges and add a single vertex that we connect to every vertex that has degree 1 except for s and t . If t is on the same path as s , there will be two connected components: One consisting of the path between s and t , and one containing everything else. In particular, there is a component of size k and one of size $n - k$. If t is not on the same path as s , there is just a single connected component of size n . To see that the reduction is correct, notice that the first part of the formula (Sz) forces at least one vertex to be part of the solution. The latter part of the formula once more forces the solution set to be closed under reachability. By construction, there is a solution of size at most k iff t was on the same path as s . \blacktriangleleft

4 Classifying Parameterized ESO Classes: Basic Graphs

We saw in Section 3 that the parameterized complexity of weighted ESO classes depends strongly on the first-order quantifier pattern p and on whether we are interested in the equal-to, at-least, or at-most case – but it does *not* matter whether we consider arbitrary logical structures, only directed graphs, or only undirected graphs: the complexity is always the same. The situation changes if we restrict attention to *basic graphs*, which are undirected graphs without self-loops: We get different tractability frontiers. This is an interesting effect since the only difference between undirected graphs and basic graphs is that some vertices may have self-loops – and self-loops are usually neither needed nor used in hardness proofs, just think of the clique problem, the vertex cover problem, or the dominating set problem. Nevertheless, it turns out that “a single extra bit per vertex” and sometimes even “a single self-loop” allows us to encode harder problems than without.

To establish the tractability frontier for basic graphs, we can, of course, recycle many results from the previous section: Having a look at the detailed bounds listed in Lemmas 3.2, 3.7, and 3.10, we see that the upper bounds are shown for arbitrary structures and, hence, also hold for basic graphs; and many lower bounds have also already been established for basic graphs. Indeed, it turns out there are exactly two classes whose complexity “changes” when we restrict the inputs to basic graphs:

1. $\text{p-FD}_{\text{basic}}(E_1^{\geq} ae)$ lies in para-AC^0 , while $\text{p-FD}(E_1^{\geq} ae)$ does not.
2. $\text{p-FD}_{\text{basic}}(E_1^{\leq} aa)$ lies in para-AC^0 , while $\text{p-FD}(E_1^{\leq} aa)$ does not.

We have already shown the “while . . .” part in Section 3, it is the upper bounds that are new. For all other patterns p , the classification does not change. Proving the two items turns out to be technical and we devote one subsection to each of these results.

4.1 The Case $E_1^{\geq} ae$ for Basic Graphs

As mentioned, for the classification of the complexity of $\text{p-FD}_{\text{basic}}(E_1^{\geq} p)$ we can reuse all of our previous results, *except* that $\text{p-FD}_{\text{basic}}(E_1^{\geq} ae) \subseteq \text{para-AC}^0$ holds. This is the statement of Lemma 4.2, which is proved in the rest of this section. However, before we plunge into

the glorious details, let us ascertain that there are no further patterns $p \neq ae$ for which $\text{p-FD}_{\text{basic}}(E_1^{\geq} p)$ becomes any easier: To see this, note that for any p with $p \not\preceq ae$ we have $aa \preceq p$ or $eae \preceq p$ or $ae \preceq p$; and for aa , eae , and ae we have already established hardness for *basic* graphs in Lemma 3.7. For completeness, we spell out the resulting structure:

► **Theorem 4.1** (Dichotomy for $\text{p-FD}_{\text{basic}}(E_1^{\geq} p)$). *Let p be a pattern.*

1. $\text{p-FD}_{\text{basic}}(E_1^{\geq} p) \subseteq \text{para-AC}^0$, if $p \preceq e^*a$ or $p \preceq ae$.
2. $\text{p-FD}_{\text{basic}}(E_1^{\geq} p)$ contains a $\text{W}[1]$ -hard problem, if $aa \preceq p$, $eae \preceq p$, or $ae \preceq p$.


► **Lemma 4.2.** $\text{p-FD}_{\text{basic}}(E_1^{\geq} ae) \subseteq \text{para-AC}^0$.

For the surprisingly difficult proof we employ machinery first used in [15] and in [17, Section 3.3]: Our objective is to represent the problems in $\text{p-FD}_{\text{basic}}(E_1^{\geq} ae)$ as special kinds of graph coloring problems – and to then show that we can solve these problems in para-AC^0 .

Proof idea. Following [15], a *pattern graph* $P = (C, A^{\oplus}, A^{\ominus})$ consists of a set of *colors* C , a set $A^{\oplus} \subseteq C \times C$ of \oplus -arcs, and a set $A^{\ominus} \subseteq C \times C$ of \ominus -arcs (note that A^{\oplus} and A^{\ominus} need not be disjoint). In our paper, we will only need the case that there are only two colors, so $C = \{\text{black}, \text{white}\}$ will always hold, and we call such a pattern graph *binary*. In the rest of the section, *pattern* always refers to a *binary pattern graph* (and no longer to quantifier prenex patterns – we are only interested in the single pattern $E_1^{\geq} ae$ anyway). Observe that there are 256 possible binary pattern graphs. A *superpattern* of a pattern $P = (C, A^{\oplus}, A^{\ominus})$ is any pattern $P' = (C, B^{\oplus}, B^{\ominus})$ with $A^{\oplus} \subseteq B^{\oplus}$ and $A^{\ominus} \subseteq B^{\ominus}$. A \oplus -*superpattern* is a superpattern with $A^{\ominus} = B^{\ominus}$.

For a basic graph $B = (V, E)$, a *coloring* of B is a function $c: V \rightarrow C$. However, unlike standard coloring problems, where vertices connected by an edge must have different colors, what constitutes an allowed coloring is dictated by the pattern graph via a *witness function*: A mapping $w: V \rightarrow V$ is called a *witness function for a coloring* c if for all $x \in V$ we have

1. $x \neq w(x)$,
2. if $\{x, w(x)\} \in E$, then $(c(x), c(w(x))) \in A^{\oplus}$, and
3. if $\{x, w(x)\} \notin E$, then $(c(x), c(w(x))) \in A^{\ominus}$.

The idea is that a vertex x and its witness $w(x)$ are connected by “a \oplus -arc” if there is an edge between them and by “a \ominus -arc” if there is no edge between them. The pattern graph then tells us which colors are allowed for x and $w(x)$ in dependence on which kind of arc there is. For instance, for the pattern  every vertex must be connected by an edge to a vertex of the opposite color. Note that this is not the same as asking for a 2-coloring: We only impose a requirement on the edge (corresponding to a \oplus -arc) between x and $w(x)$, other edges are not relevant. In more detail, consider a triangle with the vertices $\{x, y, z\}$ and the coloring $c(x) = \text{black}$, $c(y) = c(z) = \text{white}$ and the witness function $w(x) = y$ and $w(y) = w(z) = x$. Then the coloring is legal with respect to the pattern and the witness function, despite that fact that a triangle is not 2-colorable.

If there exists a coloring c together with a witness function w for B with respect to P , we say that B is *P -saturated by c and w* . The saturation problem $\text{SATURATION}(P)$ for a pattern P is then simply the set of all basic graphs $B = (V, E)$ that can be P -saturated (via some coloring c and witness function w). The relation between the saturation problem and $E_1^{\geq} ae$ is as follows: We want the witness function to tell us for each x in $\forall x$ which y in $\exists y$ we must pick to make a formula of the form $\exists S \forall x \exists y \psi$ true: *We color a vertex black to indicate that it should be included in S , otherwise we color it white*. In this way, one can associate a pattern graph with each $E_1^{\geq} ae$ -formula.

► **Fact 4.3** ([17, Fact 3.3] for a single quantifier). *For every ESO formula ϕ with quantifier pattern E_1ae there is a binary pattern graph P such that $\text{MODELS}(\phi) = \text{SATURATION}(P)$.*

(Strictly speaking, this only holds for basic graphs B with at least two vertices. For this reason, in the following we always assume that $|V| \geq 2$ holds.)

Adapting this approach to the parameterized setting is straightforward: Define the *weight* of a binary coloring as the number of vertices that are colored black. This leads to the following parameterized problem and transfer of Fact 4.3 to the parameterized setting:

► **Problem 4.4** ($\text{p-SATURATION}^{\geq}(P)$ for a fixed binary pattern graph $P = (C, A^{\oplus}, A^{\ominus})$).

Instance: A basic graph $B = (V, E)$ and an integer $k \in \mathbb{N}$.

Parameter: k .

Question: Can B be P -saturated via a coloring of weight at least k ?

▷ **Claim 4.5.** For every weighted ESO formula ϕ with quantifier pattern $E_1^{\geq}ae$ there is a binary pattern graph P such that $\text{p-MODELS}(\phi) = \text{p-SATURATION}^{\geq}(P)$.

With the above claim, it remains to show for all 256 binary pattern graphs P that $\text{p-SATURATION}^{\geq}(P) \in \text{para-AC}^0$ holds. The (surprisingly diverse and nontrivial) treatment of the cases can be found in the technical report version [2]. ◀

4.2 The Case $E_1^{\leq}aa$ for Basic Graphs

The classification of the complexity of $E_1^{\leq}p$ also changes when we restrict the admissible input structures to be basic graphs: $\text{p-FD}_{\text{basic}}(E_1^{\leq}aa) \subseteq \text{para-AC}^0$ holds by Lemma 4.7 and, once more, this is the only change. Proving the lemma will be considerably easier than in the previous section, but still demanding. Before we start, we summarize the resulting landscape for completeness. Note that all bounds other than the just-mentioned new upper bound have already been shown in Lemma 3.10.

► **Theorem 4.6** (Trichotomy for $\text{p-FD}_{\text{basic}}(E_1^{\leq}p)$). *Let p be a pattern.*

1. $\text{p-FD}_{\text{basic}}(E_1^{\leq}p) \subseteq \text{para-AC}^0$ if $p \preceq e^*a$ or $p \preceq aa$.
2. $\text{p-FD}_{\text{basic}}(E_1^{\leq}p) \subseteq \text{para-AC}^{0\uparrow}$ but $\text{p-FD}_{\text{basic}}(E_1^{\leq}p) \not\subseteq \text{para-AC}^0$, if $aaa \preceq p$ or $eea \preceq p$, and $p \preceq e^*a^*$.
3. $\text{p-FD}_{\text{basic}}(E_1^{\leq}p)$ contains a W[1]-hard problem, if $ae \preceq p$.

► **Lemma 4.7.** $\text{p-FD}_{\text{basic}}(E_1^{\leq}aa) \subseteq \text{para-AC}^0$.

Proof idea. As in the previous section, we can reuse some ideas from the literature, but need to take care of some extra complications caused by the need to limit the sizes of the solution sets. In particular, we will use the notion of *cardinality constraints* introduced in [17] for the study of the E_1aa case: For two sets $C, D \subseteq \{0, 1, 2\}$ define $\text{p-CSP}^{\leq}\{C, D\}$ as follows. The instances for this problem consist of a finite universe U , a function P that maps each two-element subset $\{x, y\} \subseteq U$ to either C or D (so, unlike normal constraint satisfaction problems, a constraint must be stated for every single pair of variables), and a number k . A *solution* for P is a subset $X \subseteq U$ of size $|X| \leq k$ such that for all two-element subsets $\{x, y\} \subseteq U$ we have $|\{x, y\} \cap X| \in P(x, y)$. We call $(U, P^{-1}(C))$ the *C-graph* of P and note that this is just the set of edges that are mapped to C by P . The *D-graph* is defined as $(U, P^{-1}(D))$; and observe that every two-element set belongs to exactly one of these two graphs except when $C = D$ in which case both graphs are the complete cliques. It is shown in [17, Lemma 3.1] that all problems in $\text{FD}(Eaa)$ reduce to $\text{CSP}\{C, D\}$ (without the “ $\leq k$ ” restrictions) for some C and D . We need to following variant:

▷ Claim 4.8. All problems in $\text{p-FD}(E_1^{\leq}aa)$ reduce to $\text{p-CSP}^{\leq}\{C, D\}$ for some C and D via para-AC^0 reductions.

Proof. The reduction is a trivial reencoding in which the solutions of the CSP instances are exactly the sets that satisfy the formula when assigned to the existentially bound second-order variable. In particular, solutions and assigned sets have the same sizes and satisfy the same size restrictions. ◁

It remains to show $\text{p-CSP}^{\leq}\{C, D\} \in \text{para-AC}^0$ for all $C, D \subseteq \{0, 1, 2\}$. For this, we have to go over the possible choices in a case distinction. Once more, the detailed treatment of the cases can be found in the technical report version [2]. ◀

5 Conclusion

We gave a complete characterization of the tractability frontier of weighted ESO logic over basic graphs, undirected graphs, and arbitrary structures. While in some cases our results mirror the classical complexity landscape, other cases yield clearly different results. The proofs differ significantly from the classical setting and make extensive use of tools from parameterized complexity theory. Especially for the class $\text{p-FD}_{\text{basic}}(E^{\geq}ae)$, sophisticated machinery is needed to establish the upper bound. Whether we require solutions to have size exactly k , at most k , or at least k plays a central role in the complexity of the describable problems. While the class $\text{p-FD}_{\text{basic}}(E^{\geq}ae)$ can be shown to be included in para-AC^0 , the classes $\text{p-FD}_{\text{basic}}(E^=ae)$ and $\text{p-FD}_{\text{basic}}(E^{\leq}ae)$ both contain $\text{W}[2]$ -hard problems. Similarly, while $\text{p-FD}_{\text{basic}}(E^{\leq}aa)$ is contained in para-AC^0 , both $\text{p-FD}_{\text{basic}}(E^=aa)$ and $\text{p-FD}_{\text{basic}}(E^{\geq}aa)$ contain $\text{W}[1]$ -hard problems.

An obvious further line of research is to consider the prefixes $E_i^=p$, $E_i^{\geq}p$, and $E_i^{\leq}p$ for $i \geq 2$, that is, the non-monic case, and also multiple monadic quantifiers. While in the classical setting it turns out [15] that we can normally reduce non-monic quantifiers to (possibly multiple) monadic ones, it is not clear whether the same happens in the parameterized setting. Just pinpointing the complexity of, say, $\text{p-FD}_{\text{basic}}(E_2^{\geq}ae)$ seems difficult.

A different line of inquiry is to further investigate the patterns that lead to intractable problems: In the unweighted setting, all ESO-definable problems lie in NP and if the class is not contained in P, then it contains an NP-complete problem. Our intractability results range from $\text{W}[1]$ -completeness to para-NP -completeness. Can we find for every t a pattern for which we get classes that contain $\text{W}[t]$ -hard problems and are contained in $\text{W}[t]$?

Our results also shed some light on *graph modification problems*, where we have a fixed first-order formula ϕ and are given a pair (G, k) . The objective is to modify the graph as little as possible (for instance, by deleting as few vertices as possible) such that for the resulting graph G' we have $G' \models \phi$. Fomin et al. [14] have recently shown a complexity dichotomy regarding the number quantifier alternations in ϕ . Since it is not difficult to encode the “to be deleted vertices” using a $\exists^{\leq D}$ quantifier, at least the upper bounds from our paper also apply to vertex deletion problems. We believe that our results can be extended to also cover lower bounds and, thereby, to give exact and complete classifications of the complexity of vertex deletion problems in terms of the quantifier pattern of ϕ .



References

- 1 Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.
- 2 Max Bannach, Florian Chudigiewitsch, and Till Tantau. Existential second-order logic over graphs: Parameterized complexity. Technical Report abs/2310.01134, Cornell University, 2023. URL: <https://arxiv.org/abs/2310.01134>.
- 3 Max Bannach, Christoph Stockhusen, and Till Tantau. Fast parallel fixed-parameter algorithms via color coding. In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, pages 224–235, 2015. doi:10.4230/LIPIcs.IPEC.2015.224.
- 4 Max Bannach and Till Tantau. Computing kernels in parallel: Lower and upper bounds. In *13th International Symposium on Parameterized and Exact Computation, IPEC 2018, August 20-24, 2018, Helsinki, Finland*, pages 13:1–13:14, 2018. doi:10.4230/LIPIcs.IPEC.2018.13.
- 5 Paul Beame, Russell Impagliazzo, and Toniann Pitassi. Improved depth lower bounds for small distance connectivity. *Comput. Complex.*, 7(4):325–345, 1998. doi:10.1007/s000370050014.
- 6 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6(1-6):66–92, 1960.
- 7 Yijia Chen and Jörg Flum. Some lower bounds in parameterized ac^0 . In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 27:1–27:14, 2016. doi:10.4230/LIPIcs.MFCS.2016.27.
- 8 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012. URL: http://www.cambridge.org/fr/knowledge/isbn/item5758776/?site_locale=fr_FR.
- 9 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.
- 10 Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical logic (2. ed.)*. Springer, 1994.
- 11 Thomas Eiter, Yuri Gurevich, and Georg Gottlob. Existential second-order logic over strings. *J. ACM*, 47(1):77–131, 2000. doi:10.1145/331605.331609.
- 12 Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of Computation*, 7:43–74, 1974.
- 13 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006. doi:10.1007/3-540-29953-X.
- 14 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. On the parameterized complexity of graph modification to first-order logic properties. *Theory Comput. Syst.*, 64(2):251–271, 2020. doi:10.1007/s00224-019-09938-8.
- 15 Georg Gottlob, Phokion G. Kolaitis, and Thomas Schwentick. Existential Second-Order Logic Over Graphs: Charting the Tractability Frontier. *Journal of the ACM*, 51(2):312–362, 2004. doi:10.1145/972639.972646.
- 16 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
- 17 Till Tantau. Existential second-order logic over graphs: A complete complexity-theoretic classification. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 703–715, 2015. doi:10.4230/LIPIcs.STACS.2015.703.

On the Complexity of Finding a Sparse Connected Spanning Subgraph in a Non-Uniform Failure Model

Matthias Bentert  

Department of Informatics, University of Bergen, Norway

Jannik Schestag  

Faculteit Elektrotechniek, Wiskunde en Informatica, TU Delft, The Netherlands

Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena, Germany

Frank Sommer   

Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena, Germany

Abstract

We study a generalization of the classic SPANNING TREE problem that allows for a non-uniform failure model. More precisely, edges are either *safe* or *unsafe* and we assume that failures only affect unsafe edges. In UNWEIGHTED FLEXIBLE GRAPH CONNECTIVITY we are given an undirected graph $G = (V, E)$ in which the edge set E is partitioned into a set S of safe edges and a set U of unsafe edges and the task is to find a set T of at most k edges such that $T - \{u\}$ is connected and spans V for any unsafe edge $u \in T$. UNWEIGHTED FLEXIBLE GRAPH CONNECTIVITY generalizes both SPANNING TREE and HAMILTONIAN CYCLE. We study UNWEIGHTED FLEXIBLE GRAPH CONNECTIVITY in terms of fixed-parameter tractability (FPT). We show an almost complete dichotomy on which parameters lead to fixed-parameter tractability and which lead to hardness. To this end, we obtain FPT-time algorithms with respect to the vertex deletion distance to cluster graphs and with respect to the treewidth. By exploiting the close relationship to HAMILTONIAN CYCLE, we show that FPT-time algorithms for many smaller parameters are unlikely under standard parameterized complexity assumptions. Regarding problem-specific parameters, we observe that UNWEIGHTED FLEXIBLE GRAPH CONNECTIVITY admits an FPT-time algorithm when parameterized by the number of unsafe edges. Furthermore, we investigate a below-upper-bound parameter for the number of edges of a solution. We show that this parameter also leads to an FPT-time algorithm.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Graph algorithms analysis

Keywords and phrases Flexible graph connectivity, NP-hard problem, parameterized complexity, below-guarantee parameterization, treewidth

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.4

Related Version A continuously updated version of the paper is available at <https://arxiv.org/abs/2308.04575>.

Funding *Matthias Bentert*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).

Jannik Schestag: Supported by the German Academic Exchange Service (DAAD), project 57556279.

Frank Sommer: Supported by the Deutsche Forschungsgemeinschaft (DFG), project EAGR, KO 3669/6-1.

Acknowledgements This work was initiated at the research retreat of the Algorithmics and Computational Complexity group of TU Berlin held in Darlingerode in September 2022.



© Matthias Bentert, Jannik Schestag, and Frank Sommer;
licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 4; pp. 4:1–4:12

Leibniz International Proceedings in Informatics



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

1 Introduction

Computing a spanning tree is a fundamental task in computer science with a huge variety of applications in network design [14] and clustering problems [29]. In SPANNING TREE, one is given a graph G , and the aim is to find a set $T \subseteq E(G)$ of minimal size such that each pair of vertices in G is connected via edges in T . It is well known that SPANNING TREE can be solved in polynomial time [22, 25]. This classic spanning tree model has, however, a major drawback: all edges are seen as equal. In many scenarios, for example in the construction of supply chains [27], this is not sufficient: some connections might be more fragile than others. To overcome this issue, several different network-design and connectivity problems are studied with additional robustness constraints [13, 26]. In this work, we continue this line of research and investigate a graph model in which the edge set is partitioned into *safe* edges S and *unsafe* edges U [1, 2, 8]. In contrast to several other variants of robust connectivity, these two edge types enable us to model a non-uniform failure scenario [1]. With these types at hand, we can relax the model of a spanning tree in the sense that one unsafe edge may fail. An edge set T of a graph $G = (V, S, U)$ is an *unsafe spanning connected subgraph* if a) T is spanning for V , and b) $T - \{e\}$ is connected for each unsafe edge $e \in T$ [1]. This leads to the following problem:

UNWEIGHTED FLEXIBLE GRAPH CONNECTIVITY (UFGC)

Input: A graph $G = (V, S, U)$ and an integer k .

Question: Is there an unsafe spanning connected subgraph T of G with $|T| \leq k$?

In the following, we refer to T as a solution. UFGC generalizes several well-studied classic graph problems. Examples include 2-EDGE CONNECTED SPANNING SUBGRAPH [20, 26] ($S = \emptyset$) and HAMILTONIAN CYCLE [3, 19] ($S = \emptyset$ and $k = |V(G)|$). Thus, in sharp contrast to the classic SPANNING TREE problem, UFGC is NP-hard.

Related Work. If the graph is additionally equipped with an edge-cost function, the corresponding problem, in which one aims to find a solution of minimum total weight, is known as FLEXIBLE GRAPH CONNECTIVITY [1, 8]. FLEXIBLE GRAPH CONNECTIVITY is mainly studied in terms of approximation: Adjashvili et al. [1] provided a polynomial-time 2.527-approximation algorithm which was improved by Boyd et al. [8] to a polynomial-time 2-approximation. Recently, a generalization (p, q) -FLEXIBLE GRAPH CONNECTIVITY was introduced [8] and studied in terms of approximation. In this model, up to p unsafe edges may fail and the result shall be q -edge connected. Clearly, FLEXIBLE GRAPH CONNECTIVITY is the special case where $p = 1$ and $q = 1$. Boyd et al. [8] provided a $(q + 1)$ -approximation for the case $p = 1$ and a $\mathcal{O}(q \log(n))$ approximation for the general case. For the special case of $p = 2$, an improved approximation of $\mathcal{O}(q)$ was provided by Chekuri and Jain [9], and for the special case of $q = 1$ a $\mathcal{O}(1)$ approximation was shown by Bansal et al. [6].

Our model with safe and unsafe edges is based on prior work by e.g. Adjashvili et al., who studied the approximability of the classic (s, t) -PATH and (s, t) -FLOW problems in this model [2]. Even previous to Adjashvili et al. [2], some studies already indirectly investigated problems in this setting: There have been some studies of classic graph-connectivity problems where one wishes for the stronger requirement of 2-edge connectivity. In terms of our setting, this corresponds to the case that all edges are unsafe. One prominent example is the aforementioned 2-EDGE CONNECTED SPANNING SUBGRAPH [10, 21, 26]. There exists a $4/3$ -approximation [10, 21, 26], and an $6/5$ -approximation in cubic graphs [10]. Furthermore, for its generalization q -EDGE CONNECTED SPANNING SUBGRAPH approximation algorithms

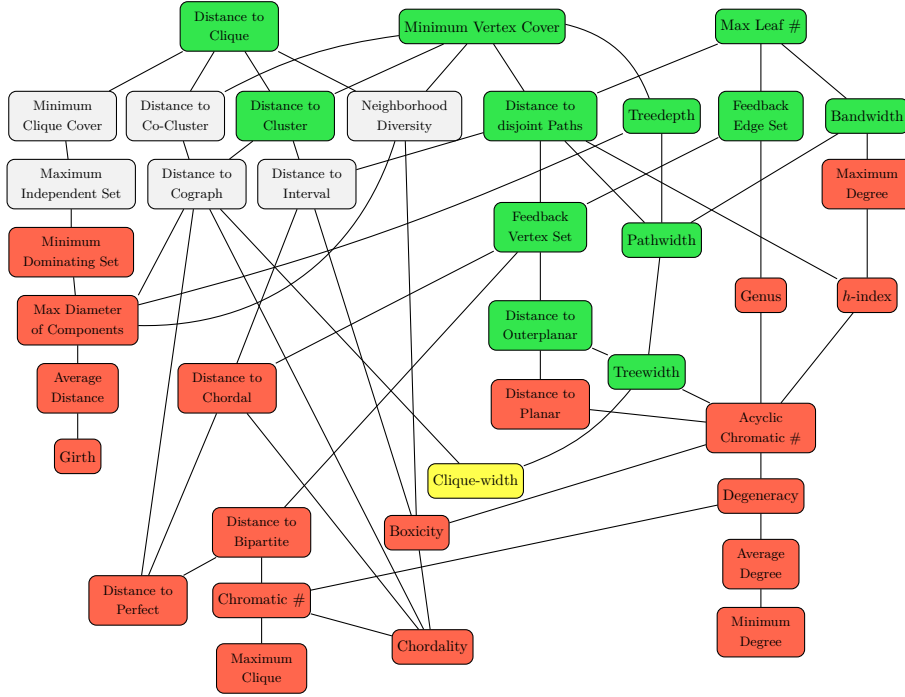
of ratio $1 + 1/(2q) + \mathcal{O}(1/(q^2))$ are known [17]. q -EDGE CONNECTED SPANNING SUBGRAPH is also studied in terms of parameterized complexity: Basavaraju et al. [4] provided an FPT-time algorithm with respect to the number of edges deleted from G to obtain a solution. A variant of 2-EDGE CONNECTED SPANNING SUBGRAPH in digraphs was studied by Bang-Jensen and Yeo [5]. They provided an FPT-time algorithm for a solution-size related parameter below an upper bound.

Our Results. We study the parameterized complexity of UFGC with respect to many natural parameters. We investigate problem specific and also structural graph parameters. A first idea to solve UFGC might be the following: merge each safe component into a single vertex by computing an arbitrary spanning tree of this component and then use existing algorithms for 2-EDGE CONNECTED SPANNING SUBGRAPH to solve the remaining instance. This approach, however, does not yield a minimal solution: Consider a C_4 on unsafe edges where we add one safe edge. Then, the unique minimal solution consists of all four unsafe edges. Hence, we need new techniques to solve UFGC optimally.

In terms of problem specific parameters, we first study parameterization by the number $|U|$ of unsafe edges. We provide an FPT-time algorithm for $|U|$ (Proposition 3.1), by exploiting the fact that DISJOINT SUBGRAPHS admits an FPT-time algorithm. Second, we investigate parameterizations above a lower bound and below an upper bound for the solution size. For the former parameterization, we obtain hardness due to the connection to HAMILTONIAN CYCLE and for the latter we obtain an FPT-time algorithm (Theorem 3.2). Our algorithm is inspired by an algorithm of Bang-Jensen and Yeo [5] for MINIMUM SPANNING STRONG SUBGRAPH. In this problem one is given a digraph D and one wants to find a minimum spanning strong digraph of D . The main technical hurdle in our adaption is that in UFGC we have two different edge types which have to be treated differently compared with the problem studied by Bang-Jensen and Yeo [5].

We also study parameterization by structural graph parameters; see Figure 1 towards a dichotomy for UFGC. For many parameters such as maximum degree and domination number we obtain para-NP-hardness, due to the connection to HAMILTONIAN CYCLE. For other parameters, we obtain FPT-time algorithms. For example, for the treewidth tw of the input graph and the (vertex) deletion distance to cluster graphs. With these two algorithms at hand, we obtain an almost complete border between parameters which allow for FPT-time algorithms and those that do not.

For the treewidth tw , we present an FPT-time algorithm based on dynamic programming with running time $n \cdot 2^{\mathcal{O}(\text{tw} \log(\text{tw}))}$ (Proposition 4.1). In order to achieve this running time, we define and exploit an encoding of size tw^{tw} to check all possibilities on how the current bag of the tree decomposition is connected with the already considered vertices. Finally, we show our main technical result: UFGC parameterized by the vertex-deletion distance to cluster graphs admits an FPT-time algorithm (Theorem 4.2). Therein, we use a combination of the following two main ingredients: First, given a modulator K , that is, a set of vertices such that $G' = G - K$ is a cluster graph, we can safely connect all vertices in K using vertices from $\mathcal{O}(|K|)$ cliques in G' in any solution. We call these cliques the *backbone* of the solution and we can guess in FPT time the structure of the backbone. Second, we show how to compute the size of a smallest solution that implements a backbone in FPT time using algorithms for both MAXIMUM BIPARTITE MATCHING and DISJOINT SUBGRAPHS in the process. Due to lack of space, several proofs (marked with (\star)) are deferred to the full version.



■ **Figure 1** The relations between structural graph parameters and our respective results for UFGC. A parameter k is marked green (●) if UFGC admits an FPT-time algorithm for k , yellow (●) if it is W[1]-hard with respect to k , and red (●) if it is NP-hard for constant k (para-NP-hard). We do not know the status for parameters with white boxes. An edge from a parameter α to a parameter β below α means that there is a function f such that $\beta \leq f(\alpha)$ in every graph. Hardness results for α imply the same hardness results for β and an FPT-time algorithm for β implies an FPT-time algorithm for α .

Preliminaries. For $n \in \mathbb{N}$, by $[n]$ we denote the set $\{1, \dots, n\}$. Throughout this work, all logarithms have 2 as their base. For a graph $G = (V, S, U)$, we denote by $V(G)$ and $E(G) := S \cup U$ its *vertex set* and *edge set*, respectively. Furthermore, by $n := |V(G)|$ we denote the number of vertices. Let $Z \subseteq V(G)$ be a vertex set. By $G[Z]$ we denote the *subgraph induced* by Z , and by $G - Z := G[V(G) \setminus Z]$ we denote the graph obtained by *removing* the vertices of Z . We denote by $N_G(Z) := \{y \in V(G) \setminus Z : \{y, z\} \in E(G), z \in Z\}$ and $N_G[Z] := N_G(Z) \cup Z$, the *open* and *closed neighborhood* of Z , respectively. For all these notations, whenever Z is a singleton $\{z\}$ we may write z instead of $\{z\}$. We may drop the subscript \cdot_G when it is clear from the context. Let $u, v \in V(G)$. We say that u and v are *safely connected* if there exists a path from u to v using only safe edges or if there exist two paths P_1 and P_2 from u to v such that $E(P_1) \cap E(P_2) \subseteq S$. We say that u and v are *unsafe connected* if u and v are not safely connected and if there exists a path from u to v . For more details on graph notation we refer to the standard monograph [28].

A parameterized problem is *fixed-parameter tractable* if every instance (I, k) can be solved in $f(k) \cdot |I|^{\mathcal{O}(1)}$ time for some computable function f . For more details on parameterized complexity, we refer to the standard monographs [11, 12].

2 Basic Observations

In this section, we explore the aforementioned connection between UFGC and HAMILTONIAN CYCLE and some implications thereof. To this end, note that if the graph contains no safe edges, then each vertex needs to be contained in at least one cycle in the solution. We show that a solution of size n to UFGC (if such a solution exists) corresponds to a Hamiltonian cycle.

► **Observation 2.1** (\star). *Let G be a graph and let $I := (G' = (V(G), \emptyset, E(G)), n)$. Then G contains a Hamiltonian cycle if and only if I is a yes-instance of UFGC.*

Since Observation 2.1 describes a polynomial-time reduction from HAMILTONIAN CYCLE to UFGC, we can directly transfer any NP-hardness results for HAMILTONIAN CYCLE on restricted graph classes to UFGC for the special case that there are no safe edges. It is known that HAMILTONIAN CYCLE remains NP-hard on subcubic bipartite planar graphs [3], on split graphs [19], and on graphs with constant maximum degree [18]. Moreover, HAMILTONIAN CYCLE is NP-hard for graphs with exactly one universal vertex, as shown next.

► **Observation 2.2** (\star). *HAMILTONIAN CYCLE is NP-hard even if there is exactly one universal vertex.*

We obtain the following simple corollary for UFGC.

► **Corollary 2.3.** *UFGC is NP-hard, even if there are no safe edges, k equals n , and the graph G a) is subcubic, planar and bipartite, b) is a split graph, c) has domination number one, or d) has constant maximum degree.*

Moreover, since HAMILTONIAN CYCLE is W[1]-hard parameterized by clique-width cw [15], and cannot be solved in $f(cw) \cdot n^{o(cw)}$ time [16], we obtain the following.

► **Corollary 2.4.** *UFGC is W[1]-hard with respect to the clique-width cw , even if $S = \emptyset$ and $k = n$. Moreover, this restricted version cannot be solved in $f(cw) \cdot n^{o(cw)}$ time.*

► **Observation 2.5** (\star). *UFGC is NP-hard even if the bisection width is one.*

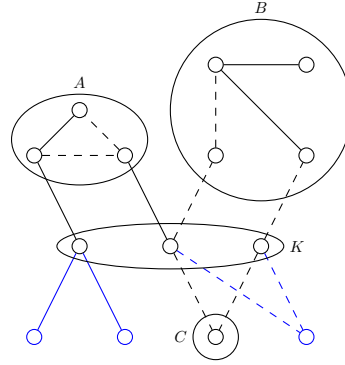
3 Problem-Specific Parameters

In this section, we study problem-specific parameters. In particular, we show that UFGC is fixed-parameter tractable when parameterized by the number of unsafe edges in the input graph and we study some *above-lower-bound* and *below-upper-bound* parameterizations for the solution size. Such parameters are frequently studied [23, 24]. However, since Observation 2.1 shows hardness for UFGC where $k = n$ and since each solution has at least $n - 1$ edges, the parameterization above the lower bound of $n - 1$ is hopeless. Hence, we focus on a parameterization below an upper bound. More precisely, we show that an optimal solution of G consists of at most $2n - 4$ edges. Then, we aim to find a solution for G with $k = 2n - 4 - \ell$ edges for small values of ℓ .

► **Proposition 3.1** (\star). *UFGC is fixed-parameter tractable when parameterized by $|U|$.*

We continue by presenting an FPT-time algorithm for the below lower bound parameter $\ell := 2n - 4 - k$ for UFGC.

► **Theorem 3.2** (\star). *UFGC can be solved in $\ell^{8\ell} \cdot \text{poly}(n)$ time, where $\ell := 2n - 4 - k$.*



■ **Figure 2** An example of a solution. Safe edges are depicted with solid lines and unsafe edges are depicted with dashed lines. The modulator K and three connecting components A , B , and C of a minimal backbone are drawn in black. All other vertices (drawn in blue) are not part of the minimal backbone as all vertices in K are already safely connected in the black subgraph. The connecting component A is a cyclic component and B and C are usual connecting components.

4 Structural Graph Parameters

In this section, we study two structural graph parameters. We start by giving an FPT-time algorithm for the parameter treewidth tw . Afterwards, we develop a far more intricate FPT-time algorithm for the parameter (vertex-deletion) distance to cluster graphs. Recall that a cluster graph is a graph in which each connected component is a clique.

► **Proposition 4.1** (\star). *UFGC parameterized by tw is solvable in $\mathcal{O}(n \cdot 2^{37 \text{tw} \log(\text{tw})})$ time.*

We continue with the algorithm for distance to cluster graphs.

► **Theorem 4.2.** *UFGC parameterized by the (vertex-deletion) distance ℓ to cluster graphs can be solved in $f(\ell) \cdot n^3$ time for some computable function f .*

Proof. We start by computing a set K of ℓ vertices in $\mathcal{O}(3^\ell \cdot n^3)$ time such that $G' = G[V \setminus K]$ is a cluster graph as follows. Note that a graph G is a cluster graph if and only if it does not contain an induced P_3 , that is, three vertices a, b , and c with $\{a, b\}, \{b, c\} \in E(G)$ and $\{a, c\} \notin E(G)$. Hence, we can find an induced P_3 in $\mathcal{O}(n^3)$ if it exists and then branch on which of the three vertices in the P_3 to include in K . Note that K needs to contain at least one of the three vertices and the resulting search tree has therefore depth ℓ and size 3^ℓ .

We next give a few definitions required to give a more detailed description of the algorithm afterwards. We call a subgraph of a solution $H = (V_H, S_H, U_H)$ a *backbone* if it contains all vertices in K and each pair of vertices in V_H is safely connected in H . See Figure 2 for an example. Given such a backbone $H = (V_H, S_H, U_H)$, a *connecting component* is a connected component in $H' = H[V_H \setminus K]$. We say that a backbone is minimal if the removal of any connecting component results in some pair of vertices in K being not safely connected anymore. We distinguish between two types of connecting components: *cyclic* and *usual*. A cyclic connecting component is a cycle (with some connections to vertices in K). Note that the number of edges in a cycle equals its number of vertices. A usual connecting component is a tree. Its number of edges is one less than its number of vertices but if it contains unsafe edges, then not all vertices are safely connected within the connecting component. Observe that we can indeed assume that each connecting component is either usual or cyclic as any connecting component that is not a tree contains at least as many edges as vertices. Hence,

we can replace such a connecting component by any cycle through all of its vertices. Such a cycle exists within any clique of size at least three (any permutation of the vertices results in a cycle and any connecting component inside a clique of size at most two is a tree) and any pair of vertices in a cyclic connecting component is safely connected.

Next, we distinguish between three types of cliques in G' . To this end, let C be a connected component in G' , that is, a clique in the cluster graph. The three types are based on the edges between C and K (note that all edges between vertices in C and the rest of the graph are to vertices in K) and on the connected components within C if we ignore all unsafe edges. We call such components *strong components*. A clique C is *strong*, if each strong component in C contains at least one vertex with a safe edge to a vertex in K . In this case, we can add C to the backbone using $|C|$ (safe) edges (a maximal spanning forest plus for each strong component one edge connecting it to K). The second type, we call *weak cliques*. A weak clique is not a strong clique but it is connected to K by a safe edge or by two unsafe edges with different endpoints in C . Since weak cliques are not strong cliques, there is some strong component in them, which is not connected to K via only safe edges. Hence, to safely connect this strong component L to K , we require at least $|L| + 1$ edges. Since we require at least $|C \setminus L|$ edges to connect the remaining vertices, we require at least $|C| + 1$ edges to safely connect C to the backbone. For weak cliques, $|C| + 1$ edges are sufficient as we can use a) a safe edge between C and K and a Hamiltonian cycle in C or b) two unsafe edges to different vertices in C and any Hamiltonian path between the two endpoints within C . We call the third type of clique *singletons*. A singleton is only connected to K by unsafe edges and all of these edges have the same endpoint in C . Note that if there is at most one unsafe edge between K and C (and assuming that $K \neq \emptyset$), then there cannot be a solution as C cannot be safely connected to K . We call them singletons because we can reduce such a clique to a single vertex in a preprocessing step. Since all connections to K are through one vertex $v \in C$, we have to safely connect all vertices in C to v . This can either be done via a spanning tree consisting only of safe edges (if such a tree exists) or via any Hamiltonian cycle otherwise¹. Which case applies can be checked in linear time by checking whether there is exactly one strong component in C .

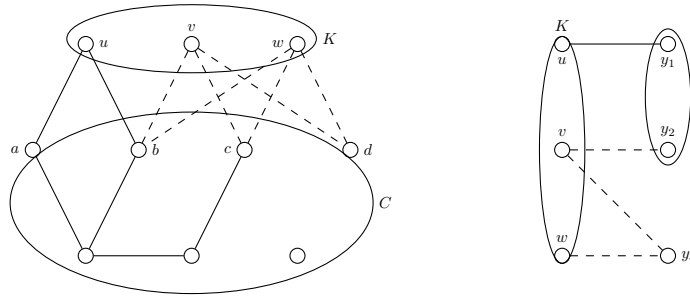
We are now in a position to describe the algorithm. First, we guess which edges between vertices in K belong to a solution T and the number p of connecting components in a minimal backbone H of T . Note that $p \leq 2\ell$ since any connecting component in H provides an (unsafe) connection between two vertices in K and any cycle of unsafe connections implies also safe connections. Hence, in the worst case each cycle is of length two and we require $2\ell - 2$ connections to implement a “safe spanning tree” between the vertices in K . Next, we guess the structure of H , that is, for each connecting component P in H , we guess the following (see also Figure 3).

1. Which vertices in K are adjacent to vertices in P in T ?
2. How large is the set Y of vertices in P that are neighbors to vertices in K in T ?²
3. Which edges between K and Y are contained in T ?³
4. Which pairs of vertices in Y are safely connected within Y ? (That is, a partition of the vertices in Y)
5. Is P a usual or a cyclic component?

¹ The Hamiltonian cycle exists if there are at least three vertices in C . If $|C| = 2$ and there is only an unsafe edge between the two vertices, then there cannot be a solution and we can return *no*.

² Note that $|Y| \leq 2\ell$ by the same argument that shows $p \leq 2\ell$.

³ Note that we something like “there are three vertices $y_1, y_2, y_3 \in Y$ and the solution contains the safe edge $\{u, y_1\}$ and the unsafe edges $\{v, y_2\}, \{v, y_3\}$, and $\{w, y_3\}$ ”. However, we do not guess which vertex in the input graph is a vertex in Y . For an example, we refer to Figure 3.



■ **Figure 3** The left side depicts the modulator K and one clique C in $G - K$. Safe edges are depicted with solid lines and unsafe edges are depicted with dashed lines. To reduce visual clutter, we do not show the unsafe edges between two vertices in C . The right side depicts one possible guess for a connecting component. It contains three vertices y_1, y_2 , and y_3 , some edges between K and a partition of the guessed vertices (in our case y_1 and y_2 are guessed to be safely connected within C). Note that in the graph on the left side there are many different possibilities to realize the guess on the right side. One possibility is $y_1 = a, y_2 = b$ and $y_3 = d$. A second possibility is $y_1 = b, y_2 = c$ and $y_3 = d$ and a third possibility is $y_1 = a, y_2 = c$ and $y_3 = b$.

Moreover, we guess which connecting components of the minimal backbone are contained in the same clique in G' , that is, we guess a partition of the p connecting components. Finally, we guess for each part of the partition the type of clique that contains the connecting components and how the rest of the clique (that is, all vertices in the clique that are not contained in any connecting component) is connected to the minimal backbone.

We distinguish between the following three types of connections. To this end, let C be a clique in G' that hosts at least one connecting component of H and let $C' = C \setminus V_H$ be the set of vertices that are not contained in a connecting component. If there is a connecting component in \mathcal{P} that contains at least one unsafe edge, then we can replace this edge with a Hamiltonian path through all vertices in C' . If this is the case or if $C' = \emptyset$, then we say that C' is *empty*. Otherwise, if we can safely connect all vertices in C' to the backbone using $|C'|$ edges, then we say that C' is *efficiently connected* to the backbone. This is the case if each vertex in C' is contained in a strong component in C that contains a) a vertex in some $P_i \in \mathcal{P}$ or b) a vertex with an incident safe edge to a vertex in K . If neither of the two cases above applies, then we require at least $|C'| + 1$ edges to connect the vertices in C' to the backbone. Note that in this case, we can always find a path through all vertices in C' and connect the two ends to any vertex in $C \setminus C'$. We call this type of connection *inefficient*.

Observe that if two solutions lead to exactly the same set of guesses, then they have the same number of edges⁴ as the difference between the number of edges and vertices in their minimal backbones and the types of the remaining cliques (both of cliques containing parts of the minimal backbone and those which do not) is the same for both solutions. As argued above, the difference between the number of edges and vertices in these cliques in a solution is completely determined by their type.

It remains to show how to check whether a guess leads to a solution and to analyze the running time. Towards the former, we first show how to test whether a given clique C of a guessed type can host a guessed set $\mathcal{P} = \{P_1, \dots, P_c\}$ of connecting components such that the rest of the clique has the guessed connection type. We can handle most combinations of clique type and connection type with a general approach. One special case has to be treated differently: The connection type is efficient, each connecting component $P_i \in \mathcal{P}$ is

⁴ Assuming that the solutions are minimal, that is, they do not contain edges whose removal yields a smaller solution.

a usual component which safely connects all respective vertices in Y and the clique C is a weak clique. First, we describe why this case is different from the others. Then, we show how to handle this special case and how to handle all other cases. The difference is that if the connection type is “ C' is empty” or inefficient, then we can pretty much ignore the connection type as we can always greedily find a solution independent of the connecting component. If one of the connecting components in \mathcal{P} is a cyclic component or a usual component with at least one unsafe edge in it, then the connection type is “ C' is empty”.⁵ The same is true if the clique C is a singleton. If C is a strong clique, then we can also ignore C' as we can always greedily find an efficient connection independent of the connecting component. Only if the connection type is efficient, \mathcal{P} only consists of usual components which safely connect all respective vertices in Y , and C is a weak clique, then we somehow need to “hit all strong components in C' ” which do not have a safe edge to a vertex in K using the connecting components.

The next step in this proof is to describe our algorithm to check if a clique C can host a set $\mathcal{P} = \{P_1, P_2, \dots, P_c\}$ of connecting components where each P_i is a usual connecting component consisting only of safe edges. Due to space constraints, we only show this algorithm in this extended abstract; the running time analysis is deferred to the full version. Informally speaking, we first show that each P_i is contained in a different strong component in C . We then use MAXIMUM BIPARTITE MATCHING to first check in which strong components V_j in C each P_i can be contained in. We then check whether we can assign each P_i to some V_j such that P_i can be contained in V_j and each strong component V_j which does not have a safe edge to a vertex in K contains some P_i .

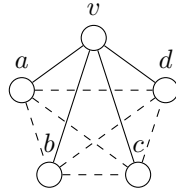
▷ Claim 4.3 (\star). We can check in $\mathcal{O}(\ell^2 \cdot n^3)$ time whether a clique C can host a set $\mathcal{P} = \{P_1, P_2, \dots, P_c\}$ of connecting components where each P_i is a usual connecting component consisting only of safe edges.

Next, we present the general algorithm for all remaining cases, that is, the connection type is not efficient, the clique C is not a weak clique, or the set of connecting components contains an unsafe edge. Here, we can ignore the connection type and we only need to check whether the guessed set \mathcal{P} of connecting components can be hosted in a clique C . Since C is a clique, any pair of vertices in C is connected by an edge. Hence, we can trivially connect any set of t vertices with unsafe connections using $t - 1$ edges. We only need to check two points:

1. Is there for each vertex in Y (the neighbors of vertices in K in C) a distinct vertex in C with all the required edges to vertices in K and
2. can all sets of vertices that are guessed to be pairwise connected via paths of safe edges inside C be connected in this way?

Note that the latter point is unfortunately not as simple as checking whether all vertices belong to the same strong component in C as the example in Figure 4 shows. We solve both points as follows. We built a graph with two vertices y, y' for each vertex $y \in Y$ and a vertex v for each vertex $v \in C$. There are edges $\{y, v\}$ and $\{y', v\}$ if the following holds. For each edge $\{x, y\}$ with $x \in K$ that is guessed to be in the solution, the edge $\{x, v\}$ is contained in the input graph G and the edge $\{x, v\}$ is safe if and only if the edge $\{x, y\}$ is guessed to be safe. Moreover, there is an edge between two vertices $u, v \in C$ if and only

⁵ We may assume that there is only one cyclic component in \mathcal{P} as we can always merge two cyclic components in one clique into one cyclic component. We need to consider the special case where the cyclic component contains exactly two vertices with edges to vertices in K as we need to ensure that there is at least one additional vertex not contained in any of the other connecting components in \mathcal{P} .



■ **Figure 4** A clique with five vertices is shown. Safe edges are depicted with solid lines and unsafe edges are depicted with dashed lines. Suppose we want to connect a to b and c to d using only safe edges. All vertices belong to the same strong component in C but the only solution is to include all four safe edges in the backbone. If each pair of vertices was connected via a safe edge, however, we could connect v with one of the terminal pairs and directly connect the other terminal pair with a safe edge. In this case we only used 3 edges. Intuitively, we were able to use one edge less by ignoring the connection between the two pairs of terminals. Since we assume our guess to be correct, we do not need to connect these terminals inside C as they will be connected via some paths outside of C . Thus, we want each set of vertices that are guessed to be pairwise connected via paths of safe edges inside C to form their own connected component when considering the graph induced by C in the solution.

if there is a safe edge between them in G . Let $\mathcal{R} = \{R_1, R_2, \dots, R_r\}$ with $R_i \subseteq Y$ be a partition of the vertices in Y according to which vertices are pairwise connected via paths of safe edges. Let $R'_i = \{y, y' \mid y \in R_i\}$ be the corresponding vertices in our constructed graph. As mentioned earlier, we need to consider the special case where one of the connecting components is a cyclic component with exactly two vertices in it as we need to ensure that there is at least one additional vertex that can be included in the cyclic component. In this case, we add two new vertices z, z' to the graph, connect each of them to all vertices v with $v \in C$, and define the set $R'_0 = \{z, z'\}$. We now solve DISJOINT CONNECTED SUBGRAPHS where each set R'_i is one terminal set. Next, we show that if there is a set of disjoint connected subgraphs each connecting the vertices in one set R'_i , then this corresponds to a solution to both points. Note that we may again assume that each vertex in C can only fill the role of one vertex in R_i and hence selecting any vertex adjacent to y in the solution gives us a matching between the vertices in C and the vertices in Y (any vertex $y \in R_i$ needs to be connected to at least y' and hence such a neighbor exists). Moreover, since the solution is a set of disjoint connected subgraphs and we only included safe edges between vertices in C , we are also ensured that this solution corresponds to a set of connecting components as guessed. Conversely, if there is a set of disjoint subgraphs in C that exactly correspond to our set of guesses, then this is also a solution to the instance of DISJOINT CONNECTED SUBGRAPHS if we additionally connect each pair of vertices y, y' to the respective vertex in C . Thus, we have found a way to check whether C can host a guessed set \mathcal{P} of connecting components.

After checking which cliques could potentially host each set \mathcal{P} of guessed connecting components, it remains to check whether all of these guesses can be fulfilled at the same time. To this end, we need to check whether all $q \leq p$ sets $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_q$ can be hosted each by a distinct clique. We do this using the textbook $\mathcal{O}(nm)$ -time algorithm for MAXIMUM BIPARTITE MATCHING as follows. We build a bipartite graph with one vertex v_i for each set \mathcal{P}_i and a vertex u_j for each connected component (clique) C_j in G' . There is an edge $\{v_i, u_j\}$ in the graph if and only if C_j can host \mathcal{P}_i . It then only remains to check whether there is a matching of size q as in this case each set \mathcal{P}_i is matched to a distinct clique in G' .

For space reasons, we defer the analysis of the running time to the full version.

▷ **Claim 4.4** (\star). The algorithm runs in time $f(\ell) \cdot n^3$ for some computable function f . ◀

We mention that each cluster graph is also a co-graph. Whether the parameter distance to co-graphs also allows for an FPT-time algorithm remains an open question.

5 Conclusion

In this work, we started an investigation into the parameterized complexity of UFGC. Our main results are FPT-time algorithms for a below-upper-bound parameter, the treewidth, and the vertex-deletion distance to cluster graphs, respectively. Moreover, we give a fairly comprehensive dichotomy between parameters that allow for FPT-time algorithms and those that lead to $W[1]$ -hardness (or in most cases even para-NP-hardness).

Nonetheless, several open questions remain. First, what is the status of the parameters that we were not able to resolve; for example does parameterization by the distance to cographs or interval graphs allow for an FPT-time algorithm? Second, is there an XP-time algorithm for UFGC parameterized by the clique-width. Third, it would be interesting to study the existence of polynomial kernels for the parameters that yield FPT-time algorithms. Unfortunately, some of these can be excluded due to the close relation to HAMILTONIAN CYCLE. In particular, HAMILTONIAN CYCLE (and hence also UFGC) does not admit a polynomial kernel with respect to the distance to outerplanar graphs unless $\text{coNP} \not\subseteq \text{NP/poly}$ [7]. Moreover, using the framework of AND-cross compositions, it is not hard to also exclude polynomial kernels for the parameters treedepth and bandwidth unless $\text{coNP} \not\subseteq \text{NP/poly}$. However, this still leaves quite a few parameters ready to be investigated in the future.

Last but not least, UFGC should only be regarded as a first step towards generalizing SPANNING TREE to more robust connectivity requirements. It is interesting to see whether (some of) our positive results can be lifted to the more general problem (p, q) -FLEXIBLE GRAPH CONNECTIVITY. Therein, the solution graph should still be q connected even if up to p unsafe edges fail. Also, one can study other problems in the setting where the edge set is partitioned into safe and unsafe edges from the lens of parameterized complexity; one possibility is (p, q) -FLEXIBLE (s, t) -PATH [2].

References

- 1 David Adjiashvili, Felix Hommelsheim, and Moritz Mühlenthaler. Flexible graph connectivity. *Mathematical Programming*, 192(1):409–441, 2022.
- 2 David Adjiashvili, Felix Hommelsheim, Moritz Mühlenthaler, and Oliver Schaudt. Fault-tolerant edge-disjoint $s - t$ paths - beyond uniform faults. In *Proceedings of the 18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT '22)*, volume 227 of *LIPICs*, pages 5:1–5:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- 3 Takanori Akiyama, Takao Nishizeki, and Nobuji Saito. NP-completeness of the hamiltonian cycle problem for bipartite graphs. *Journal of Information processing*, 3(2):73–76, 1980.
- 4 Jørgen Bang-Jensen, Manu Basavaraju, Kristine Vitting Klinkby, Pranabendu Misra, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized algorithms for survivable network design with uniform demands. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '18)*, pages 2838–2850. SIAM, 2018.
- 5 Jørgen Bang-Jensen and Anders Yeo. The minimum spanning strong subdigraph problem is fixed parameter tractable. *Discrete Applied Mathematics*, 156(15):2924–2929, 2008.
- 6 Ishan Bansal, Joseph Cheriyan, Logan Grout, and Sharat Irahimpur. Improved approximation algorithms by generalizing the primal-dual method beyond uncrossable functions. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP '23)*, volume 261 of *LIPICs*, pages 15:1–15:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- 7 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernel bounds for path and cycle problems. *Theoretical Computer Science*, 511:117–136, 2013.

- 8 Sylvia C. Boyd, Joseph Cheriyan, Arash Haddadan, and Sharat Ibrahimpur. Approximation algorithms for flexible graph connectivity. In *Proceedings of the 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '21)*, pages 9:1–9:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 9 Chandra Chekuri and Rhea Jain. Approximation algorithms for network design in non-uniform fault models. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP '23)*, volume 261 of *LIPICs*, pages 36:1–36:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- 10 Ali Çivril. A new approximation algorithm for the minimum 2-edge-connected spanning subgraph problem. *Theoretical Computer Science*, 943:121–130, 2023.
- 11 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshтанov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.
- 13 Andreas Emil Feldmann, Anish Mukherjee, and Erik Jan van Leeuwen. The parameterized complexity of the survivable network design problem. In *Proceedings of the 5th Symposium on Simplicity in Algorithms (SOSA '22)*, pages 37–56. SIAM, 2022.
- 14 Corinne Feremans, Martine Labbé, and Gilbert Laporte. Generalized network design problems. *European Journal of Operational Research*, 148(1):1–13, 2003.
- 15 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshтанov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010.
- 16 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshтанov, Saket Saurabh, and Meirav Zehavi. Clique-width III: hamiltonian cycle and the odd case of graph coloring. *ACM Transactions on Algorithms*, 15(1):9:1–9:27, 2019.
- 17 Harold N. Gabow and Suzanne Gallagher. Iterated rounding algorithms for the smallest k -edge connected spanning subgraph. *SIAM Journal on Computing*, 41(1):61–103, 2012.
- 18 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 19 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Academic Press, 1980.
- 20 Woonghee Tim Huh. Finding 2-edge connected spanning subgraphs. *Operations Research Letters*, 32(3):212–216, 2004.
- 21 Christoph Hunkenschroder, Santosh S. Vempala, and Adrian Vetta. A $4/3$ -approximation algorithm for the minimum 2-edge connected subgraph problem. *ACM Transactions on Algorithms*, 15(4):55:1–55:28, 2019.
- 22 Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.
- 23 Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: Maxsat and maxcut. *Journal of Algorithms*, 31(2):335–354, 1999.
- 24 Meena Mahajan, Venkatesh Raman, and Somnath Sikdar. Parameterizing above or below guaranteed values. *Journal of Computer and System Sciences*, 75(2):137–153, 2009.
- 25 Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.
- 26 András Sebő and Jens Vygen. Shorter tours by nicer ears: $7/5$ -approximation for the graph-tsp, $3/2$ for the path version, and $4/3$ for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014.
- 27 Lawrence V Snyder, Maria P Scaparra, Mark S Daskin, and Richard L Church. Planning for disruptions in supply chain networks. In *Models, methods, and applications for innovative decision making*, pages 234–257. Informs, 2006.
- 28 Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2000.
- 29 Ying Xu, Victor Olman, and Dong Xu. Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees. *Bioinformatics*, 18(4):536–545, 2002.

Difference Determines the Degree: Structural Kernelizations of Component Order Connectivity

Sriram Bhyravarapu ✉

The Institute of Mathematical Sciences, HBNI, Chennai, India

Satyabrata Jana ✉ 

The Institute of Mathematical Sciences, HBNI, Chennai, India

Saket Saurabh ✉ 

The Institute of Mathematical Sciences, HBNI, Chennai, India
University of Bergen, Norway

Roohani Sharma ✉ 

Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

We consider the question of polynomial kernelization of a generalization of the classical VERTEX COVER problem parameterized by a parameter that is provably smaller than the solution size. In particular, we focus on the c -COMPONENT ORDER CONNECTIVITY problem (c -COC) where given an undirected graph G and a non-negative integer t , the objective is to test whether there exists a set S of size at most t such that every component of $G - S$ contains at most c vertices. Such a set S is called a c -coc set. It is known that c -COC admits a kernel with $\mathcal{O}(ct)$ vertices. Observe that for $c = 1$, this corresponds to the VERTEX COVER problem.

We study the c -COMPONENT ORDER CONNECTIVITY problem parameterized by the size of a d -coc set (c -COC/ d -COC), where $c, d \in \mathbb{N}$ with $c \leq d$. In particular, the input is an undirected graph G , a positive integer t and a set M of at most k vertices of G , such that the size of each connected component in $G - M$ is at most d . The question is to find a set S of vertices of size at most t , such that the size of each connected component in $G - S$ is at most c . In this paper, we give a kernel for c -COC/ d -COC with $\mathcal{O}(k^{d-c+1})$ vertices and $\mathcal{O}(k^{d-c+2})$ edges. Our result exhibits that the difference in d and c , and not their absolute values, determines the exact degree of the polynomial in the kernel size.

When $c = d = 1$, the c -COC/ d -COC problem is exactly the VERTEX COVER problem parameterized by the solution size, which has a kernel with $\mathcal{O}(k)$ vertices and $\mathcal{O}(k^2)$ edges, and this is asymptotically tight [Dell & Melkebeek, JACM 2014]. We also show that the dependence of $d - c$ in the exponent of the kernel size cannot be avoided under reasonable complexity assumptions.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Kernelization, Component Order Connectivity, Vertex Cover, Structural Parameterizations

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.5

Funding *Sriram Bhyravarapu*: Supported by the SERB-DST via grant PDF/2021/003452.

Saket Saurabh: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416), and Swarnajayanti Fellowship (No. DST/SJF/MSA01/2017-18).

1 Introduction

The design of parameterized algorithms and kernelization has traditionally relied on the size of the solution as a crucial parameter. Nonetheless, when a problem is established as fixed-parameter tractable based on the solution size, it becomes natural to explore the problem using a parameter that is provably smaller than the solution size.



© Sriram Bhyravarapu, Satyabrata Jana, Saket Saurabh, and Roohani Sharma; licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 5; pp. 5:1–5:14

Leibniz International Proceedings in Informatics



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

Over the past decade, several interesting papers have explored these inquiries, particularly in the realm of kernelization [2, 7, 9–12, 15]. Notable contributions in this area include polynomial kernels for the VERTEX COVER problem parameterized by the *feedback vertex set* [10] and the *odd cycle transversal* [12]. Hols, Kratsch, and Pieterse [9] provide a comprehensive perspective on most of the aforementioned structural kernelization of VERTEX COVER. Additionally, kernelization of VERTEX COVER with respect to above-guarantee parameters has also been studied [11]. More recently, in another direction of work, Bougeret, Jansen & Sau, gave a characterization for which structural parameters, that serve as modulators of minor-closed graph classes, VERTEX COVER admits polynomial kernels [1].

In this paper, we consider a generalized version of the VERTEX COVER problem known as the c -COMPONENT ORDER CONNECTIVITY (c -COC) problem. In the c -COC problem, we are given a graph G and an integer t , and the objective is to identify a set of at most t vertices, say S , such that the size of each connected component of $G - S$ is at most c . Such a set S is referred to as a c -coc set. It is worth noting that when c equals 1, the c -COC problem is equivalent to the VERTEX COVER problem. The current best-known kernel for the VERTEX COVER problem parametrized by solution size (t) consists of $2t - c \log t$ vertices [14] for all $c > 0$. Although previously there was a kernel for VERTEX COVER with $\mathcal{O}(t)$ vertices and $\mathcal{O}(t^2)$ edges [4] which is asymptotically best. For c -COC we can obtain a simple kernel with $\mathcal{O}((t + c)t)$ vertices by deleting vertices of degree at least $t + c$, iteratively. Kumar and Lokshtanov [13] designed a kernel with $2ct$ vertices running in time $n^{\mathcal{O}(c)}$. Finally, Xiao [18] obtained a kernel with $9ct$ vertices running in time $n^{\mathcal{O}(1)}$. Here, the polynomial in the running time does not depend on c .

Observe that when $d \geq c$, the size of a d -coc set is at most the size of a c -coc set. This observation leads us to a natural hierarchy of parameterized problems known as c -COC parameterized by a d -coc set, where c and d are positive integers satisfying $c \leq d$. We refer to these parameterized problems as c -COC/ d -COC.

c -COC/ d -COC

Parameter: k

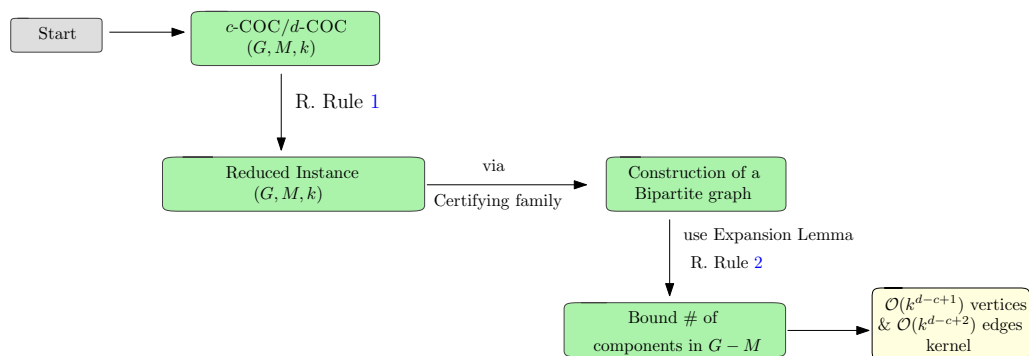
Input: An undirected graph G , an integer t , a set $M \subseteq V(G)$ such that $|M| \leq k$ and for each connected component C of $G - M$, $|C| \leq d$

Question: Does there exist a set $S \subseteq V(G)$ such that $|S| \leq t$ and for each connected component C' of $G - S$, $|C'| \leq c$?

It is natural to ask how do we get the modulator. Either, we can assume that it is given as part of the input or we can obtain a set M of size at most $(d + 1)\text{opt}$, where opt is the size of a smallest d -coc set. Indeed, start with an empty M , and while M is not a d -coc set, greedily select a arbitrary connected subgraph with $d + 1$ vertices and include each of these $d + 1$ vertices into M and remove them from the graph. Thus, from now onwards we assume that M is given as part of the input.

Our main result shows that c -COC/ d -COC admits a polynomial kernel with $\mathcal{O}(k^{d-c+1})$ vertices and $\mathcal{O}(k^{d-c+2})$ edges, where k is the size of M , a d -coc set. Notably, our result establishes that the degree of the polynomial in the kernel size is solely determined by the difference between d and c , rather than the specific values of d and c . To illustrate, both 5-COC/7-COC and 23-COC/25-COC exhibit kernels of size $\mathcal{O}(k^3)$. The formal statement of our main result is presented in Theorem 1.

► **Theorem 1.** c -COC/ d -COC admits a kernel with $\mathcal{O}(k^{d-c+1})$ vertices and $\mathcal{O}(k^{d-c+2})$ edges.



■ **Figure 1** A summary of the main steps of our kernelization.

Note that when $c = d = 1$, the c -COC/ d -COC problem corresponds to the VERTEX COVER problem parameterized by the size of the solution. In this scenario, our result is asymptotically consistent with the best-known bounds for VERTEX COVER.

In the light of Theorem 1, the subsequent question arises as to whether the exponent of k in the kernel size can be made a constant value and be independent of $d - c$. Specifically, does c -COC/ d -COC admit a uniform kernel of size $f(d, c)k^{\mathcal{O}(1)}$, where f is a function that only depends on d and c ? However, we demonstrate that this is not possible. In particular, we establish that VERTEX COVER/ d -COC does not admit a kernel of size $\mathcal{O}(k^{d-\epsilon})$ for any $\epsilon > 0$ and positive integer d . This result is precisely formalized in Theorem 2. VERTEX COVER therefore does not admit a kernel that is uniformly polynomial in the value of d . The phenomenon that the degree of the kernel size for VERTEX COVER has to increase when using smaller and smaller structural parameterization is well-known [8].

► **Theorem 2.** *For every $\epsilon > 0$ and every positive integer d , VERTEX COVER/ d -COC has no compression of vertex size $\mathcal{O}(k^{d-\epsilon})$ unless $\text{co-NP} \subseteq \text{NP/poly}$.*

Our methods

In order to construct a kernel for c -COC/ d -COC, our algorithm employs the Expansion Lemma, a combinatorial tool that played a crucial role in developing a quadratic kernel for the FEEDBACK VERTEX SET problem. Given an input instance (G, M, k, t) of c -COC/ d -COC, we generate sets of “certifying families” for every subset $T \subseteq M$ that correspond to certain components in $G - M$. In particular, the idea is to understand the following. Suppose we do not include any vertex from T in our solution. Then for which components C of $G - M$ do we need to select a strictly larger number of vertices than what is required to *locally* solve the problem in C . These components (in fact, a subset of these) are part of a certifying family corresponding to T . By utilizing these certifying families, we construct a bipartite graph and apply the Expansion Lemma to identify an irrelevant component in $G - M$. Through repeated applications of the Expansion Lemma, we can upper bound the number of components in $G - M$ by $\mathcal{O}(k^{d-c+1})$. Since the size of M is at most k , and each component in $G - M$ contains at most d vertices, we can bound the number of vertices in the kernel to $\mathcal{O}(d \cdot k^{d-c+1} + k)$ and the number of edges to $\mathcal{O}(d^2 \cdot k^{d-c+2} + k^2)$. A summary of the key steps in our kernelization algorithm is provided in Figure 1. Moreover, our lower bound results are established through a parameter-preserving reduction from the d -SAT problem.

2 Preliminaries

In this paper, we consider finite, undirected graphs. For a graph G , we use $V(G)$ and $E(G)$ to refer to its vertex and edge sets, respectively. By $|G|$ we denote the number of vertices in G . We use $\text{comp}(G)$ to denote the size of the largest component in G , defined as $\text{comp}(G) = \max\{|V(C)| : C \text{ is a component of } G\}$. Thus, an n -vertex graph G is connected if and only if $\text{comp}(G) = n$. Given two vertex-disjoint sets X and Y from $V(G)$, the set $N_X(Y) = N(Y) \cap X$ represents the subset of vertices in X that has at least one neighbour in Y . For any positive integer ℓ and a subgraph $H \subseteq G$, the ℓ -component order connectivity of H , denoted as $\text{coc}_\ell(H)$, is defined as the size of the minimum set $X \subseteq V(H)$ such that $\text{comp}(H - X) \leq \ell$. In other words, we have $\text{coc}_\ell(H) = \min\{|X| : \text{comp}(H - X) \leq \ell, X \subseteq V(H)\}$. We use $[q]$ to denote the set $\{1, \dots, q\}$. For details on parameterized complexity, kernelization, and compression we refer to the textbooks [3] and [6].

3 Kernelization

We represent an instance of c -COC/ d -COC as (G, M, k, t) , where G is a graph, $M \subseteq V(G)$ is a subset of vertices with size at most k , and $\text{comp}(G - M) \leq d$. Recall that the problem seeks to determine whether there exists a set $S \subseteq V(G)$ of size at most t such that $\text{comp}(G - S) \leq c$. We use \mathcal{C} to denote the set of all components in $G - M$. For any component C in \mathcal{C} , since $|C| \leq d$, we have $\text{coc}_c(C) \leq d - c$, where $\text{coc}_c(C)$ denotes the size of the smallest vertex set $X \subseteq V(C)$ such that $\text{comp}(C - X) \leq c$. Let $\mathcal{C}_\ell = \{C \in \mathcal{C} \mid \text{coc}_c(C) = \ell\}$ denote the set of components C in $G - M$ for which the size of a smallest c -coc is ℓ . It is possible for the set \mathcal{C}_ℓ to be empty. Note that for each $C \in \mathcal{C}$, $\text{coc}_c(C) \leq d - c$. Consequently, for each $\ell > d - c$, we have $\mathcal{C}_\ell = \emptyset$. In the subsequent section, we show that the number of components in \mathcal{C}_ℓ , for any $\ell \in \{0, 1, \dots, d - c\}$, can be upper bounded by $\mathcal{O}(k^{d-c+1})$ (after the application of certain reduction rules). Once this is accomplished the bounds on the number of vertices and edges in the kernel follow immediately, as each component has at most d vertices. Hence, in the remaining we focus on bounding the size of each set \mathcal{C}_ℓ .

Note that the family \mathcal{C}_ℓ can be constructed in polynomial time. Indeed, for each component C of $G - M$, the value of $\text{coc}_c(C)$ can be computed in $2^{|C|} \cdot |C|^{\mathcal{O}(1)}$ time by considering all possible subsets of C as c -coc sets. Given that $|C| \leq d$ for each connected component C , the computation of $\text{coc}_c(C)$ can be done in time that only depends on d (which is a constant). Recall that (G, M, k, t) represents an instance of c -COC/ d -COC. Consider a vertex set $T \subseteq M$ and a component $C \in \mathcal{C}_\ell$. We use $\text{local}(T, C)$ to denote the size of the smallest set $X \subseteq V(C)$ such that $\text{comp}(C - X) \leq c$ and $N_C(T) \subseteq X$, where $N_C(T) = N(T) \cap V(C)$. Informally, $\text{local}(T, C)$ represents the size of the smallest solution corresponding to $\text{coc}_c(C)$ in $G[C]$ that must include all the neighbors of T in C . Notably, for any pair T and C , the value of $\text{local}(T, C)$ can be computed in $2^{|C|} \cdot |C|^{\mathcal{O}(1)}$ time by examining all subsets of C that are supersets of the neighborhood of T in C , considering them as solution sets of $G[C]$, and determining the minimum possible set among them. Since $\text{coc}_c(C) = \ell$ for each component $C \in \mathcal{C}_\ell$, we make the following observation.

► **Observation 3.** *For each pair (T, C) where $T \subseteq M$ and $C \in \mathcal{C}_\ell$, we have $\text{local}(T, C) \geq \ell$.*

Certifying family for a fixed ℓ . Let $\mathcal{T} = \{T \mid T \subseteq M \text{ and } |T| \leq \ell + 1\}$ denote the set of all subsets of M of size at most $(\ell + 1)$. We refer to each $T \in \mathcal{T}$ as an *unordered tuple* of size $|T|$. For every $T \in \mathcal{T}$, we define a set of components \mathcal{F}_T associated with T as

$\mathcal{F}_T = \{C : C \in \mathcal{C}_\ell, \text{local}(T, C) > \ell\}$. We refer to such a family \mathcal{F}_T as a *certifying family* for T . Essentially, if $C \in \mathcal{F}_T$, then there exists *no solution* of $G[C]$ corresponding to $\text{coc}_c(C)$ of size ℓ that includes all the neighbors of T in C .

Given two disjoint vertex sets V_1 and V_2 , the boundary of V_1 with respect to V_2 , denoted by $\text{bdry}_{V_2}(V_1)$, is defined as the set $N(V_2) \cap V_1$. We present two key lemmas that are crucial for our analysis.

► **Lemma 4.** *Let $C \in \mathcal{C}_\ell$ such that $|\text{bdry}_M(C)| \geq \ell + 1$. Then there exists $T \in \mathcal{T}$ satisfying $C \in \mathcal{F}_T$.*

Proof. Since $C \in \mathcal{C}_\ell$, an optimal c -coc set in $G[C]$ has size ℓ but each c -coc set containing $N_G(T) \cap C$ has size more than ℓ because $|N_C(T)| > \ell$. Now consider an arbitrary set $C^* \subseteq \text{bdry}_M(C)$ of size $(\ell + 1)$. Let $U \subseteq N_M(C)$ be a set containing a neighbor of each vertex in C^* (arbitrarily select a neighbor of each vertex in C^*). Clearly, the size of $|U| \leq \ell + 1$ and $C^* \subseteq N_C(U)$. According to the definition of a certifying family, the component C is associated with U .

This concludes the proof. ◀

► **Lemma 5.** *Let $C \in \mathcal{C}_\ell$ be a component with the property that $|\text{bdry}_M(C)| \leq \ell$. Then either there exists a tuple $T \in \mathcal{T}$ such that $C \in \mathcal{F}_T$, or there exists a vertex set $U \subseteq V(C)$ with $|U| = \ell$, satisfying $\text{bdry}_M(C) \subseteq U$ and $\text{comp}(C - U) \leq c$.*

Proof. Consider the vertex set $N_M(C) \subseteq M$. Let $X \subseteq N_M(C)$ be a set containing a neighbor of each vertex in $\text{bdry}_M(C)$ (arbitrarily select a neighbor of each vertex in $\text{bdry}_M(C)$). Clearly, the size of $|X| \leq |\text{bdry}_M(C)| \leq \ell$. Furthermore, we have $N_C(X) = \text{bdry}_M(C)$.

If C belongs to the certifying family \mathcal{F}_X , then our assertion is proven. So we assume that $C \notin \mathcal{F}_X$. According to the definition of \mathcal{F}_X , this implies $\text{local}(X, C) = \ell$. Therefore, there exists a solution U associated with $\text{local}(X, C)$ such that $U \subseteq V(C)$, $|U| = \ell$, $\text{bdry}_M(C) \subseteq U$, and $\text{comp}(C - U) \leq c$. ◀

Lemmas 4 and 5, essentially, say that a component C of $G - M$ is not in any certifying family if there exists a minimum size local solution for the component C that contains all the boundary vertices ($\text{bdry}_M(C)$). This observation leads to the following reduction rule.

► **Reduction Rule 1.** *Consider a component $C \in \mathcal{C}_\ell$ for which there is no tuple $T \in \mathcal{T}$ that satisfies that $C \in \mathcal{F}_T$. Then we remove C from the graph G and reduce the value of t by ℓ . The resulting instance is $(G - C, M, k, t - \ell)$.*

To apply Reduction Rule 1 finding such a component takes $k^{\ell+1} \cdot 2^d \cdot n^{\mathcal{O}(1)}$ time. The correctness of the Reduction Rule 1 follows from the following Lemma 6.

► **Lemma 6.** *Reduction Rule 1 is safe.*

Proof. The forward direction is straightforward. Let S be a solution to the instance (G, M, k, t) . Since $G - C$ is a subgraph of G and $\text{coc}_c(C) = \ell$, there are at least ℓ vertices of C in S . Therefore, $S \setminus C$ is a solution for $(G - C, M, k, t - \ell)$.

In the backward direction, let S_1 be a solution to $(G - C, M, k, t - \ell)$. We aim to show that there exists a vertex set $Z \subseteq V(C)$ such that $|Z| = \ell$ and $S \cup Z$ is a solution to (G, M, k, t) . We consider two cases based on the size of $\text{bdry}_{M \setminus S_1}(C)$.

Case 1. $\text{bdry}_{M \setminus S_1}(C) \geq \ell + 1$. Based on Lemma 4, there exists $T \in \mathcal{T}$ such that $T \subseteq M$, $|T| \leq (\ell + 1)$, and $C \in \mathcal{F}_T$. This contradicts our assumption that there is no $T \in \mathcal{T}$ satisfying $C \in \mathcal{F}_T$.

Case 2. $\text{bdry}_{M \setminus S_1}(C) \leq \ell$. Based on Lemma 5, we have two possibilities: either there exists $T \in \mathcal{T}$ such that $|T| \subseteq M \setminus S_1$, $|T| \leq (\ell + 1)$, and $C \in \mathcal{F}_T$, or there is a vertex set $U \subseteq V(C)$ satisfying $|U| = \ell$, $\text{bdry}_{M \setminus S_1}(C) \subseteq U$, and $\text{comp}(C - U) \leq c$. However, based on our assumption, the former condition cannot occur. Therefore, there must exist a vertex set $U \subseteq V(C)$ satisfying $|U| = \ell$, $\text{bdry}_{M \setminus S_1}(C) \subseteq U$, and $\text{comp}(C - U) \leq c$. In this case, we set $Z = U$.

This completes the proof. ◀

Expansion Lemma. From now onwards, we assume that we have an instance (G, M, k, t) of c -COC/ d -COC, on which we have applied Reduction Rule 1 exhaustively. Now, we bound the number of components in \mathcal{C}_ℓ using the *expansion lemma* in strengthened form of [16]. Let us first recall the definition of expansion and the expansion lemma.

► **Definition 7** (*q-expansion* [3]). Let H be a bipartite graph with vertex bipartition (X, Y) and q be a positive integer. A set of edges $E^* \subseteq E(H)$ is called a *q-expansion of X into Y* if (i) each vertex of X is incident with exactly q edges of E^* , and (ii) E^* saturates exactly $q|X|$ vertices in Y .

► **Lemma 8** (Expansion Lemma [[3], Lemma 2.18]). *Let $q \in \mathbb{N}$ and G be a bipartite graph with vertex bipartition (P, Q) such that $|Q| > q \cdot |P|$ and there are no isolated vertices in Q . Then there exist nonempty vertex sets $X \subseteq P$ and $Y \subseteq Q$ such that (i) X has a q -expansion E^* into Y , (ii) no vertex in Y has a neighbor outside X . Furthermore, two such sets X and Y and such vertex w can be found in the time that is polynomial in the size of G .*

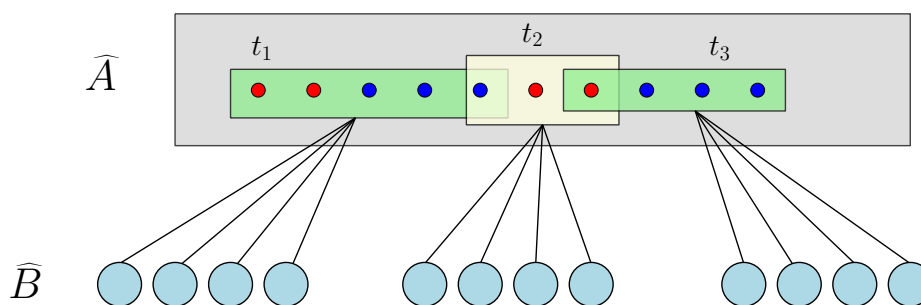
Next we mention q -Expansion Lemma given by Fomin et al. [5] which is a generalization of a result due to Thomassé [[17], Theorem 2.3].

► **Lemma 9** (The q -Expansion Lemma [[5], Lemma 5.1]). *Let $q \in \mathbb{N}$ and G be a bipartite graph with vertex bipartition (P, Q) such that $|Q| > q \cdot t$, where t is the size of a maximum matching in G , and there are no isolated vertices in Q . Then there exist nonempty vertex sets $X \subseteq P$ and $Y \subseteq Q$ such that (i) X has a q -expansion E^* into Y , (ii) no vertex in Y has a neighbor outside X . Furthermore, two such sets X and Y and such vertex w can be found in the time that is polynomial in the size of G .*

For our purpose we use the expansion lemma in strengthened form given by Philip et al. [16] which is following.

► **Lemma 10** (Strong q -Expansion Lemma [[16], Lemma 5]). *Let $q \in \mathbb{N}$ and G be a bipartite graph with vertex bipartition (P, Q) such that $|Q| > q \cdot t$, where t is the size of a maximum matching in G , and there are no isolated vertices in Q . Then there exist nonempty vertex sets $X \subseteq P$ and $Y \subseteq Q$ such that (i) X has a q -expansion E^* into Y , (ii) no vertex in Y has a neighbor outside X , and (iii) there is a vertex $w \in Y$ such that w is not incident to any edge in E^* (or, E^* does not saturate w). Furthermore, two such sets X and Y and such vertex w can be found in the time that is polynomial in the size of G .*

Note that the statement of Lemma 10 remains valid even for $|Q| > q \cdot |P|$, as $|P| \geq t$. Now, in order to apply the expansion lemma, we first construct an auxiliary bipartite graph where this lemma is applied.



■ **Figure 2** An example of 4-expansion from \widehat{A} into \widehat{B} . t_1, t_2, t_3 represents vertices of corresponding tuples in $\mathcal{T}_{\widehat{A}}$. The red-colored vertices denote the solution vertices from the modulator.

Construction of an auxiliary bipartite graph $H = (A, B)$. Let us recall the set \mathcal{T} , which consists of all subsets of M with size at most $\ell + 1$, and the corresponding certifying families $\{\mathcal{F}_T : T \in \mathcal{T}\}$. We will now construct a bipartite graph H with vertex partitions A and B using the following procedure:

- For each tuple $T_i \in \mathcal{T}$, we introduce a vertex t_i in the part A .
- For each component $C_j \in \mathcal{C}_\ell$, we include a vertex c_j in the part B .
- For each pair of vertices $t_i \in A$ and $c_j \in B$, we add an edge $t_i c_j$ in H if and only if C_j belongs to the certifying family \mathcal{F}_{T_i} .

We are now ready to give the reduction rule. From this point onwards, we fix the following value for q .

$$q = (\ell + 2) + c(\ell + 1)$$

Consider the bipartite graph $H = (A, B)$ that was constructed above. It is important to note that if the instance (G, M, k, t) is reduced using Reduction Rule 1, then there are no isolated vertices in the vertex set B .

► **Reduction Rule 2.** If $|B| > q \cdot |A|$, then call the algorithm provided by the Expansion Lemma to compute sets $\widehat{A} \subseteq A$ and $\widehat{B} \subseteq B$ such that

- no vertex in \widehat{B} has a neighbor outside \widehat{A} , i.e., $N(\widehat{B}) \subseteq \widehat{A}$,
- there is a q -expansion \widehat{E} from \widehat{A} into \widehat{B} , and
- there is a vertex $b \in \widehat{B}$ such that b is not incident with \widehat{E} .

Consider the component $C \in \mathcal{C}_\ell$ corresponding to the vertex b in B . Then we remove C from the graph G and reduce the value of t by ℓ . The resulting instance is $(G - C, M, k, t - \ell)$.

Before analyzing the safeness of Reduction Rule 2, we look at the following lemma.

► **Lemma 11.** Suppose \widehat{E} represents a q -expansion from \widehat{A} into \widehat{B} , and let b be a vertex in \widehat{B} that satisfies the condition of Reduction Rule 2. Further, let S_1 be a solution to the problem $(G - C, M, k, t - \ell)$. Then, there exists another solution S_2 of $(G - C, M, k, t - \ell)$ that satisfies the following properties: (i) $|S_2| \leq |S_1|$; (ii) for each vertex t_i in \widehat{A} , S_2 intersects with the corresponding vertex set $T_i \subseteq M$, that is $S_2 \cap T_i \neq \emptyset$.

Proof. Let $\mathcal{T}_{\widehat{A}}$ be the set of tuples corresponding to the vertices in \widehat{A} . Let \mathcal{T}_1 be a subset of $\mathcal{T}_{\widehat{A}}$, containing those tuples T for which T intersects with S_1 (i.e., $T \cap S_1 \neq \emptyset$). On the other hand, let \mathcal{T}_2 be defined as the set of tuples in $\mathcal{T}_{\widehat{A}}$ that are not in \mathcal{T}_1 , i.e., $\mathcal{T}_2 = \mathcal{T}_{\widehat{A}} \setminus \mathcal{T}_1$. In other words, \mathcal{T}_2 comprises all the tuples T from $\mathcal{T}_{\widehat{A}}$ that satisfy $T \cap S_1 = \emptyset$.

For each vertex t_i in \widehat{A} that corresponds to a tuple in the set \mathcal{T}_2 , we define two sets: \mathcal{C}_{1,t_i} and \mathcal{C}_{2,t_i} . The set \mathcal{C}_{1,t_i} consists of each component C_j in \mathcal{C}_ℓ such that $t_i c_j$ is an edge in \widehat{E} and $|V(C_j) \cap S_1| = \ell$. The set \mathcal{C}_{2,t_i} consists of each component C_j in \mathcal{C}_ℓ such that $t_i c_j$ is an edge in \widehat{E} and $|V(C_j) \cap S_1| \geq \ell + 1$. For an illustration see Figure 2.

First we show that $|\mathcal{C}_{1,t_i}| < c(\ell + 1)$. On the contrary, assume that $|\mathcal{C}_{1,t_i}| \geq c(\ell + 1)$. For any component C_j belonging to \mathcal{C}_{1,t_i} , we observe that $c_j t_i \in \widehat{E}$ implies $C_j \in \mathcal{F}_{T_i}$. Consequently, it follows that $\text{local}(T_i, C_j) > \ell$. However, since $C_j \in \mathcal{C}_{1,t_i}$, we have $|C_j \cap S_1| = \ell$. Hence, we conclude that $N_{C_j}(T_i) \setminus S_1 \neq \emptyset$. Thus there always exists a vertex $x \in T_i$ and $y \in (N_{C_j}(T_i) \setminus S_1)$ such that $xy \in E(G)$. As $|\mathcal{C}_{1,t_i}| \geq c(\ell + 1)$, we can select a vertex each from each of the components in \mathcal{C}_{1,t_i} and obtain a set Y containing $c(\ell + 1)$ vertices from $G - M$, where $Y \cap S_1 = \emptyset$ and each vertex $y \in Y$ has a neighbor in T_1 . Since, $|T_i| \leq \ell + 1$ (since $T_i \in \mathcal{T}$), and $T_i \cap S_1 = \emptyset$, we can deduce, by applying the pigeon-hole principle, the existence of a vertex x in T_i such that $\deg_{G-C-S_1}(x) \geq c$. Consequently, we have a component of size at least $c + 1$ in $G - S_1$. This contradicts the fact that S_1 is a solution for the c -COC/ d -COC problem on the instance $(G - C, M, k, t - \ell)$. Therefore, we can conclude that $|\mathcal{C}_{1,t_i}| \leq c(\ell + 1)$.

Given that $q = (\ell + 2) + c(\ell + 1)$, and $|\mathcal{C}_{1,t_i}| \leq c(\ell + 1)$, we can deduce that $|\mathcal{C}_{2,t_i}| \geq \ell + 2$. We denote the set of all vertices contained in some tuple in \mathcal{T}_2 as $V(\mathcal{T}_2)$, defined formally as $V(\mathcal{T}_2) := \{v \mid \exists T \in \mathcal{T}_2 : v \in M \cap T\}$. Now, we propose a new solution denoted as S_2 . Let $U = \bigcup_i V(\mathcal{C}_{2,t_i})$ and $\mathcal{C}' = \bigcup_i \mathcal{C}_{2,t_i}$.

$$S_2 = (S_1 \setminus U) \bigcup V(\mathcal{T}_2) \bigcup_{C_j \in \mathcal{C}'} Z_j,$$

Here, $Z_j \subseteq V(C_j)$ represents a set (the exact choice of Z_j is deferred to later in the proof) that corresponds to $\text{coc}_c(C_j)$. Here we want to mention that we do not want to choose an arbitrary ℓ -size coc in $G[C_j]$: rather we want to choose one that contains the neighborhood of $(M \setminus (S_1 \cup V(\mathcal{T}_2)))$ in C_j . We define a set X to *correspond to* $\text{coc}_c(H)$ if $|X| = \text{coc}_c(H)$ and $\text{comp}(H - X) \leq c$. Recall that $\text{coc}_c(C_j) = \ell$.

Towards the proof, we need to establish two conditions: firstly, $|S_2| \leq |S_1|$, and secondly, that S_2 is a solution of $(G - C, M, k, t - \ell)$.

- (i) $|S_2| \leq |S_1|$. In this comparison, we are examining the sizes of S_1 and S_2 . Observe that we are only editing (deleting or adding) vertices that appear in the tuples in \mathcal{T}_2 and the components \mathcal{C}_{2,t_i} where t_i corresponds to a specific tuple T_i in \mathcal{T}_2 . Let us define the size of the solution outside \mathcal{T}_2 and \mathcal{C}_{2,t_i} as f . We also define r as the sum of the sizes of \mathcal{C}_{2,t_i} for all relevant tuples $T_i \in \mathcal{T}_2$, i.e., $r = \sum_i |\mathcal{C}_{2,t_i}|$. Since $|\mathcal{C}_{2,t_i}| \geq \ell + 2$, we can conclude that $r \geq (\ell + 2) \cdot |\mathcal{T}_2|$. Based on these definitions, we can establish that $|S_1| \geq f + r(\ell + 1)$ and $|S_2| = f + |V(\mathcal{T}_2)| + r\ell$. Now according to the definition of a certifying family, we have $|V(\mathcal{T}_2)| \leq (\ell + 1) \cdot |\mathcal{T}_2|$.

Now,

$$\begin{aligned} |S_2| &= f + r\ell + |V(\mathcal{T}_2)| \\ &\leq f + r\ell + (\ell + 1) \cdot |\mathcal{T}_2| \\ &\leq f + r\ell + (\ell + 2) \cdot |\mathcal{T}_2| \\ &\leq f + r\ell + r \\ &\leq f + r(\ell + 1) \\ &\leq |S_1| \end{aligned}$$

- (ii) S_2 is a solution of $(G - C, M, k, t - \ell)$. Next, we show that S_2 serves as a solution for $(G - C, M, k, t - \ell)$. We analyze a component $C_j \in \mathcal{C}_{2,t_i}$. There are two possible scenarios depending on the size of $\text{bdry}_{M \setminus S_2}(C_j)$.

- $\text{bdry}_{M \setminus S_2}(C_j) \leq \ell$.
Based on Lemma 5, we can have two possibilities. Either there exists a tuple $T_i \subseteq M \setminus S_2$ such that $|T_i| \leq (\ell + 1)$ and C_j belongs to the certifying family \mathcal{F}_T , or there exists a vertex set $U \subseteq V(C_j)$ that satisfies the following conditions: $|U| = \ell$, $\text{bdry}_{M \setminus S_2}(C_j) \subseteq U$, and $\text{comp}(C_j - U) \leq c$.
 - If there is a T_i belonging to \mathcal{T} such that C_j is in \mathcal{F}_{T_i} , it and $C_j \in \mathcal{C}_{2,t_i}$ implies that t_i must be a part of \hat{A} . However, this leads to a contradiction because it means T_i has a non-empty intersection with S_2 , which contradicts the fact that T_i is a subset of $M \setminus S_2$.
 - If there exists a subset U of the vertex set $V(C_j)$ such that $|U| = \ell$, $\text{bdry}_{M \setminus S_2}(C_j) \subseteq U$ and $\text{comp}(C_j - U) \leq c$, then we can define the set Z_j as U , which represents the set corresponding to $\text{coc}_c(C_j)$.
- $\text{bdry}_{M \setminus S_2}(C_j) \geq \ell + 1$.
Lemma 4 guarantees the existence of $T_i \in \mathcal{T}$ that fulfills the following conditions: $T_i \subseteq M \setminus S_2$, $|T_i| \leq (\ell + 1)$, and $C_j \in \mathcal{F}_T$. As a result, t_i must belong to \hat{A} according to Lemma 10. However, this implies that $T_i \cap S_2$ cannot be empty, which contradicts the fact that T_i is a subset of $M \setminus S_2$.

when $C_j \in \mathcal{C}_{1,t_i}$, the vertices $S_1 \cap C_j \subseteq S_2$ and no neighbor of $C_j \setminus S_1$ have been added to S_1 . So we are fine for this case. Hence the proof follows. \blacktriangleleft

The correctness of Reduction Rule 2 follows from the lemma below.

► **Lemma 12.** *Reduction Rule 2 is safe.*

Proof. The forward direction is straightforward. Suppose S is a solution to the instance (G, M, k, t) . Given that $G - C$ is a subgraph of G and $\text{coc}_c(C) = \ell$, we can conclude that S contains at least ℓ vertices from C . Hence, $S \setminus C$ forms a solution for $(G - C, M, k, t - \ell)$.

In the backward direction, let S_1 represent a solution for $(G - C, M, k, t - \ell)$. We will show that there exists a vertex set $Z \subseteq V(C)$ such that $S \cup Z$ forms a solution for (G, M, k, t) . Consider the sets \hat{A} , \hat{B} , and \hat{E} that satisfy the assumptions outlined in Reduction Rule 2. Within these assumptions, there exists a vertex $b \in \hat{B}$ associated with a vertex $w \in \hat{A}$. Specifically, $T \subseteq N(C)$ and $\text{local}(T, C) \geq \ell + 1$, where C represents the component in \mathcal{C}_ℓ corresponding to the vertex b in B , and $T \subseteq M$ is the tuple associated with the vertex w . However, due to the property of \hat{E} , there is no edge $e \in \hat{E}$ in which w and b are the endpoints. At this point, we invoke the algorithm provided by Lemma 11 to compute the set S_2 . As per Lemma 11, for each $t' \in \hat{A}$, we have $S_2 \cap T' \neq \emptyset$, where $T' \subseteq M$ represents the vertex set associated with the vertex t' in \hat{A} . Depending on the size of $\text{bdry}_{M \setminus S_2}(C)$, we encounter two cases.

Case 1. $\text{bdry}_{M \setminus S_2}(C) \geq \ell + 1$. By applying Lemma 4, we can establish the existence of $T_i \in \mathcal{T}$ that satisfies the following conditions: $T_i \subseteq M \setminus S_2$, $|T_i| \leq (\ell + 1)$, and $C \in \mathcal{F}_T$. Consequently, t_i must belong to \hat{A} (based on the Expansion Lemma, $b \in \hat{B}$ and $N(\hat{B}) \subseteq \hat{A}$). However, this implies that $T_i \cap S_2 \neq \emptyset$, which contradicts the fact that $T_i \subseteq M \setminus S_2$.

Case 2. $\text{bdry}_{M \setminus S_2}(C) \leq \ell$. Using Lemma 5, we can conclude that one of the following two cases holds: Either there exists $T_i \subseteq M \setminus S_2$, where $|T_i| \leq (\ell + 1)$ and $C \in \mathcal{F}_T$, or, there exists a vertex set $U \subseteq V(C)$ satisfying $|U| = \ell$, $\text{bdry}_{M \setminus S_2}(C) \subseteq U$, and $\text{comp}(C - U) \leq c$.

- If there exists $T_i \in \mathcal{T}$ such that $C \in \mathcal{F}_{T_i}$, then it follows that t_i must be in \hat{A} . However, this implies that $T_i \cap S_2 \neq \emptyset$, which contradicts the fact that $T_i \subseteq M \setminus S_2$.

5:10 Difference Determines the Degree: Structural Kernelizations of COC

- In the case where there exists a vertex set $U \subseteq V(C)$ that satisfies the conditions $|U| = \ell$, $\text{bdry}_{M \setminus S_2}(C) \subseteq U$, and $\text{comp}(C - U) \leq c$, we can set Z equal to U .

This completes the proof. \blacktriangleleft

Putting them all together, we get the following theorem.

► **Theorem 1.** c -COC/ d -COC admits a kernel with $\mathcal{O}(k^{d-c+1})$ vertices and $\mathcal{O}(k^{d-c+2})$ edges.

Proof. Consider the instance of the c -COC/ d -COC problem denoted as (G, M, k, t) . We begin by partitioning all the components \mathcal{C} into a maximum of $d - c$ parts denoted as $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_{d-c}$. Each part \mathcal{C}_ℓ is defined as the collection of components C in $G - M$ satisfying $\text{coc}_c(C) = \ell$, where $\ell \in [d - c]$. In other words, \mathcal{C}_ℓ contains components C for which the minimum-sized set $X \subseteq V(C)$ that guarantees $\text{comp}(C - X) \leq c$ is exactly ℓ . Towards solving the problem, we initially focus on each set \mathcal{C}_ℓ individually, aiming to reduce the number of components contained within each set.

We exhaustively apply Reduction Rules 1 and 2 to the set \mathcal{C}_ℓ for each $\ell \leq d - c$. Each reduction rule is capable of removing at least one vertex from the graph and can be executed in polynomial time. The running time of the algorithm takes into account the time required for applying Reduction Rule 2 as well as constructing the auxiliary bipartite graph, which facilitates the application of the expansion lemma. The construction of the bipartite graph can be accomplished in $k^{d-c+1} \cdot n^{\mathcal{O}(1)}$ time. Consequently, the entire kernelization procedure runs within polynomial time, specifically $n^{\mathcal{O}(1)}$. The output of the algorithm is the resulting instance (G', M', k', t') , which is guaranteed to be a kernel, meaning that no further reduction can be applied to it. The correctness of the algorithm is derived from the proofs establishing the safeness of the reduction rules (Lemmas 6 and 12).

We now argue about the size of the kernel. When Reduction Rules 1 and 2 are not applicable, we can establish that $|B| \leq q \cdot |A|$, where $q = (\ell + 2) + c(\ell + 1)$, and A and B represent the vertex sets of the auxiliary bipartite graph. Recall that each vertex $a \in A$ corresponds to a set of at most $\ell + 1 \leq (d - c) + 1$ vertices from M , and each vertex $b \in B$ corresponds to a set of at most d vertices. Consequently, we have $|A| \leq \mathcal{O}(k^{d-c+1})$ and $|B| \leq \mathcal{O}(k^{d-c+1})$. By combining these bounds, we can deduce that the size of the vertex set in the reduced instance (G', M', k', t') is upper bounded by $\mathcal{O}(k^{d-c+1})$. Additionally, the degree of each vertex in M' is bounded by $k + (d \cdot \mathcal{O}(k^{d-c+1}))$, while the degree of each vertex in $G' - M'$ is bounded by $k + d$. As a result, the size of the edge set in the reduced instance is upper bounded by $\mathcal{O}(k^{d-c+2})$. In conclusion, the size of the vertex set in the reduced instance is upper bounded by $\mathcal{O}(k^{d-c+1})$, and the size of the edge set is upper bounded by $\mathcal{O}(k^{d-c+2})$. Hence the proof follows. \blacktriangleleft

4 Kernel Lower bound

In this section, we show a lower bound for the size of the kernel of the problem we considered in this paper, under some complexity-theoretic assumptions. We prove it by giving a *parameter preserving transformation* from d -CNF-SAT to c -COC/ d -COC and using the VERTEX COVER kernelization lower bound due to Dell and Van Melkebeek [4].

Given a CNF formula where each clause has at most d literals, the d -CNF-SAT problem asks to find a boolean assignment of values to the variables such that each clause is satisfiable. The following two theorems are known due to Dell and Van Melkebeek [4].

► **Theorem 13** (Lower Bound for d -CNF-SAT [4]). *Let $d \geq 3$ be an integer. For any $\epsilon > 0$, the d -CNF-SAT problem parameterized by the number of variables (n) does not admit a polynomial compression with size $\mathcal{O}(n^{d-\epsilon})$, unless $\text{co-NP} \subseteq \text{NP/poly}$.*

► **Theorem 14** (Lower Bound for VERTEX COVER [4]). *For any $\epsilon > 0$, the VERTEX COVER problem parameterized by the solution size (k) does not admit a polynomial compression with size $\mathcal{O}(k^{2-\epsilon})$, unless $\text{co-NP} \subseteq \text{NP/poly}$.*

► **Definition 15** (Parameter preserving transformation (PPT)). *Let Π_1 and Π_2 be two parameterized problems. We say that there exists a parameter preserving transformation from Π_1 to Π_2 if there exists a polynomial time algorithm \mathcal{B} that given an instance (x, k) of Π_1 , constructs an instance (x', k') of Π_2 such that*

- $(x, k) \in \Pi_1$ if and only if $(x', k') \in \Pi_2$, and
- $k' \leq \mathcal{O}(k)$.

Below we provide a parameter preserving transformation from d -CNF-SAT to c -COC/ d -COC when $c = 1$. In particular, we prove the following lemma.

► **Lemma 16** (Reduction from d -CNF-SAT to VERTEX COVER/ d -COC). *There exists a parameter preserving transformation from the d -CNF-SAT parameterized by the number of variables to VERTEX COVER/ d -COC. In the VERTEX COVER/ d -COC problem, the size of the modulator is twice the number of variables present in the d -CNF-SAT formula.*

Proof. Let Φ be a d -CNF formula, an instance of d -CNF-SAT, consisting of n variables, denoted as $\{x_1, x_2, \dots, x_n\}$, and m clauses $\{C_1, C_2, \dots, C_m\}$. Since Φ is a d -CNF formula, each clause contains at most d literals. We construct an instance (G, k, t) for VERTEX COVER/ d -COC using the following construction:

- For each variable x , we introduce two vertices denoted as x^1 and x^2 , and connect them with an edge (x^1, x^2) .
- For a clause C_j consisting of d_j literals, we include a clique of size d_j . Within the clique, we label the vertices as follows: a vertex is named $v_{i,j}$ if the literal x_i or \bar{x}_i is present in clause C_j .
- For every $i \in [n], j \in [m]$, if x_i is a literal in clause C_j , we add the edge $(x_i, v_{i,j})$. Similarly, if \bar{x}_i is a literal in clause C_j , we add the edge $(\bar{x}_i, v_{i,j})$.

▷ **Claim 17.** There exists a vertex subset $S \subseteq V(G)$ of size $2n$ such that $\text{comp}(G - S) \leq d$.

Proof. Let S be defined as the set containing elements x_i and \bar{x}_i for all $i \in [n]$. Clearly $|S| = 2n$. Considering the construction of graph G , it can be observed that every component in $G - S$ forms a clique with a maximum size of d . Therefore, we have $\text{comp}(G - S) \leq d$. ◁

▷ **Claim 18.** Φ is satisfiable if and only if G has a vertex cover of size $n - m + \sum_{i=1}^m |C_i|$.

Proof. In the forward direction, assuming that Φ is satisfiable, we show the existence of a vertex cover in G with a size of $n - m + \sum_{i=1}^m |C_i|$. To construct this vertex cover, we proceed as follows:

- For every variable x , we include x in the set S if it is assigned the value **true** in the satisfying assignment. Otherwise, we add \bar{x} to S .
- For each clause C_j , assuming that variable x_a makes clause C_j satisfiable, we add all vertices from the corresponding clique to S except for the vertex $v_{a,j}$.

5:12 Difference Determines the Degree: Structural Kernelizations of COC

It is evident that the set S constructed as described above forms a vertex cover. Furthermore, the size of S is bounded by $n - m + \sum_{i=1}^m |C_i|$ due to the construction process.

In the backward direction, let R be a vertex cover of G with a size of $n - m + \sum_{i=1}^m |C_i|$. Since any vertex cover in C_i must have size at least $|C_i| - 1$, it follows that for each i , exactly one of x_i and \bar{x}_i is present in R . Now, we construct an assignment β for the variables in Φ as follows: we set x_i to **true** if x_i is in R , and we set x_i to **false** otherwise. Our objective is to show that β satisfies all the clauses. Consider the clique corresponding to clause C_j . We observe that exactly $d_j - 1$ vertices are present in R . Therefore, there exists a vertex, denoted as $v_{a,j}$, which is not in R . However, since R is a vertex cover, it must contain the vertex x_a in order to cover the edge $(v_{a,j}, x_a)$. Since x_a is assigned the value **true**, the literal x_a satisfies the clause C_j . Hence, every clause in Φ is satisfied by the assignment β . \triangleleft

As the transformation of d -CNF-SAT to VERTEX COVER/ d -COC can be performed in $n^{\mathcal{O}(1)}$ time, the lemma follows from the above two claims. \blacktriangleleft

Now we have the following theorem.

► Theorem 2. *For every $\epsilon > 0$ and every positive integer d , VERTEX COVER/ d -COC has no compression of vertex size $\mathcal{O}(k^{d-\epsilon})$ unless $\text{co-NP} \subseteq \text{NP/poly}$.*

Proof. Our proof is divided into three cases in order to prove that, for any $\epsilon > 0$ and an integer $d \in \mathbb{N}$, there exists no polynomial time algorithm that can transform a given instance of VERTEX COVER/ d -COC to an equivalent instance of any arbitrary problem with $\mathcal{O}(k^{d-\epsilon})$ bits, unless $\text{co-NP} \subseteq \text{NP/poly}$.

Case 1. $d = 1$. In the case where $d = 1$, the problem known as VERTEX COVER/ 1 -COC refers to the VERTEX COVER problem parameterized by solution size. In this particular case, the result stated in Theorem 14 proves the theorem.

Case 2. $d = 2$. We can observe that the size of a 2-COC set is at most the size of a minimum vertex cover of the graph. As a result, 2-COC can be considered a parameter smaller than 1-COC (vertex cover). Therefore, if VERTEX COVER/ 2 -COC admits a compression of $\mathcal{O}(k^{2-\epsilon})$ bits, it would imply that the VERTEX COVER problem parameterized by the solution size also has a compression of $\mathcal{O}(k^{2-\epsilon})$ bits. However, this contradicts the result stated in Theorem 14.

Case 3. $d \geq 3$. Suppose we have an instance (G, k, t) of VERTEX COVER/ d -COC, where $d \geq 3$, and there exists a polynomial time algorithm \mathcal{A} that can transform (G, k, t) into an equivalent instance I of an arbitrary problem L such that I can be represented using $\mathcal{O}(k^{d-\epsilon})$ bits. To demonstrate the implications of this assumption, let us consider an instance Π of d -CNF-SAT with n variables and m clauses. First, we utilize the polynomial time algorithm described in Lemma 16 to transform Π into an instance $(G, 2n, n + \sum_{j=1}^m d_j - 1)$ of VERTEX COVER/ d -COC. Next, we apply the algorithm \mathcal{A} to this transformed instance $(G, 2n, n + \sum_{j=1}^m d_j - 1)$, resulting in an equivalent instance I of problem L . Based on our assumption, we know that I can be represented using $\mathcal{O}(2n^{d-\epsilon}) = \mathcal{O}(n^{d-\epsilon})$ bits. However, this leads to a contradiction with $\text{co-NP} \subseteq \text{NP/poly}$, as stated in Theorem 13.

This completes the proof. \blacktriangleleft

5 Conclusion

In this paper, we show that c -COC/ d -COC admits a polynomial kernel with $\mathcal{O}(k^{d-c+1})$ vertices and $\mathcal{O}(k^{d-c+2})$ edges, where k is the size of the minimum d -coc set. Importantly, we observe that the degree of the polynomial in the kernel size is solely determined by the difference between d and c , and is independent of the specific values of d and c . Furthermore, we establish that obtaining a uniform kernel for the problem, where the exponent of k is independent of $d-c$, is unlikely under reasonable complexity assumptions. This result contributes valuable insights to the field of kernelization for VERTEX COVER, particularly regarding c -COMPONENT ORDER CONNECTIVITY, when considering parameterizations smaller than the conventional solution size.

References



- 1 Marin Bougeret, Bart M. P. Jansen, and Ignasi Sau. Bridge-depth characterizes which minor-closed structural parameterizations of vertex cover admit a polynomial kernel. *SIAM J. Discret. Math.*, 36(4):2737–2773, 2022. doi:10.1137/21m1400766.
- 2 Marin Bougeret and Ignasi Sau. How much does a treedepth modulator help to obtain polynomial kernels beyond sparse graphs? *Algorithmica*, 81(10):4043–4068, 2019. doi:10.1007/s00453-018-0468-8.
- 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 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.
- 5 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting forbidden minors: Approximation and kernelization. *SIAM J. Discret. Math.*, 30(1):383–410, 2016. doi:10.1137/140997889.
- 6 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing: Theory of parameterized preprocessing*. Cambridge University Press, United Kingdom, 2019. Publisher Copyright: © Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi 2019. doi:10.1017/9781107415157.
- 7 Fedor V. Fomin and Torstein J. F. Strømme. Vertex cover structural parameterization revisited. In Pinar Heggernes, editor, *Graph-Theoretic Concepts in Computer Science - 42nd International Workshop, WG 2016, Istanbul, Turkey, June 22-24, 2016, Revised Selected Papers*, volume 9941 of *Lecture Notes in Computer Science*, pages 171–182, 2016. doi:10.1007/978-3-662-53536-3_15.
- 8 Archontia C. Giannopoulou, Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. Uniform kernelization complexity of hitting forbidden minors. *ACM Trans. Algorithms*, 13(3):35:1–35:35, 2017. doi:10.1145/3029051.
- 9 Eva-Maria C. Hols and Stefan Kratsch. Smaller parameters for vertex cover kernelization. In Daniel Lokshtanov and Naomi Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, volume 89 of *LIPICs*, pages 20:1–20:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.IPEC.2017.20.
- 10 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited - upper and lower bounds for a refined parameter. *Theory Comput. Syst.*, 53(2):263–299, 2013. doi:10.1007/s00224-012-9393-4.
- 11 Stefan Kratsch. A randomized polynomial kernelization for vertex cover with a smaller parameter. *SIAM J. Discret. Math.*, 32(3):1806–1839, 2018. doi:10.1137/16M1104585.
- 12 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.

- 13 Mithilesh Kumar and Daniel Lokshtanov. A $2lk$ kernel for l -component order connectivity. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPICs*, pages 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.IPEC.2016.20.
- 14 Michael Lampis. A kernel of order $2k - c \log k$ for vertex cover. *Inf. Process. Lett.*, 111(23-24):1089–1091, 2011. doi:10.1016/j.ip1.2011.09.003.
- 15 Diptapriyo Majumdar, Venkatesh Raman, and Saket Saurabh. Polynomial kernels for vertex cover parameterized by small degree modulators. *Theory Comput. Syst.*, 62(8):1910–1951, 2018. doi:10.1007/s00224-018-9858-1.
- 16 Geevarghese Philip, Varun Rajan, Saket Saurabh, and Prafullkumar Tale. Subset feedback vertex set in chordal and split graphs. *Algorithmica*, 81(9):3586–3629, 2019. doi:10.1007/s00453-019-00590-9.
- 17 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, 2010. doi:10.1145/1721837.1721848.
- 18 Mingyu Xiao. Linear kernels for separating a graph into components of bounded size. *J. Comput. Syst. Sci.*, 88:260–270, 2017. doi:10.1016/j.jcss.2017.04.004.

The Parameterised Complexity Of Integer Multicommodity Flow

Hans L. Bodlaender  

Utrecht University, The Netherlands

Isja Mannens  

Utrecht University, The Netherlands

Jelle J. Oostveen  

Utrecht University, The Netherlands

Sukanya Pandey  

Utrecht University, The Netherlands

Erik Jan van Leeuwen  

Utrecht University, The Netherlands

Abstract

The INTEGER MULTICOMMODITY FLOW problem has been studied extensively in the literature. However, from a parameterised perspective, mostly special cases, such as the DISJOINT PATH problem, have been considered. Therefore, we investigate the parameterised complexity of the general INTEGER MULTICOMMODITY FLOW problem. We show that the decision version of this problem on directed graphs for a constant number of commodities, when the capacities are given in unary, is XNLP-complete with pathwidth as parameter and XALP-complete with treewidth as parameter. When the capacities are given in binary, the problem is NP-complete even for graphs of pathwidth at most 13. We give related results for undirected graphs. These results imply that the problem is unlikely to be fixed-parameter tractable by these parameters.

In contrast, we show that the problem does become fixed-parameter tractable when weighted tree partition width (a variant of tree partition width for edge weighted graphs) is used as parameter.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Graph algorithms analysis; Theory of computation → Problems, reductions and completeness

Keywords and phrases multicommodity flow, parameterised complexity, XNLP-completeness, XALP-completeness

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.6

Related Version *Full Version:* <https://doi.org/10.48550/arXiv.2310.05784>

Funding *Isja Mannens:* The research of Isja Mannens was supported by the project CRACKNP that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 853234).

Jelle J. Oostveen: The research of Jelle Oostveen was supported by the NWO grant OCENW.KLEIN.114 (PACAN).

1 Introduction

The MULTICOMMODITY FLOW problem is the generalisation of the textbook flow problem where instead of just one commodity, multiple different commodities have to be transported through a network. The problem models important operations research questions (see e.g. [32]). Although several optimisation variants of this problem exist [32], we consider only the variant where for each commodity, a given amount of flow (the demand) has to be sent from the commodity's source to its sink, subject to a capacity constraint on the total



© Hans L. Bodlaender, Isja Mannens, Jelle J. Oostveen, Sukanya Pandey, and Erik Jan van Leeuwen; licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 6; pp. 6:1–6:19

Leibniz International Proceedings in Informatics



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

amount flow through each arc. The nature and computational complexity of the problem is highly influenced by the graph (undirected or directed, its underlying structure) and the capacities, demands, and flow value (integral or not, represented in unary or binary). When the flow values are allowed to be fractional, the problem can be trivially solved through a linear program (see e.g. [22, 25]).

We focus on INTEGER MULTICOMMODITY FLOW, where all the given capacities and demands are integers and the output flow must be integral. The INTEGER MULTICOMMODITY FLOW problem is widely studied and well known to be NP-hard even if all capacities are 1, on both directed and undirected graphs, even when there are only two commodities [11]. On directed graphs, it is NP-hard even for two commodities of demand 1 [14]. These strong hardness results have led to a range of heuristic solution methods as well as a substantial body of work on approximation algorithms. For surveys, see e.g., [1, 32, 33].

An important special case of INTEGER MULTICOMMODITY FLOW and the main source of its computational hardness is the EDGE DISJOINT PATHS problem. It can be readily seen that INTEGER MULTICOMMODITY FLOW is equivalent to EDGE DISJOINT PATHS when all capacities and demands are 1. Indeed, all aforementioned hardness results stem from this connection. The EDGE DISJOINT PATHS problem has been studied broadly in its own right (see e.g. the surveys by Frank [16] and Vygen [31]), including a large literature on approximation algorithms. See, amongst others [21, 30] for further hardness and inapproximability results. On undirected graphs, EDGE DISJOINT PATHS is fixed-parameter tractable parameterised by the number of source-sink pairs [28, 24].

Investigation of the parameterised complexity of EDGE DISJOINT PATHS has recently been continued by considering structural parameterisations. Unfortunately, the problem is NP-hard for graphs of treewidth 2 [26] and even for graphs with a vertex cover of size 3 [13]. It is also $W[1]$ -hard parameterised by the size of a vertex set whose removal leaves an independent set of vertices of degree 2 [19]. From an algorithmic perspective, Ganian and Ordyniak [19] showed that EDGE DISJOINT PATHS is in XP parameterised by tree-cut width. Zhou et al. [34] give two XP algorithms for EDGE DISJOINT PATHS for graphs of bounded treewidth: one for when the number of paths is small, and one for when a specific condition holds on the pairs of terminals. Ganian et al. [20] give an FPT algorithm parameterised by the treewidth and degree of the graph. Friedrich et al. [17, 18] give approximation algorithms for multicommodity flow on graphs of bounded treewidth.

These results naturally motivate the question: *What can we say about the parameterised complexity of the general INTEGER MULTICOMMODITY FLOW problem under structural parameterisations?* We are unaware of any explicit studies in this direction. We do note that the result of Zhou et al. [34] implies an XP algorithm on graphs of bounded treewidth for a bounded number of commodities if the capacities are given in unary. We are particularly interested in whether this result can be improved to an FPT algorithm, which is hitherto unknown.

Our Setting and Contributions

We consider the INTEGER MULTICOMMODITY FLOW problem for a small, fixed number of commodities. In particular, INTEGER ℓ -COMMODITY FLOW is the variant in which there are ℓ commodities. Furthermore, we study the setting where some well-known structural parameter of the input graph, particularly its pathwidth or treewidth, is small.

Our main contribution is to show that INTEGER 2-COMMODITY FLOW is unlikely to be fixed-parameter tractable parameterised by treewidth and or by pathwidth. Instead of being satisfied with just a $W[t]$ -hardness result for some t or any t , we seek stronger results using the recently defined complexity classes XNLP and XALP. An overview of our results can be found in Table 1.

■ **Table 1** Overview of our results for INTEGER 2-COMMODITY FLOW. para-NP-complete = NP-complete for fixed value of parameter. (1) = capacities of arcs inside bags can be arbitrary, capacities of arcs between bags are bounded by weighted tree partition width. (2) Approximation, see Theorem 1.11; conjectured in FPT. For the undirected case, the same results hold, except that for the para-NP-completeness for the parameters pathwidth and treewidth, we need a third commodity.

Parameter	unary capacities	binary capacities
pathwidth	XNLP-complete	para-NP-complete
treewidth	XALP-complete	para-NP-complete
weighted tree partition width	FPT (1)	FPT (1)
vertex cover	(2); in XP	(2); open

XNLP is the class of parameterised problems that can be solved on a non-deterministic Turing machine in $f(k)|x|^{O(1)}$ time and $f(k) \log |x|$ memory for a computable function f , where $|x|$ is the size of the input x . The class XNLP (under a different name) was first introduced by Elberfeld et al. [9]. Bodlaender et al. [2, 5, 7] showed a number of problems to be XNLP-complete with pathwidth as parameter. In particular, [2] gives XNLP-completeness proofs for several flow problems with pathwidth as parameter.

In this work, we prove XNLP-completeness (and stronger) results for INTEGER ℓ -COMMODITY FLOW. These give a broad new insight into the complexity landscape of INTEGER MULTICOMMODITY FLOW. We distinguish how the capacities of arcs and edges are specified: these can be given in either unary or binary. First, we consider the unary case:

► **Theorem 1.1.** *INTEGER 2-COMMODITY FLOW with capacities given in unary, parameterised by pathwidth, is XNLP-complete.*

► **Theorem 1.2.** *UNDIRECTED INTEGER 2-COMMODITY FLOW with capacities given in unary, parameterised by pathwidth, is XNLP-complete.*

These hardness results follow by reduction from the XNLP-complete CHAINED MULTICOLOURED CLIQUE problem [6], a variant of the perhaps more familiar MULTICOLOURED CLIQUE problem [12]. We follow a common strategy in such reductions, using vertex selection and edge verification gadgets. However, a major hurdle is to use flows to select vertices and verify the existence of edges to form the sought-after cliques. To pass this hurdle, we construct gadgets that use Sidon sets as flow values, combined with gadgets to check that a flow value indeed belongs to such a Sidon set.

For the parameter treewidth, we are able to show a slightly stronger result. Recently, Bodlaender et al. [6] introduced the complexity class XALP, which is the class of parameterised problems that can be solved on a non-deterministic Turing machine that has access to an additional stack, in $f(k)|x|^{O(1)}$ time and $f(k) \log |x|$ space (excluding the space used by the stack), for a computable function f , where $|x|$ again denotes the size of the input x . Many problems that are XNLP-complete with pathwidth as parameter are XALP-complete with treewidth as parameter. This also holds for the INTEGER MULTICOMMODITY FLOW problem:

► **Theorem 1.3 (♣).** *INTEGER 2-COMMODITY FLOW with capacities given in unary, parameterised by treewidth, is XALP-complete.*

The reduction is from the XALP-complete TREE-CHAINED MULTICOLOURED CLIQUE problem [7] and follows similar ideas as the above reduction. Combining techniques of the proofs of Theorems 1.2 and 1.3 gives the following result.

► **Theorem 1.4** (♣). *UNDIRECTED INTEGER 2-COMMODITY FLOW with capacities given in unary, parameterised by treewidth, is XALP-complete.*

Assuming the *Slice-wise Polynomial Space Conjecture* [27, 5], these results show that XP-algorithms for INTEGER 2-COMMODITY FLOW or UNDIRECTED INTEGER 2-COMMODITY FLOW for graphs of small pathwidth or treewidth cannot use only $f(k)|x|^{O(1)}$ memory. Moreover, the results imply these problems are $W[t]$ -hard for all positive integers t .

If the capacities are given in binary, then the problems become even harder.

► **Theorem 1.5.** *INTEGER 2-COMMODITY FLOW with capacities given in binary is NP-complete for graphs of pathwidth at most 13.*

► **Theorem 1.6** (♣). *UNDIRECTED INTEGER 3-COMMODITY FLOW with capacities given in binary is NP-complete for graphs of pathwidth at most 18.*

Finally, we consider a variant of the INTEGER MULTICOMMODITY FLOW problem where the flow must be *monochrome*, i.e. a flow is only valid when no edge carries more than one type of commodity. Then, we obtain hardness even for parameterisation by the vertex cover number of the graph, for both variants of the problem.

► **Theorem 1.7** (♣). *INTEGER 2-COMMODITY FLOW WITH MONOCHROME EDGES is NP-hard for binary weights and vertex cover number 6, and $W[1]$ -hard for unary weights when parameterised by the vertex cover number.*

► **Theorem 1.8** (♣). *UNDIRECTED INTEGER 2-COMMODITY FLOW WITH MONOCHROME EDGES is NP-hard for binary weights and vertex cover number 6, and $W[1]$ -hard for unary weights when parameterised by the vertex cover number.*

To complement our hardness results, we prove two algorithmic results. Bodlaender et al. [2] had given FPT algorithms for several flow problems, using the recently defined notion of weighted tree partition width as parameter (see [3, 2]). Weighted tree partition width can be seen as a variant of the notion of *tree partition width* for edge-weighted graphs, introduced by Seese [29] in 1985 under the name *strong treewidth*. See Section 2 for formal definitions of these parameters. The known hardness for the vertex cover number [13] implies that EDGE DISJOINT PATHS is NP-hard even for graphs of tree partition width 3. Here, we prove that:

► **Theorem 1.9.** *The INTEGER ℓ -COMMODITY FLOW problem can be solved in time $2^{2^{b^{3\ell b}}} n^{O(1)}$, where b is the breadth of a given tree partition of the input graph.*

► **Theorem 1.10.** *The UNDIRECTED INTEGER ℓ -COMMODITY FLOW problem can be solved in time $2^{2^{b^{3\ell b}}} n^{O(1)}$, where b is the breadth of a given tree partition of the input graph.*

For the standard INTEGER 2-COMMODITY FLOW problem with the vertex cover number $\text{vc}(G)$ of the input graph G as parameter, we conjecture that this problem is in FPT. As a partial result, we can give the following approximation algorithms:

► **Theorem 1.11** (♣). *There is a polynomial-time algorithm that, given an instance of INTEGER 2-COMMODITY FLOW on a graph G with demands d_1, d_2 , either outputs that there is no flow that meets the demands or outputs a 2-commodity flow of value at least $d_i - O(\text{vc}(G)^3)$ for each commodity $i \in [2]$.*

► **Theorem 1.12 (♣).** *There is a polynomial-time algorithm that, given an instance of UNDIRECTED INTEGER 2-COMMODITY FLOW on a graph G with demands d_1, d_2 , either outputs that there is no flow that meets the demands or outputs a 2-commodity flow of value at least $d_i - O(\text{vc}(G)^3)$ for commodity $i \in [2]$.*

Proofs of theorems marked by ♣ appear in the full version. For other theorems, we are only able to provide proof sketches in the limited space.

2 Preliminaries

In this paper, we consider both directed and undirected graphs. Graphs are directed unless explicitly stated otherwise. Arcs and edges are denoted as vw (an arc from v to w , or an edge with v and w as endpoints).

We use the interval notation for intervals of integers, e.g., $[-1, 3] = \{-1, 0, 1, 2, 3\}$. We simplify this notation for intervals that start at 1, i.e. $[k] = [1, k]$. Moreover, we use $\mathbb{N} = \{1, 2, \dots\}$ and $\mathbb{N}_0 = \{0, 1, 2, \dots\}$.

A *Sidon set* is a set of positive integers $\{a_1, a_2, \dots, a_n\}$ such that all pairs have a different sum, i.e., when $a_i + a_{i'} = a_j + a_{j'}$ then $\{i, i'\} = \{j, j'\}$. Sidon sets are also Golomb rulers and vice versa – in a Golomb ruler, pairs of different elements have unequal differences: if $i \neq i'$ and $j \neq j'$, then $|a_i - a_{i'}| = |a_j - a_{j'}|$, then $\{i, i'\} = \{j, j'\}$. A construction by Erdős and Turán [10] for Sidon sets implies the following, cf. the discussion in [8].

► **Theorem 2.1.** *A Sidon set of n elements in $[4n^2]$ can be found in $O(n\sqrt{n})$ time and logarithmic space.*

We now formally define our flow problems. A *flow network* is a pair (G, c) of a directed (undirected) graph $G = (V, E)$ and a function $c: E \rightarrow \mathbb{N}_0$ that assigns to each arc (edge) a non-negative integer *capacity*. We generally use $n = |V|$ and $m = |E|$.

For a positive integer ℓ , an ℓ -commodity flow in a flow network with sources $s_1, \dots, s_\ell \in V$ and sinks $t_1, \dots, t_\ell \in V$ is a ℓ -tuple of functions $f^1, \dots, f^\ell: E \rightarrow \mathbb{R}_{\geq 0}$, that fulfils the following conditions:

- **Flow conservation.** For all $i \in [\ell]$, $v \notin \{s_i, t_i\}$, $\sum_{vw \in E} f^i(vw) = \sum_{vw \in E} f^i(vw)$.
- **Capacity.** For all $vw \in E$, $\sum_{i \in [\ell]} f^i(vw) \leq c(vw)$.

An ℓ -commodity flow is an *integer ℓ -commodity flow* if for all $i \in [c]$, $vw \in E$, $f^i(vw) \in \mathbb{N}_0$. The *value for commodity i* of an ℓ -commodity flow equals $\sum_{s_i w \in E} f^i(s_i w) - \sum_{w s_i \in E} f^i(w s_i)$. We shorten this to “flow” when it is clear from context what the value of ℓ is and whether we are referring to an integer or non-integer flow.

The main problem considered in the paper now is as follows:

INTEGER ℓ -COMMODITY FLOW

Input: A flow network $G = (V, E)$ with capacities c , sources $s_1, \dots, s_\ell \in V$, sinks $t_1, \dots, t_\ell \in V$, and demands $d_1, \dots, d_\ell \in \mathbb{N}$.

Question: Does there exist an integer ℓ -commodity flow in G which has value d_i for each commodity $i \in [\ell]$?

The INTEGER MULTICOMMODITY FLOW problem is the union of all INTEGER ℓ -COMMODITY FLOW problems for all non-negative integers ℓ .

For undirected graphs, flow still has direction, but the capacity constraint changes to:

- **Capacity.** For all $vw \in E$, $\sum_{i \in [\ell]} f^i(vw) + f^i(wv) \leq c(vw)$.

The undirected version of the INTEGER ℓ -COMMODITY FLOW problem then is as follows:

UNDIRECTED INTEGER ℓ -COMMODITY FLOW

Input: An *undirected* flow network $G = (V, E)$ with capacities c , sources $s_1, \dots, s_\ell \in V$, sinks $t_1, \dots, t_\ell \in V$, and demands $d_1, \dots, d_\ell \in \mathbb{N}$.

Question: Does there exist an integer ℓ -commodity flow in G which has value d_i for each commodity $i \in [\ell]$?

We use the well-known parameters *treewidth* and *pathwidth* without giving an explicit definition. For the parameter (weighted) tree partition width, refer to [2] (see also [3]). We also use these parameters for directed graphs. In that case, the direction of edges is ignored.

The classes XNLP and XALP were defined in the introduction. XNLP-hardness and XALP-hardness are defined with respect to pl-reductions. The main difference with the more standard parameterized reductions is that the computation of the reduction must be done with logarithmic space. In most cases, existing parameterized reductions are also pl-reductions; logarithmic space is achieved by not storing intermediate results but recomputing these when needed.

Our hardness results stem from two variants of MULTICOLOURED CLIQUE (see [12]):

CHAINED MULTICOLOURED CLIQUE

Input: A graph $G = (V, E)$, a partition of V into V_1, \dots, V_r , such that $|i - j| \leq 1$ for each edge $uv \in E(G)$ with $u \in V_i$ and $v \in V_j$, and a function $c: V \rightarrow [k]$.

Parameter: k .

Question: Is there a set of vertices $W \subseteq V$ such that for all $i \in [r - 1]$, $W \cap (V_i \cup V_{i+1})$ is a clique, and for each $i \in [r]$ and $j \in [k]$, there is a vertex $v \in W \cap V_i$ with $c(v) = j$?

TREE-CHAINED MULTICOLOURED CLIQUE

Input: A graph $G = (V, E)$, a tree partition $(\{V_i \mid i \in I\}, T = (I, F))$ with T a tree of maximum degree 3, and a function $c: V \rightarrow [k]$.

Parameter: k .

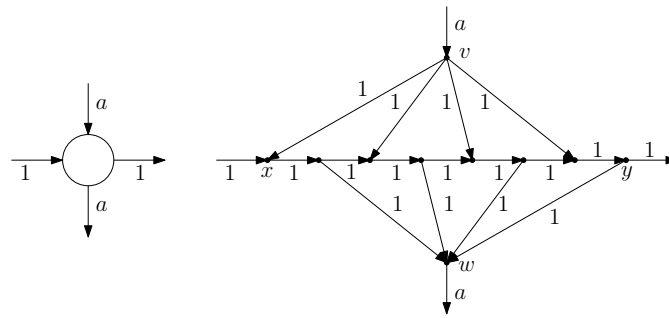
Question: Is there a set of vertices $W \subseteq V$ such that for all $ii' \in F$, $W \cap (V_i \cup V_{i'})$ is a clique, and for each $i \in I$ and $j \in [k]$, there is a vertex $v \in W \cap V_i$ with $c(v) = j$?

► **Theorem 2.2** (From [6] and [7]). *CHAINED MULTICOLOURED CLIQUE is XNLP-complete, and TREE-CHAINED MULTICOLOURED CLIQUE is XALP-complete.*

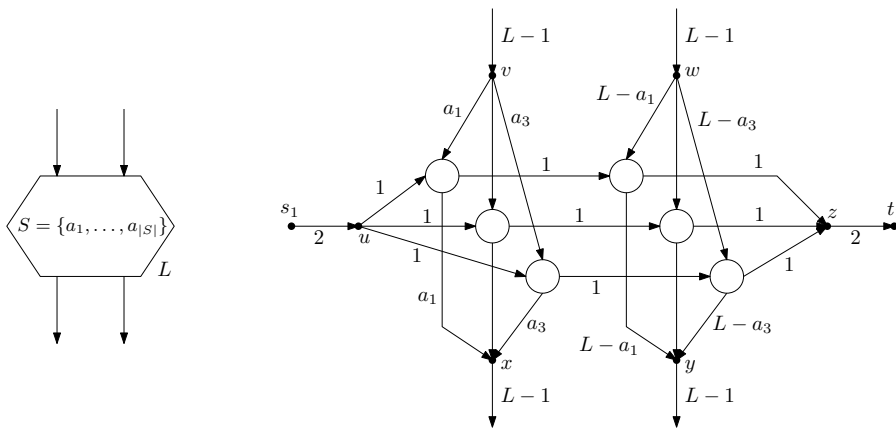
3 Hardness Results – Unary Capacities

We prove our hardness results for INTEGER MULTICOMMODITY FLOW with unary capacities, parameterised by pathwidth. We aim to reduce from CHAINED MULTICOLOURED CLIQUE. It is good to know that all constructions will have disjoint sources and sinks for the different commodities. We will set the demands for each commodity equal to the total capacity of the outgoing arcs from the sources, which is equal to the total capacity of the incoming arcs to the sinks. Thus, the flow over such arcs will be equal to their capacity. Furthermore, throughout this section, our constructions will have two commodities. We name the commodities 1 and 2, with sources s_1, s_2 and sinks t_1, t_2 , respectively.

We first introduce two types of gadgets: subgraphs that fulfil certain properties and that are used in the hardness constructions. Given an integer a , the *a-Gate gadget* (see Figure 1) either can move 1 unit of flow from one commodity from left to right, or at most a units



■ **Figure 1** The a -Gate gadget. Left: the schematic representation of the gadget, with its entry and exit arcs. Right: the full construction for $a = 4$, with arcs labelled by their capacities.



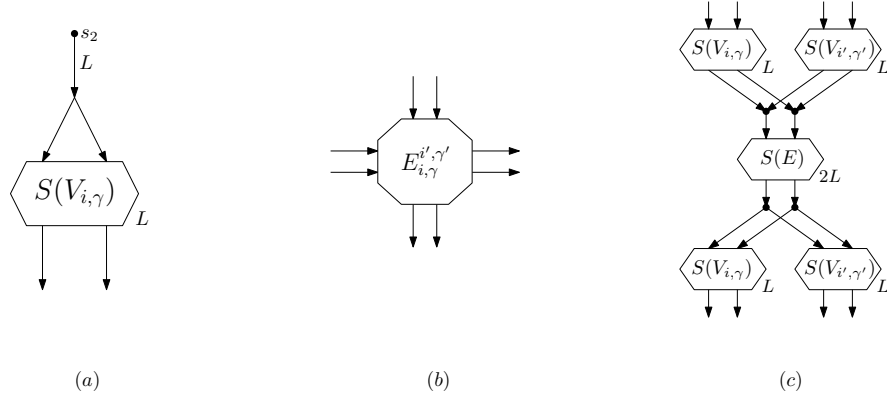
■ **Figure 2** The (S, L) -Verifier gadget. Left: a schematic representation of the gadget, with its entry and exit arcs. The value on the bottom-right of the schematic representation denotes the sum of the incoming flows to the gadget. Right: the graph that realises the gadget for $S = \{a_1, a_2, a_3\}$, with arcs labelled by their capacities (the unlabelled arcs have capacities a_2 and $L - a_2$ respectively; their labels are omitted for clarity).

of flow from the other commodity from top to bottom, but not both. Hence, it models a form of choice. This gadget will grow in size with a , and thus will only be useful if the input values are given in unary. Given a set S of integers and a large integer L (larger than any number in S), the (S, L) -Verifier (see Figure 2) is used to check if the flow over an arc belongs to a number in S . The a -Gate gadget is used as a sub-gadget in this construction. In our reduction, later, we will use appropriately constructed sets S to select vertices or to check for the existence of edges.

Both types of gadget have constant pathwidth. The arcs incoming and outgoing of the gadget are called the *entry arcs* and *exit arcs* respectively.

Our hardness construction will be built using only Verifier gadgets as subgadgets. The entry arcs and exit arcs of this gadget are meant to transport solely flow of commodity 2. Hence, in the remainder, it helps to think of only commodity 2 being transported along the edges, to focus on the exact value of that flow. This value indicate which vertex is selected or whether two selected vertices are adjacent.

► **Theorem 1.1.** *INTEGER 2-COMMODITY FLOW with capacities given in unary, parametrised by pathwidth, is XNLP-complete.*



■ **Figure 3** (a) A Vertex selector gadget. (b) A schematic representation of an Edge check gadget for $V_{i,\gamma}$ and $V_{i',\gamma'}$. (c) The construction of an Edge check gadget.

Proof sketch. Proof of membership in XNLP follows immediately from a dynamic programming algorithm. For the hardness, we reduce from CHAINED MULTICOLOUR CLIQUE (see Theorem 2.2). Suppose we have an instance of CHAINED MULTICOLOUR CLIQUE, with a graph $G = (V, E)$, colouring $c : V \rightarrow [k]$, and partition V_1, \dots, V_r of V .

Build a Sidon set with $|V|$ numbers by applying the algorithm of Theorem 2.1. Following the same theorem, the numbers are in $[4|V|^2]$. Set $L = 4|V|^2 + 1$ to be a “large” integer. To each vertex $v \in V$, we assign a unique element of the set S , denoted by $S(v)$. For any subset $V' \subseteq V$, let $S(V') = \{S(v) \mid v \in V'\}$. For any subset $E' \subseteq E$, let $S(E') = \{S(u) + S(v) \mid uv \in E'\}$.

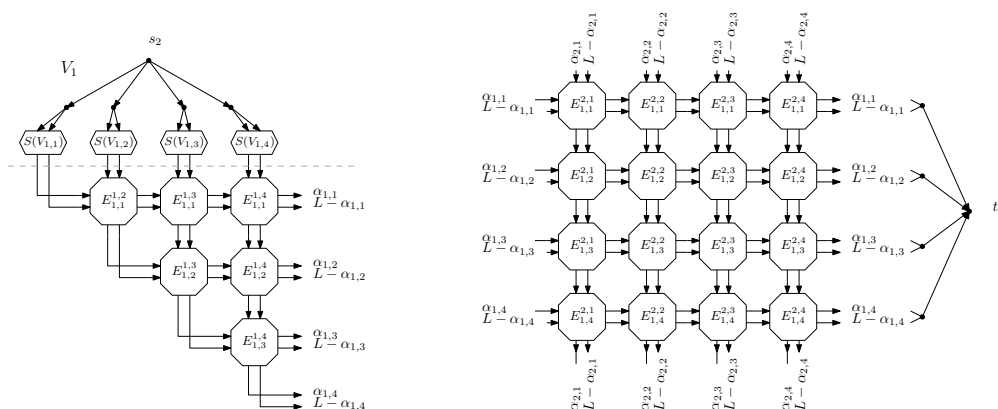
We now describe several (further) gadgets that we use to build the full construction. Let $V_{i,\gamma}$ be the vertices in V_i with colour γ . Each set $V_{i,\gamma}$ is called a *class*. For each class $V_{i,\gamma}$, we use a *Vertex selector gadget* (see Figure 3) to select the vertex from $V_{i,\gamma}$ that should be in the solution to the CHAINED MULTICOLOURED CLIQUE instance. We select some $v \in V_{i,\gamma}$ if and only if the left branch receives $S(v)$ flow and the right branch receives $L - S(v)$ flow.

For each pair of incident classes, we construct an *Edge check gadget* (see Figure 3). That is, we have an Edge check gadget for all classes $V_{i,\gamma}$ and $V_{i',\gamma'}$ with $|i - i'| \leq 1$, and $\{i, \gamma\} \neq \{i', \gamma'\}$. An Edge check gadget will check if two incident classes have vertices selected that are adjacent. If the entry arcs have flow (of commodity 2) of value $S(v)$, $L - S(v)$, $S(w)$, and $L - S(w)$ consecutively, then there is a valid flow if and only if $vw \in E$. Note that the sum $S(v) + S(w)$ is unique, because S is a Sidon set, and thus so is $2L - (S(v) + S(w))$. Hence, the only way for the flow to split up again and leave via the exit arcs is to split into $S(v)$, $S(w)$, $L - S(v)$, and $L - S(w)$; otherwise, it cannot pass the $(S(V_{i,\gamma}), L)$ -Verifier or the $(S(V_{i',\gamma'}), L)$ -Verifier at the bottom of the Edge check gadget. Hence, the exit arcs again have flow of values $S(v)$, $L - S(v)$, $S(w)$, and $L - S(w)$ consecutively, just like the entry arcs.

With these gadgets in hand, we now describe the global structure of the reduction. For each class $V_{i,\gamma}$, we first create a Vertex selector gadget (as in Figure 3).

We then create Edge check gadgets to check, for any $i \in [r]$, that the selected vertices in $V_{i,\gamma}$ for all $\gamma \in [k]$ are adjacent in G . The construction is shown in Figure 4a. We call this the *Triangle gadget* for V_i .

Next, we create Edge check gadgets to check, for any $i \in [r - 1]$, that the selected vertices in $V_{i,\gamma}$ and $V_{i+1,\gamma'}$ for all $\gamma, \gamma' \in [k]$ are indeed adjacent in G . The construction is shown in Figure 4b. We call this the *Square gadget* for V_i and V_{i+1} .



(a) The Triangle gadget for V_1 consists of Edge check gadgets to enforce the selected vertices form a clique in V_1 . Here, $k = 4$. The Vertex selector gadgets for each class $V_{1,\gamma}$ are also shown. The $\alpha_{1,\gamma}$'s denote the amount of flow selected in the corresponding $(S(V_{1,\gamma}), L)$ -Verifiers.

(b) The Square gadget for V_1 and V_2 has Edge check gadgets to enforce the selected vertices form a clique between V_1 and V_2 . Here, $k = 4$.

■ Figure 4

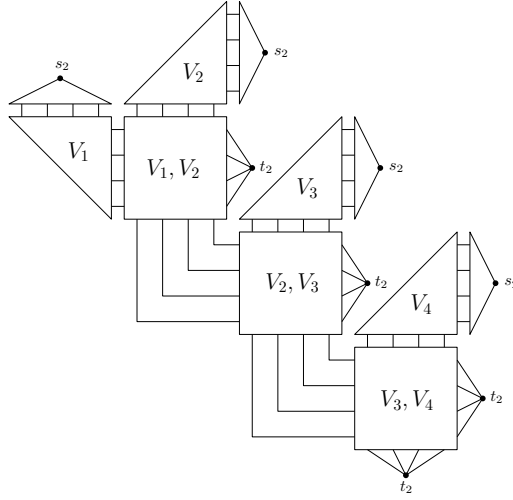
Then, we connect the Vertex selector, Triangle and Square gadgets as in Figure 5. We now set the demand for commodity 1 to the sum of the capacities of the outgoing arcs of s_1 (which is equal to the sum of the capacities of the incoming arcs of t_1). We set the demand for commodity 2 to the sum of the capacities of the outgoing arcs of s_2 (which is equal to the sum of the capacities of the incoming arcs of t_2). This completes the construction.

▷ Claim 3.1 (♣). The constructed graph has pathwidth at most $8k + O(1)$.

▷ Claim 3.2. The given CHAINED MULTICOLOUR CLIQUE instance has a solution if and only if the constructed instance of INTEGER 2-COMMODITY FLOW has a solution.

Proof sketch. For the forward direction, assume there exists a chained multicolour clique W in G . Recall that one vertex is picked per $V_{i,\gamma}$ class by definition and thus W has size rk . If $v \in V_{i,\gamma} \cap W$ for some $i \in [r], \gamma \in [k]$, then in the Vertex selector gadget of $V_{i,\gamma}$, we send $S(v)$ units of flow of commodity 2 to the left and $L - S(v)$ units of flow to the right into the Verifier gadget (see Figure 3). In any Verifier gadget, we route the flow so that it takes the path with capacity equal to the flow. This flow is then routed through all Edge check gadgets of the Triangle and Square gadgets, in the manner presented above in the description of Edge check gadgets. Since W is a chained multicolour clique, the corresponding edge exist in E and the flow can indeed pass through the $(S(E), 2L)$ -Verifier gadget of each Edge check gadget. All flow of commodity 1 is routed through the unused gates in the Verifier gadgets, which is possible as we only use one vertical path per gadget for the flow corresponding to a vertex or edge (see Figure 2). It follows that we use all arcs from s_1 and to t_1 to capacity.

For the other direction, suppose there is a 2-commodity flow in the constructed graph with all arcs from s_1 and s_2 and to t_1 and t_2 used to capacity. Since the constructed graph is acyclic, we can apply induction on its topological ordering to show that flow of commodity 1 never leaves a Verifier gadget downwards. Then the construction of the Verifier gadget ensures that that the amount flow of commodity 2 that passes through it always corresponds to some $\alpha \in S$ for the associated set S of the gadget, and the left and right exit arcs carry α and $L - \alpha$ units of flow of commodity 2 respectively. Now, the arc from s_2 in a Vertex selector



■ **Figure 5** Overview of the complete structure of the reduction for $r = 4$. Triangles represent a structure as in Figure 4a, and squares a structure as in Figure 4b. Directions are not drawn, but clear from Figure 4a and 4b. The labels inside each block (say V_i or V_i, V_{i+1}) denote that flow corresponding to vertices of this set (i.e. V_i or V_i and V_{i+1}) is flowing in a block. Note that all points labelled s_2, t_2 are indeed the same vertex.

gadget for some class $V_{i,\gamma}$ must have L flow of commodity 2 and this must be split in α and $L - \alpha$, with $\alpha \in S(V_{i,\gamma})$. We place the corresponding vertex v in the chained multicoloured clique. In any Edge check gadget, the flow of $\alpha \in S(V_{i,\gamma})$, $L - \alpha$ and $\beta \in S(V_{i',\gamma'}), L - \beta$ combines to a unique sum $\alpha + \beta$ and $2L - (\alpha + \beta)$, and assures that the edge between the corresponding vertices is present. The flow must split back up into α , $L - \alpha$ and β , $L - \beta$ by the unique sum due to the fact that S is a Sidon set. We get that the chosen vertices indeed form a chained multicolour clique. \triangleleft

Finally, using Theorem 2.1 and standard log-space techniques, the constructed graph with its capacities can be built with $O(f(k) + \log n)$ space, for some computable function f . \blacktriangleleft

To show the XALP-hardness of the INTEGER 2-COMMODITY FLOW parameterised by treewidth, we reduce from TREE-CHAINED MULTICOLOUR CLIQUE in a similar, but more involved manner (see Figure 6 for an illustration) and obtain Theorem 1.3.

We now reduce from the case of directed graphs to the case of undirected graphs in a general manner, by modification of a transformation by Even et al. [11, Theorem 4]. In this way, both our hardness results (for parameter pathwidth and for parameter treewidth) can be translated to undirected graphs.

► **Lemma 3.3.** *Let G be a directed graph of an INTEGER 2-COMMODITY FLOW instance with capacities given in unary. Then in logarithmic space, we can construct an equivalent instance of UNDIRECTED INTEGER 2-COMMODITY FLOW with an undirected graph G' with $\text{pw}(G') \leq \text{pw}(G) + O(1)$, $\text{tw}(G') \leq \text{tw}(G) + O(1)$, and unit capacities.*

Proof sketch. Given a directed graph $G = (V, E)$, demands d_1 and d_2 , and capacity function $c : E \rightarrow \mathbb{N}_0$, we construct an instance G' , d'_1 and d'_2 , and $c' : E(G') \rightarrow \{0, 1\}$. To the graph G , we add four new vertices $\bar{s}_1, \bar{s}_2, \bar{t}_1, \bar{t}_2$ as new sources and sinks. We connect \bar{s}_i to s_i and \bar{t}_i to t_i by d_i parallel undirected edges of capacity 1, for each $i \in \{1, 2\}$. Next, for each arc $uv \in E$ of capacity p , we create p parallel undirected edges between u and v of capacity 1 each. Then,

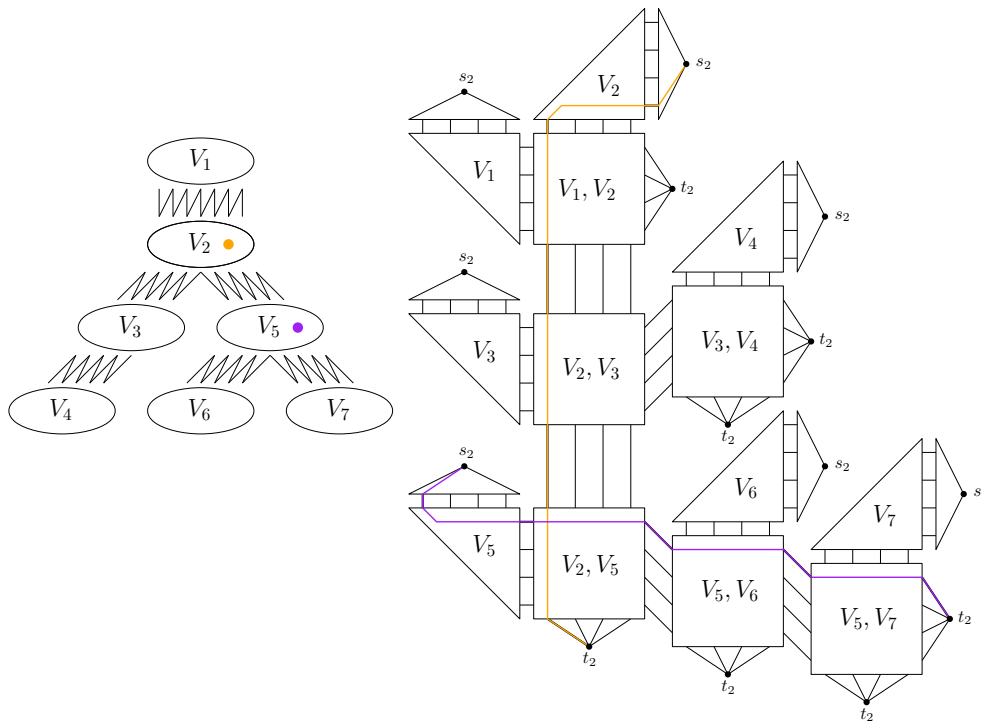


Figure 6 Overview of the structure of the reduction of Theorem 1.3. Left: the structure of the input tree partition. Right: the structure of the reduction. Triangles represent a structure as in Figure 4a, and squares a structure as in Figure 4b. Directions are not drawn, but clear from Figure 4a and 4b. The labels inside each block (say V_i or V_i, V_{i+1}) denote that flow corresponding to vertices of this set (i.e. V_i or V_i and V_{i+1}) is flowing in a block. Note that all points labelled s_2, t_2 are indeed the same vertex. Flow paths corresponding to a selected vertex in V_2 (orange) and one in V_5 (purple) are drawn as an example.

we replace each of these p undirected edges by the Diamond gadget of Figure 7(a). This is the graph G' . In G' , the demands on the two commodities are $d'_1 = d_1 + e^*$ and $d'_2 = d_2 + e^*$, where e^* is the number of edge gadgets in G' (i.e. the sum of all capacities in c).

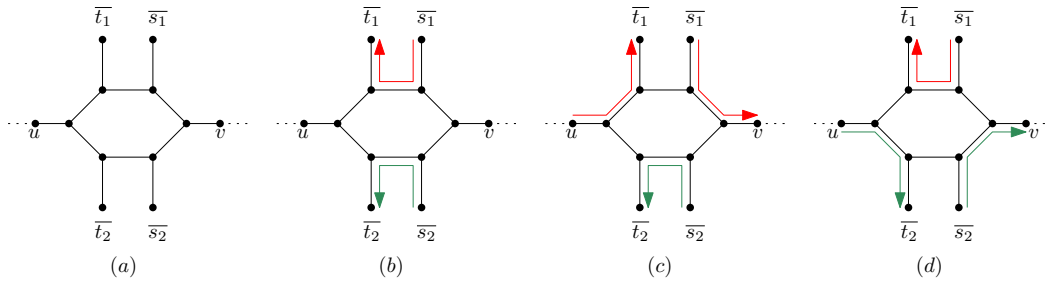
The pathwidth of G' is $\text{pw}(G) + O(1)$ and the treewidth is $\text{tw}(G) + O(1)$. Moreover, the demands d_1 and d_2 are met in G if and only if the demands d'_1 and d'_2 are met in G' . The construction can be done in logarithmic space: while scanning G , we can output G' . ◀

Theorem 1.2 and 1.4 follow immediately from this lemma and Theorem 1.1 and 1.3.

4 Hardness Results – Binary Capacities

Our previous reduction strategy relied heavily on a -Gate gadgets, which have size linear in a , and thus only work in the case a unary representation of the capacities is given. For the case of binary capacities, we can prove stronger results by reducing from 2-PARTITION. However, we need a completely new chain of gadgets and constructions.

We define three different types of (directed) gadgets. Since we use binary capacities, our goal is to double flow in an effective manner. For a given integer a , the a -Doubler gadget receives a flow and sends out $2a$ flow of the same commodity. This gadget is obtained by combining two other gadgets: the a -Switch and the Doubling a -Switch. The a -Switch gadget changes the type of flow; that is, it receives a flow from one commodity, but sends out a



■ **Figure 7** The transformation of Lemma 3.3 from directed to undirected graphs. Every arc uv with capacity c is replaced by c parallel copies of gadget (a), where $\bar{t}_1, \bar{s}_1, \bar{t}_2, \bar{s}_2$ are the same for every gadget, for all arcs. All capacities are 1. The remaining figures illustrate that the gadget either transports no flow (b), a flow of commodity 1 (c), or a flow of commodity 2 (d).

flow from the other commodity. The *Doubling a-Switch* is similar, but sends out $2a$ flow. All three types of gadgets have constant size, even in the binary setting. Moreover, any a -Doubler gadget has pathwidth 5.

A crucial property of the a -Doubler gadget are the following. It has two (vertical) entry arcs and two exit arcs. If the left entry arc carries a units of flow of commodity 2 and the right entry arc carries 0 units of flow of commodity 2, then the left exit arc carries $2a$ units of flow of commodity 2 and the right exit arc carries 0 units of flow of commodity 2. The same property holds with left and right swapped.

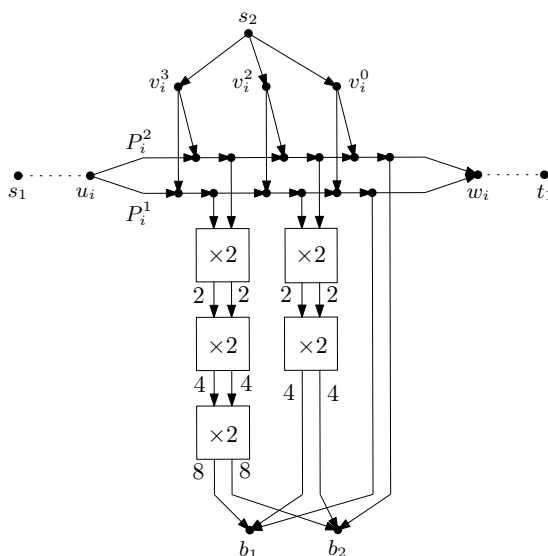
► **Theorem 1.5.** *INTEGER 2-COMMODITY FLOW with capacities given in binary is NP-complete for graphs of pathwidth at most 13.*

Proof sketch. Membership in NP is trivial. To show NP-hardness, we transform from PARTITION. Recall PARTITION problem asks, given positive integers a_1, \dots, a_n , to decide if there is a subset $S \subseteq [n]$ with $\sum_{i \in S} a_i = B$, where $B = \sum_{i=1}^n a_i / 2$. This problem is well known to be NP-complete [23].

Create the sources s_1, s_2 and the sinks t_1, t_2 . Create two vertices b_1, b_2 , both with an arc of capacity B to t_2 .

For each a_i , we build a Binary gadget that either sends a_i units of flow to b_1 or a_i units of flow to a vertex b_2 , in each case of commodity 2. This will indicate whether or not a_i is in the solution set to the PARTITION instance. This gadget is constructed as follows (see Figure 8 for the case when $a_i = 13$). Consider the binary representation a_i^p, \dots, a_i^0 of a_i . That is, $a_i = \sum_{j=0}^p 2^j a_i^j$. For each $j \in [p]$ such that $a_i^j = 1$, we create a column of chained Doubler gadgets. For each $j' < j$, create a $2^{j'}$ -Doubler gadget and identify its entry arcs with the exit arcs of the $2^{j'-1}$ -Doubler gadget (see Figure 8). Then the left exit arc of the (final) 2^{j-1} -Doubler gadget is directed to b_1 , while the right exit arc is directed to b_2 .

Create two directed paths P_i^1, P_i^2 of $2 \sum_{j=0}^p a_i^j$ vertices each (see Figure 8). We consider the vertices of each of these paths in consecutive pairs, one pair for each a_i^j that is equal to 1. For each $j \in [p]$ such that $a_i^j = 1$, create a vertex v_i^j with an arc from s_2 , an arc to the first vertex of the pair on P_i^1 corresponding to a_i^j , and an arc to the first vertex of the pair on P_i^2 corresponding to a_i^j . Then, add an arc from the second vertex of the pair on P_i^1 corresponding to a_i^j to the left entry arc of the 1-Doubler gadget of the j th column of gadgets and an arc from the second vertex of the pair on P_i^2 corresponding to a_i^j to the right entry arc of the 1-Doubler gadget of the j th column of gadgets. Finally, create a vertex u_i



■ **Figure 8** Example of the Binary gadget for $a_i = 13$ and its associated paths P_i^1 and P_i^2 and the ends u_i and w_i . Since $13 = 2^3 + 2^2 + 2^0$, we have a column with three Doublers gadgets (indicated by the square boxes), a column with two Doublers gadgets, and one with no Doublers gadgets. The vertices v_i^3 , v_i^2 and v_i^0 are also drawn. Arcs are labelled by their capacities, but unlabelled arcs have capacity 1. If 1 unit of flow of commodity 1 is sent from s_1 to t_1 , then it must pick one of P_i^1, P_i^2 to go through. Hence, the gadget ensures that either a_i units of flow of commodity 2 are sent to b_1 through the left entry and exit arcs of the Doublers gadgets, or a_i units of flow of commodity 2 are sent to b_2 through the left entry and exit arcs of the Doublers gadgets.

with an arc to the first vertex of P_i^1 and to the first vertex of P_i^2 and create a vertex w_i with an arc from the last vertex of P_i^1 and the last vertex of P_i^2 . This completes the description of the Binary gadget.

We now chain the Binary gadgets. For each $i \in [n - 1]$, add an arc from w_i to u_{i+1} . Add an arc from s_1 to u_1 and from w_n to t_1 . These arcs all have capacity 1.

We set the demand for commodity 1 to the sum of the capacities of the outgoing arcs of s_1 (which is equal to the sum of the capacities of the incoming arcs of t_1). We set the demand for commodity 2 to the sum of the capacities of the outgoing arcs of s_2 (which is equal to the sum of the capacities of the incoming arcs of t_2). This completes the construction.

▷ **Claim 4.1** (♣). The constructed graph has pathwidth at most 13.

▷ **Claim 4.2.** The given PARTITION instance has a solution if and only if the constructed instance of INTEGER 2-COMMODITY FLOW has a solution.

Proof sketch. Let $S \subseteq [n]$ be a solution to the PARTITION instance. For each $i \in [n]$, we do the following. If $i \in S$, then we send flow of commodity 2 from s_2 to b_1 , through left entry and exit arcs of the Doublers gadgets in the Binary gadget corresponding to a_i . To reach this left side of the Doublers gadgets, the flow passes through vertices and arcs of P_i^1 . We can thus send flow of commodity 1 from u_i to w_i via P_i^2 . Otherwise, if $i \notin S$, we send flow of commodity 2 from s_2 to b_2 , through right entry and exit arcs of the Doublers gadgets in the Binary gadget corresponding to a_i . To reach this right side of the Doublers gadgets, the flow passes through vertices and arcs of P_i^2 . We send flow of commodity 1 from u_i to w_i via P_i^1 .

6:14 The Parameterised Complexity of Integer Multicommodity Flow

By the properties of the Doubler gadget, b_1 will receive a_i units of flow of commodity 2 if $i \in S$ and b_2 will receive a_i units of flow of commodity 2 if $i \notin S$. Since S is a solution to PARTITION, both b_1 and b_2 receive B units of flow of commodity 2, which they can then pass on to t_2 . Moreover, we observe that we can send 1 unit of flow from s_1 to t_1 via the paths P_i^1 and P_i^2 , using $i \notin S$ and $i \in S$ respectively.

In the other direction, we see that the flow of commodity 1 starting at u_1 takes a path which is a union of $P_i^{j_i}$ paths, for $i \in [n]$ and $j_i \in \{1, 2\}$. In particular, this flow does not “leak” into any Doubler gadget. Then, similar to the above, let $S \subseteq [n]$ be the set of indices i for which the flow of commodity 2 through the Binary gadget corresponding to a_i arrives at b_1 . We can see that S is a valid solution to the PARTITION instance. \triangleleft

Finally, as each a -Doubler has constant size, the gadget for some a_i has size $O(\log^2(a_i))$, which is polynomial in the input size. Hence, the construction as a whole has size polynomial in the input size. Moreover, it can clearly be computed in polynomial time. \blacktriangleleft

We now reduce from the case of directed graphs to the case of undirected graphs in a general manner. We define a new gadget similar to the one for the unary case. However, we note that there we required a copies of the gadget if the capacity of an arc is a , which is not feasible in the case of binary capacities. Also note that increasing the capacities of the gadget by Even et al. [11, Theorem 4], here Figure 7, invalidates the gadget, as any under-capacity edge would allow flow in the other direction. Hence, we need a different gadget, the Directed edge gadget, which we do not describe in detail here.

► **Lemma 4.3 (♣).** *Let G be a directed graph of an INTEGER ℓ -COMMODITY FLOW instance with a path decomposition of width w , such that each bag contains the sources and sinks of commodities $1, \dots, \ell$. Then in polynomial time, we can construct an equivalent instance of UNDIRECTED INTEGER $\ell + 1$ -COMMODITY FLOW of pathwidth at most $w + 5$.*

By combining Lemma 4.3 and Theorem 1.5, we obtain Theorem 1.6.

5 Algorithm for Parameter Weighted Tree Partition Width

We give an FPT-algorithm for INTEGER ℓ -COMMODITY FLOW parameterised by weighted tree partition width. This algorithm assumes that a tree partition of the input graph is given. There is an algorithm by Bodlaender et al. [4] that for any graph G and integer w , runs in time $\text{poly}(w) \cdot n^2$ and either outputs a tree partition of G of width $\text{poly}(w)$ or outputs that G has no tree partition of width at most w . By some simple tricks, this can be expanded to approximate weighted tree partition width as well, at the expense of a slightly worse polynomial in w . An approximately optimal tree partition of this form would be sufficient as input to our algorithm.

► **Theorem 1.9.** *The INTEGER ℓ -COMMODITY FLOW problem can be solved in time $2^{2^b} n^{O(1)}$, where b is the breadth of a given tree partition of the input graph.*

Proof. We will describe a dynamic-programming algorithm on a given tree partition $(T, (B_x)_{x \in V(T)})$. Let $r \in V(T)$ be some node, that we will designate as the root of the tree T . For convenience, we first attach a node to every leaf, with an empty bag.

We will create a table τ , where every entry is indexed by a node x of the tree partition and a collection \mathbf{f}_x of functions f_x^i , one function for every commodity $i \in [\ell]$. We will refer to \mathbf{f}_x as a flow profile and use the superscript i to refer to the flow function for commodity i in the profile. The function

$$f_x^i : B_{p(x)} \rightarrow [-b, b],$$

where $p(x)$ the parent node of x , indicates for every $v \in B_{p(x)}$ the net difference between the amount of flow of commodity i that v receives from (indicated by a positive value) or sends to (indicated by a negative value) the vertices in the bag B_x , in the current partial solution. That is, $f_x^i(v)$ models the value of $\sum_{u \in B_x} (f^i(uv) - f^i(vu))$, where f denotes the current partial solution. Notice that this sum has value in $[-b, b]$, as the sum over all capacities of edges between bags B_x and $B_{p(x)}$ is at most b . The content of each table entry will be a boolean that indicates whether there exists a partial flow on the graph considered up to x that is consistent with the indices of the table entry.

We will build the table τ , starting at the leaves of the tree, for which we assumed the corresponding bags to be empty sets, and working towards the root. If x is a leaf in the tree partition, we set $\tau[x, \{\emptyset, \dots, \emptyset\}] = \text{True}$, where we denote by \emptyset , the unique function with the empty set as domain. Otherwise, x is some node with children y_1, \dots, y_t . We will group these children y_i in equivalence classes ξ , defined by the equivalence relation $y \sim y'$ if and only if $\tau[y, \mathbf{f}_x] = \tau[y', \mathbf{f}]$ for every flow profile \mathbf{f} . Note that there are at most $2^{b^{\ell(2^b+1)}}$ such equivalence classes, with at most $b^{\ell(2^b+1)}$ possible flow profiles $\mathbf{f}_{y_j} = (f_{y_j}^1, \dots, f_{y_j}^\ell)$ for every child y_j of x .

We will now describe an integer linear program that determines the value of $\tau[x, \mathbf{f}_x]$ for a given flow profile \mathbf{f}_x . We define a variable $X_{\xi, \mathbf{g}}$ as the number of sets in class ξ whose in- and outflow we choose to match flow profile \mathbf{g}^1 . We also define a variable Y_e^i for each edge inside the bag B_x or between B_x and its parent bag, which indicates the flow of commodity i on this edge. We will denote by $N^{\text{in}}(v)$ and $N^{\text{out}}(v)$ the set of in-neighbours and out-neighbours of v , respectively, restricted to $B_x \cup B_{p(x)}$. We now add constraints for the following properties, for every commodity $i \in [\ell]$. Flow conservation for all vertices v in the bag B_x , that are not a sink/source for commodity i :

$$\sum_{u \in N^{\text{in}}(v)} Y_{uv} + \sum_{\xi, \mathbf{g}} X_{\xi, \mathbf{g}} \cdot g^i(v) = \sum_{u \in N^{\text{out}}(v)} Y_{uv}.$$

The flow of commodity i from a source s_i (if $s_i \in B_x$):

$$- \sum_{u \in N^{\text{in}}(s_i)} Y_{us_i} + \sum_{u \in N^{\text{out}}(s_i)} Y_{us_i} - \sum_{\xi, \mathbf{g}} X_{\xi, \mathbf{g}} \cdot g^i(s_i) = d_i$$

The flow of commodity i to a sink t_i (if $t_i \in B_x$):

$$\sum_{u \in N^{\text{in}}(t_i)} Y_{ut_i} - \sum_{u \in N^{\text{out}}(t_i)} Y_{ut_i} + \sum_{\xi, \mathbf{g}} X_{\xi, \mathbf{g}} \cdot g^i(t_i) = d_i$$

The desired flow to a vertex v in the parent bag:

$$\sum_{u \in N^{\text{in}}(v) \setminus B_{p(x)}} Y_{uv} - \sum_{u \in N^{\text{out}}(v) \setminus B_{p(x)}} Y_{uv} = f_x^i(v).$$

Edge capacities and non-negative flow:

$$0 \leq \sum_{i=1}^{\ell} Y_e^i \leq c(e)$$

¹ Throughout the proof, if we sum over pairs ξ, \mathbf{g} , we only sum over flow profiles that are valid for bags in ξ . Alternatively, we can set any invalid $X_{\xi, \mathbf{g}}$ to 0 beforehand.

6:16 The Parameterised Complexity of Integer Multicommodity Flow

The number of flow profiles of each type from each class matches the number of bags in that class:

$$\sum_{g: B_x \rightarrow [-b, b]} X_{\xi, \mathbf{g}} = |\xi|.$$

$X_{\xi, \mathbf{g}}$ must be a non-negative integer:

$$X_{\xi, \mathbf{g}} \in \mathbb{N}_0$$

We then use an algorithm of Frank and Tardos [15, Theorem 5.3] to solve the ILP in time $N^{2.5N+o(N)}$, where N is the number of variables in the ILP. This number is dominated by the number of variables $X_{\xi, \mathbf{g}}$, of which there are $O(2^{b^{\ell(2^b+2)}})$. We thus find a running time of $2^{b^{\ell(2^b+2)}(2.5 \cdot 2^{b^{\ell(2^b+2)}} + o(2^{b^{\ell(2^b+2)}}))}$. If the ILP has a feasible solution, we set $\tau[x, \mathbf{f}_x] = \text{True}$; otherwise, we set $\tau[x, \mathbf{f}_x] = \text{False}$. We solve $b^{\ell(2^b+1)}$ such ILP's per bag in the decomposition and thus find a total running time of $O(2^{2^{b^{3\ell b}}})$.

Once we reach the root bag, we use a similar ILP to compute the flow on the root bag, finding a final solution. We find a total running time of $2^{2^{b^{3\ell b}}} n^{O(1)}$. ◀

Note that with some minor changes to the ILP (flow variables can be negative and there is no distinction between in/out edges), this proof also works in the undirected case, proving Theorem 1.10

6 Conclusions

The parameterised complexity analysis of integer multicommodity flow shows that the problem is already hard for several natural parameterisations, e.g., treewidth and pathwidth, even when there are only two commodities. The XNLP- and XALP-completeness imply that the problems have XP algorithms but which are likely also to use $\Omega(n^{f(k)})$ space by the Slice-wise Polynomial Space Conjecture. Moreover, the XNLP- and XALP-completeness results imply that the problems are W[t]-hard.

We end the paper with some open problems. A number of cases for undirected graphs remain unresolved. We conjecture that for several such cases, the complexity results will be analogue to the directed case. A notable open case is UNDIRECTED INTEGER 2-COMMODITY FLOW, which we conjecture is NP-complete for graphs with a pathwidth bound, but Theorem 1.6 only gives the result with three commodities.

We also conjecture that INTEGER 2-COMMODITY FLOW is fixed parameter tractable with the vertex cover number as parameter, possibly by using a dynamic programming algorithm that only needs to investigate solutions that are “close” to the approximate solution found by Theorem 1.11.

Finally, we believe that the problem may be interesting to investigate on certain graph classes, for example planar graphs of bounded treewidth or in general on graphs of treewidth or pathwidth below the bounds given by our hardness results.

References

- 1 Cynthia Barnhart, Niranjana Krishnan, and Pamela H. Vance. Multicommodity flow problems. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*, pages 2354–2362. Springer US, 2009. doi:10.1007/978-0-387-74759-0_407.
- 2 Hans L. Bodlaender, Gunther Cornelissen, and Marieke van der Wegen. Problems hard for treewidth but easy for stable gonality. In Michael A. Bekos and Michael Kaufmann, editors, *Proceedings 48th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2022*, volume 13453 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2022. doi:10.1007/978-3-031-15914-5_7.
- 3 Hans L. Bodlaender, Gunther Cornelissen, and Marieke van der Wegen. Problems hard for treewidth but easy for stable gonality. *arXiv*, abs/2202.06838, 2022. arXiv:2202.06838.
- 4 Hans L. Bodlaender, Carla Groenland, and Hugo Jacob. On the parameterized complexity of computing tree-partitions. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPICs*, pages 7:1–7:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.IPEC.2022.7.
- 5 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Lars Jaffke, and Paloma T. Lima. XNLP-completeness for parameterized problems on graphs with a linear structure. In Holger Dell and Jesper Nederlof, editors, *Proceedings 17th International Symposium on Parameterized and Exact Computation, IPEC 2022*, volume 249 of *LIPICs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.IPEC.2022.8.
- 6 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Marcin Pilipczuk, and Michal Pilipczuk. On the complexity of problems on tree-structured graphs. In Holger Dell and Jesper Nederlof, editors, *Proceedings 17th International Symposium on Parameterized and Exact Computation, IPEC 2022*, volume 249 of *LIPICs*, pages 6:1–6:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.IPEC.2022.6.
- 7 Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. In *Proceedings 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 193–204. IEEE, 2022. doi:10.1109/FOCS52979.2021.00027.
- 8 Hans L. Bodlaender and Marieke van der Wegen. Parameterized complexity of scheduling chains of jobs with delays. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020*, volume 180 of *LIPICs*, pages 4:1–4:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.IPEC.2020.4.
- 9 Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015. doi:10.1007/s00453-014-9944-y.
- 10 P. Erdős and P. Turán. On a problem of Sidon in additive number theory, and on some related problems. *Journal of the London Mathematical Society*, s1-16(4):212–215, 1941. doi:10.1112/jlms/s1-16.4.212.
- 11 Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976. doi:10.1137/0205048.
- 12 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009. doi:10.1016/j.tcs.2008.09.065.
- 13 Krzysztof Fleszar, Matthias Mnich, and Joachim Spoerhase. New algorithms for maximum disjoint paths based on tree-likeness. *Math. Program.*, 171(1-2):433–461, 2018. doi:10.1007/s10107-017-1199-3.
- 14 Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980. doi:10.1016/0304-3975(80)90009-2.
- 15 A. Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, January 1987. doi:10.1007/BF02579200.

- 16 András Frank. Packing paths, circuits, and cuts – a survey. In Bernhard Korte, László Lovász, Hans Jürgen Prömel, and Alexander Schrijver, editors, *Paths, Flows, and VLSI-Layout*, pages 47–100. Springer-Verlag, Berlin, 1990.
- 17 Tobias Friedrich, Davis Issac, Nikhil Kumar, Nadym Mallek, and Ziena Zeif. Approximate max-flow min-multicut theorem for graphs of bounded treewidth. *arXiv*, abs/2211.06267, 2022. [arXiv:2211.06267](https://arxiv.org/abs/2211.06267).
- 18 Tobias Friedrich, Davis Issac, Nikhil Kumar, Nadym Mallek, and Ziena Zeif. A primal-dual algorithm for multicommodity flows and multicuts in treewidth-2 graphs. In Amit Chakrabarti and Chaitanya Swamy, editors, *Proceedings 25th International Conference on Approximation Algorithms for Combinatorial Optimization Problems and 26th International Conference on Randomization and Computation APPROX/RANDOM 2022*, volume 245 of *LIPICs*, pages 55:1–55:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.APPROX/RANDOM.2022.55.
- 19 Robert Ganian and Sebastian Ordyniak. The power of cut-based parameters for computing edge-disjoint paths. *Algorithmica*, 83(2):726–752, 2021. doi:10.1007/s00453-020-00772-w.
- 20 Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. On structural parameterizations of the edge disjoint paths problem. *Algorithmica*, 83(6):1605–1637, 2021. doi:10.1007/s00453-020-00795-3.
- 21 Venkatesan Guruswami, Sanjeev Khanna, Rajmohan Rajaraman, F. Bruce Shepherd, and Mihalis Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *Journal of Computing and System Sciences*, 67(3):473–496, 2003. doi:10.1016/S0022-0000(03)00066-7.
- 22 George Karakostas. Faster approximation schemes for fractional multicommodity flow problems. *ACM Trans. Algorithms*, 4(1):13:1–13:17, 2008. doi:10.1145/1328911.1328924.
- 23 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 24 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce A. Reed. The disjoint paths problem in quadratic time. *J. Comb. Theory, Ser. B*, 102(2):424–435, 2012. doi:10.1016/j.jctb.2011.07.004.
- 25 Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Berlin, Heidelberg, 2000.
- 26 Takao Nishizeki, Jens Vygen, and Xiao Zhou. The edge-disjoint paths problem is np-complete for series-parallel graphs. *Discret. Appl. Math.*, 115(1-3):177–186, 2001. doi:10.1016/S0166-218X(01)00223-2.
- 27 Michal Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *ACM Trans. Comput. Theory*, 9(4):18:1–18:36, 2018. doi:10.1145/3154856.
- 28 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.
- 29 Detlef Seese. Tree-partite graphs and the complexity of algorithms. In Lothar Budach, editor, *5th International Conference on Fundamentals of Computation Theory, FCT 1985*, volume 199 of *Lecture Notes in Computer Science*, pages 412–421. Springer, 1985. doi:10.1007/BFb0028825.
- 30 F. Bruce Shepherd and Adrian R. Vetta. The inapproximability of maximum single-sink unsplittable, priority and confluent flow problems. *Theory of Computation*, 13(1):1–25, 2017. doi:10.4086/toc.2017.v013a020.
- 31 Jens Vygen. Disjoint paths. Technical Report 94816, Research Institute for Discrete Mathematics, University of Bonn, 1998.

- 32 I-Lin Wang. Multicommodity network flows: A survey, part I: Applications and formulations. *International Journal of Operations Research*, 15(4):145–153, 2018. URL: http://www.orstw.org.tw/ijor/vol15no4/IJOR2018_vol15_no4_p145_p153.pdf.
- 33 I-Lin Wang. Multicommodity network flows: A survey, part II: Solution methods. *International Journal of Operations Research*, 15(4):155–173, 2018. URL: http://www.orstw.org.tw/ijor/vol15no4/IJOR2018_vol15_no4_p155_p173.pdf.
- 34 Xiao Zhou, Syurei Tamura, and Takao Nishizeki. Finding edge-disjoint paths in partial k -trees. *Algorithmica*, 26(1):3–30, 2000. doi:10.1007/s004539910002.

Treewidth Is NP-Complete on Cubic Graphs

Hans L. Bodlaender  
Utrecht University, The Netherlands



Lars Jaffke  
University of Bergen, Norway



Paloma T. Lima  
IT University of Copenhagen, Denmark

Sebastian Ordyniak  
University of Leeds, UK

Ondřej Suchý  
Czech Technical University in Prague,
Czech Republic

Édouard Bonnet  
LIP, ENS Lyon, France

Dušan Knop  
Czech Technical University in Prague,
Czech Republic

Martin Milanič  
FAMNIT and IAM, University of Primorska,
Koper, Slovenia

Sukanya Pandey  
Utrecht University, The Netherlands

Abstract

In this paper, we show that TREEWIDTH is NP-complete for cubic graphs, thereby improving the result by Bodlaender and Thilikos from 1997 that TREEWIDTH is NP-complete on graphs with maximum degree at most 9. We add a new and simpler proof of the NP-completeness of treewidth, and show that TREEWIDTH remains NP-complete on subcubic induced subgraphs of the infinite 3-dimensional grid.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Treewidth, cubic graphs, degree, NP-completeness

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.7

Funding *Dušan Knop and Ondřej Suchý*: Dušan Knop and Ondřej Suchý acknowledge the support of the Czech Science Foundation Grant No. 22-19557S.

Martin Milanič: Martin Milanič acknowledges the support of the Slovenian Research and Innovation Agency (I0-0035, research program P1-0285 and research projects N1-0102, N1-0160, J1-3001, J1-3002, J1-3003, J1-4008, and J1-4084) and the research program CogniCom (0013103) at the University of Primorska.

Sebastian Ordyniak: Sebastian Ordyniak acknowledges support by the Engineering and Physical Sciences Research Council (EPSRC, project EP/V00252X/1).

Acknowledgements This research was conducted in the Lorentz Center, Leiden, the Netherlands, during the workshop *Graph Decompositions: Small Width, Big Challenges*, October 24–28, 2022.

1 Introduction

Treewidth is one of the most studied graph parameters, with many applications for both theoretical investigations as well as for applications. The problem of deciding the treewidth of a given graph, and finding corresponding tree decomposition, single-handedly lead to a plethora of studies, including exact algorithms, algorithms for special graph classes, approximations, upper and lower bound heuristics, parameterised algorithms and more. In this paper, we look at the basic problem to decide, for a given graph G and integer k , whether the treewidth of G is at most k .



© Hans L. Bodlaender, Édouard Bonnet, Lars Jaffke, Dušan Knop, Paloma T. Lima, Martin Milanič, Sebastian Ordyniak, Sukanya Pandey, and Ondřej Suchý;
licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 7; pp. 7:1–7:13



Leibniz International Proceedings in Informatics

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

This problem was shown to be NP-complete in 1987 by Arnborg et al. [1]; their proof also gives NP-completeness on co-bipartite graphs. As the treewidth of a graph (without parallel edges) does not change under subdivision of edges, it easily follows and is well known that TREewidth is NP-complete on bipartite graphs. In 1997, Bodlaender and Thilikos [4] modified the construction of Arnborg et al. [1] and showed that TREewidth remains NP-complete if we restrict the inputs to graphs with maximum degree 9. In this paper, we sharpen this bound of 9 to 3. Our proof uses a simple transformation, whose correctness follows from well-known facts about treewidth and simple insights. We also give a new simple proof of the NP-completeness of TREewidth on arbitrary (and on co-bipartite) graphs. We obtain a number of corollaries of the results, in particular NP-completeness of TREewidth on d -regular graphs for each fixed $d \geq 3$, and for graphs that can be embedded in a 3-dimensional grid.

Our techniques are based on the techniques in [1] and [4] with streamlined and simplified arguments, and some additional new but elementary ideas. As a starting point for the reductions, we use the NP-complete problems CUTwidth on cubic graphs and PATHwidth; the NP-completeness proofs for these were given by Monien and Sudborough [6] in 1987.

This paper is organised as follows. In Section 2, we give basic definitions and some well-known results on treewidth. In Section 3, we give a new simple proof of the NP-completeness of TREewidth on co-bipartite graphs that uses an elementary transformation from pathwidth. Section 4 gives our main result: NP-completeness for TREewidth on cubic graphs (i.e. graphs with each vertex of degree 3). In Section 5, we derive as consequences some additional NP-completeness results: on d -regular graphs for each fixed d and on graphs that can be embedded in a 3-dimensional grid. Some final remarks are made in Section 6.

2 Definitions and preliminaries

Throughout the paper, we denote the number of vertices of the graph G by n . All graphs considered in this paper are undirected. A graph G is d -regular if each vertex has degree d . We say that a graph G is *cubic* if G is 3-regular. If each vertex of G has degree at most 3, we say that G is *subcubic*. All numbers considered are assumed to be integers, and an interval $[a, b]$ denotes the set of integers $\{a, a + 1, a + 2, \dots, b - 1, b\}$. Furthermore, for a positive integer a , we denote by $[a]$ the interval $[1, a]$. A graph G is a *minor* of a graph H , if G can be obtained from H by zero or more vertex deletions, edge deletions, and edge contractions. For a graph G and a set of vertices $A \subseteq V(G)$, we write $G + \text{clique}(A)$ for the graph obtained by adding an edge between each pair of distinct non-adjacent vertices in A , i.e. by turning A into a clique.

A *tree decomposition* of a graph G is a pair (T, β) such that T is a tree and β is a mapping assigning each node x of T to a *bag* $\beta(x) \subseteq V(G)$, satisfying the following conditions: every vertex of G belongs to some bag, for every edge of G there exists a bag containing both endpoints of the edge, and for every vertex v of G , the set of nodes x of T such that $v \in \beta(x)$ induces a connected subtree of T . The *width* of a tree decomposition (T, β) is the maximum, over all nodes x of T , of the value of $|\beta(x)| - 1$. The *treewidth* of a graph G , denoted by $\text{tw}(G)$, is the minimum width of a tree decomposition of G . Path decompositions and pathwidth (denoted by $\text{pw}(G)$) are defined analogously, but with the additional requirement that the tree T is a path.

We use a number of well-known facts about treewidth and tree decompositions.

► **Lemma 1** (Folklore). *Let G be a graph, and (T, β) a tree decomposition of width k of G . Then the following statements hold.*

1. *Let W be a clique in G . Then, there is a node x of T with $W \subseteq \beta(x)$.*
2. *Suppose $v, w \in V(G)$, $\{v, w\} \notin E(G)$. If there is a node x of T , with $v, w \in \beta(x)$, then (T, β) is a tree decomposition of width k of the graph obtained by adding the edge $\{v, w\}$ to G .*
3. *Suppose $W \subseteq V(G)$. Then, there is a node x in T such that when we remove $\beta(x)$ and all incident edges from G , then each connected component of G contains at most $n/2$ vertices of W .*
4. *Let y be a leaf of T , with neighbour y' . If $\beta(y) \subseteq \beta(y')$, then removing y with its bag from the tree decomposition (T, β) yields another tree decomposition of G of width at most k .*
5. *If H is a minor of G , then $\text{tw}(H) \leq \text{tw}(G)$, and $\text{pw}(H) \leq \text{pw}(G)$.*

A graph G is *co-bipartite* if $V(G) = A \cup B$ with A a clique and B a clique (that is, the complement of G is bipartite). The following fact is also well known, and follows implicitly from the proofs of Arnborg et al. [1]. For completeness, we give a proof here.

► **Lemma 2** (See, e.g. [1]). *Let G be a co-bipartite graph, with $V(G) = A \cup B$ where A and B are cliques. Then:*

1. $\text{tw}(G) = \text{pw}(G)$.
2. *G has a path decomposition (P, β) with width equal to $\text{tw}(G)$ such that $A \subseteq \beta(p_1)$ and $B \subseteq \beta(p_r)$, where p_1 and p_r are the two endpoints of P .*

Proof. Suppose (T, β) is a tree decomposition of G of width $\text{tw}(G)$. By Lemma 1(1), there is a node x in T with $A \subseteq \beta(x)$, and a node y in T with $B \subseteq \beta(y)$. Let P be the path from x to y in T .

If T has nodes not in P , then we can apply the following step. Take a leaf z of T , not in P . Let z' be the neighbour of z in T . For each $v \in A \cap \beta(z)$, it holds that $v \in \beta(z')$ as z' is on the path from z to x , and for each $v \in B \cap \beta(z)$, it holds that $v \in \beta(z')$ as z' is on the path from z to y . So, by Lemma 1(4), we can remove z from T and obtain another tree decomposition of G . Repeating this step as long as possible gives the desired result. ◀

The *vertex separation number* of a graph G is denoted by $\text{vsn}(G)$ and defined as the minimum, over all orderings $\sigma = (v_1, \dots, v_n)$ of the vertex set of G , of the maximum, over all $i \in \{1, \dots, n\}$, of the number of vertices v_j such that $j > i$ and v_j has a neighbour in $\{v_1, \dots, v_i\}$. Kinnersley proved the following characterisation of pathwidth.

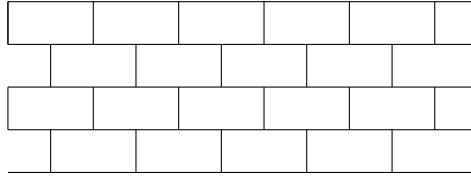
► **Theorem 3** (Kinnersley [5]). *The pathwidth of every graph equals its vertex separation number.*

TREewidth is the following decision problem: Given a graph G and an integer k , is the treewidth of G at most k ? The problems PATHwidth and VERTEX SEPARATION NUMBER are defined analogously.

In 1987, Arnborg, Corneil, and Proskurowski established NP-completeness of TREewidth in the class of co-bipartite graphs [1]. Ten years later, Bodlaender and Thilikos [4] proved that TREewidth is NP-complete on graphs with maximum degree at most 9. Monien and Sudborough [6] proved that VERTEX SEPARATION NUMBER is NP-complete on planar graphs with maximum degree at most 3. Combining this result with Theorem 3 directly shows the following.

► **Theorem 4** (Monien and Sudborough [6]). *PATHwidth is NP-complete on planar graphs with maximum degree at most 3.*

7:4 Treewidth Is NP-Complete on Cubic Graphs



■ **Figure 1** A brick wall with 5 rows and 12 columns.

A well-known type of graphs are the *walls*. A wall with r rows and c columns has $r \times c$ vertices. We refrain from giving a formal definition here, as the concept is clear from Figure 1¹

It is well known that the pathwidth and treewidth of an n by r grid equal $\min\{n, r\}$, see, e.g. [3, Lemmas 87 and 88]. Since any brick wall is a subgraph of a grid, the upper bound also holds for brick walls, and the standard construction gives the following result.

► **Lemma 5** (Folklore). *Let $B_{r,c}$ be a brick wall with r rows and c columns. Then $\text{tw}(B_{r,c}) \leq \text{pw}(B_{r,c}) \leq r$ and there is a path decomposition (P, β) of $B_{r,c}$ of width r with $\beta(p_1)$ the set of vertices on the first column of $B_{r,c}$, and $\beta(p_q)$ the set of vertices on the last column of $B_{r,c}$, where p_1 and p_r are the two endpoints of P .*

A *linear ordering* of a graph G is a bijection $f : V(G) \rightarrow \{1, \dots, n\}$. The *cutwidth* of a linear ordering of G is

$$\max_{i \in [n]} \left| \{ \{v, w\} \in E(G) \mid f(v) \leq i < f(w) \} \right|.$$

The *cutwidth* of a graph G , denoted by $\text{cw}(G)$, is the minimum cutwidth of a linear ordering of G .

The CUTWIDTH problem asks to decide, for a given graph G and integer k , whether the cutwidth of G is at most k . Monien and Sudborough [6] showed that CUTWIDTH is NP-complete on graphs of maximum degree three (using the problem name MINIMUM CUT LINEAR ARRANGEMENT). As their proof does not generate vertices of degree one, and the cutwidth of a graph does not change by subdividing an edge, from their proof, the next result follows.

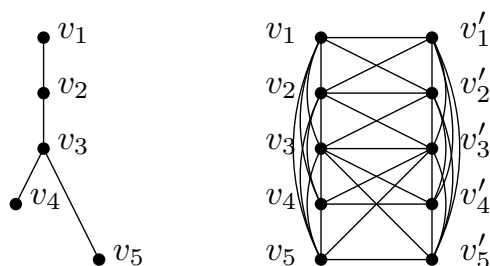
► **Theorem 6** (Monien and Sudborough [6]). *CUTWIDTH is NP-complete on cubic graphs.*

3 A simple proof for co-bipartite graphs

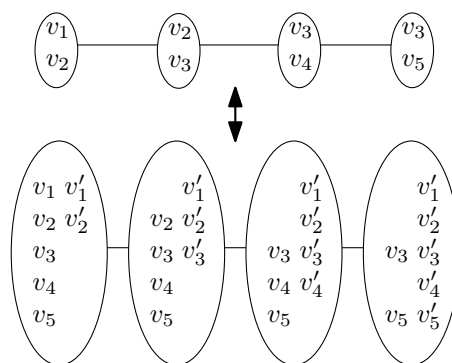
In this section, we give a new simple proof that TREEWIDTH is NP-complete. Our proof borrows elements from the NP-completeness proof from Arnborg et al. [1], but uses an easy transformation from PATHWIDTH.

Let G be a graph. We denote by $F(G)$ the graph obtained from G as follows. The vertices of $F(G)$ consist of two copies v and v' for every $v \in V(G)$; we denote by V and V' the sets $V(G)$ and $\{v' \mid v \in V(G)\}$, respectively. Moreover, the graph $F(G)$ contains for every $v \in V(G)$ an edge between v and v' , and for every edge $\{u, v\} \in E(G)$, it contains one edge between u and v' and one edge between v and u' . Finally, $F(G)$ contains all edges between every pair of distinct vertices in V and every pair of distinct vertices in V' . Note that each of the sets V and V' are cliques in $F(G)$. In particular, G is co-bipartite. An example is given in Figure 2.

¹ The most common notion of wall does not have the vertices of degree one which we see at the bottom left and top right corner of Figure 1. We keep these degree one vertices, for slightly easier notation.



■ **Figure 2** A graph G with $F(G)$.



■ **Figure 3** A path decomposition of the graph G from Figure 2 and the corresponding path decomposition of $F(G)$.

► **Lemma 7.** *Let G be a graph. Then, $\tau w(F(G)) = pw(F(G)) = n + pw(G)$, where $n = |V(G)|$.*

Proof. First, we show that $pw(F(G)) \leq n + pw(G)$. Let $k = pw(G)$. Take a path decomposition (P, β) of G of width k , with $P = (p_1, \dots, p_r)$. Now, let $\gamma(p_i)$ be a set of vertices of $F(G)$ defined as follows:

- For each $v \in V(G)$ such that there is a $j \geq i$ with $v \in \beta(p_j)$, add v to $\gamma(p_i)$.
- For each $v \in V(G)$ such that there is a $j \leq i$ with $v \in \beta(p_j)$, add v' to $\gamma(p_i)$.

An example of this construction, applied to the graphs G and $F(G)$ of Figure 2, is given in Figure 3.

We claim that (P, γ) is a path decomposition of $F(G)$ of width $n + k$. We first verify that (P, γ) is a path decomposition. The first and third conditions of path decompositions are clearly satisfied. Notice that $V \subseteq \gamma(p_1)$, and $V' \subseteq \gamma(p_r)$. So, for each edge in $F(G)$ between two vertices in V , or between two vertices in V' , there is a bag in (P, γ) containing the two endpoints of the edge, namely, the bag corresponding to the node p_1 or p_r , respectively. Consider an edge $\{v, v'\}$ for a vertex $v \in V(G)$. There is a node p_v with $v \in \beta(p_v)$, and therefore $v, v' \in \gamma(p_v)$. Consider an edge $\{v, w'\}$ in $F(G)$, corresponding to an edge $\{v, w\} \in E(G)$. There is a node p_{vw} with $v, w \in \beta(p_{vw})$. Now, $v, v', w, w' \in \gamma(p_{vw})$.

To see that the width is $n + k$, consider some bag $\gamma(p_i)$ and a vertex $v \in V(G)$. There are three possible cases:

1. For each j with $v \in \beta(p_j)$, $j > i$. Now, $v \in \gamma(p_i)$; $v' \notin \gamma(p_i)$.
2. For each j with $v \in \beta(p_j)$, $j < i$. Now, $v' \in \gamma(p_i)$; $v \notin \gamma(p_i)$.
3. If the previous two cases do not hold, then there is $j \leq i$ with $v \in \beta(p_j)$, and $j' \geq i$ with $v \in \beta(p_{j'})$. From the definition of path decompositions, it follows that $v \in \beta(p_i)$. From the construction of γ , we have $v, v' \in \gamma(p_i)$.

7:6 Treewidth Is NP-Complete on Cubic Graphs

In each of the cases, we have one vertex more in $\gamma(p_i)$ than in $\beta(p_i)$, so for each node, the size of its γ -bag is exactly n larger than the size of its β -bag. The claim follows.

Now, suppose the treewidth of G equals ℓ . From Lemma 2(2), it follows that we can assume we have a path decomposition (P, γ) of $F(G)$ of width ℓ , with P having successive bags p_1, p_2, \dots, p_r , and with $V \subseteq \gamma(p_1)$ and $V' \subseteq \gamma(p_2)$.

We now define a path decomposition (P, δ) of G , as follows. For each node x on P , set $\delta(x) = \{v \in V \mid v \in \gamma(x) \wedge v' \in \gamma(x)\}$. (Note that this is the reverse of the operation in the first part of the proof; compare with Figure 3.)

We now verify that (P, δ) is indeed a path decomposition of G . For each vertex v , $\{v, v'\}$ is an edge in $F(G)$, so there is a node x_v with $v, v' \in \gamma(x_v)$, hence $v \in \delta(x_v)$. For each edge $\{v, w\} \in E(G)$, the set $\{v, v', w, w'\}$ forms a clique in $F(G)$, so there is a node x_{vw} with $\{v, v', w, w'\} \subseteq \gamma(x_{vw})$ (see Lemma 1(1)). Hence $v, w \in \delta(x_{vw})$. Finally, for each $v \in V(G)$, the set of nodes x with $v \in \delta(x)$ is the intersection of the nodes with $v \in \gamma(x)$ and the nodes with $v' \in \gamma(x)$; the intersection of connected subtrees is connected, so the third condition in the definition of path (tree) decompositions also holds.

Finally, we show that the width of (P, δ) is $\ell - n$. Consider a vertex v , and $i \in [r]$. There must be i_v with $\{v, v'\} \subseteq \gamma(p_{i_v})$. If $i \leq i_v$, then $v \in \gamma(p_i)$; if $i \geq i_v$, then $v' \in \gamma(p_i)$ (using that $v \in \gamma(p_1)$ and $v' \in \gamma(p_r)$). So, we have $\{v, v'\} \cap \gamma(p_i) \neq \emptyset$.

Now, for each node p_i , $i \in [r]$, for each vertex v , we have that $\gamma(p_i)$ contains both vertices from the set $\{v, v'\}$ when $v \in \delta(p_i)$, and $\gamma(p_i)$ contains exactly one vertex from the set $\{v, v'\}$ when $v \notin \delta(p_i)$. So, $|\gamma(p_i)| = |\delta(p_i)| + n$. As this holds for each bag, we have that the width of (P, γ) is exactly n larger than the width of (P, δ) . It follows that $\text{pw}(G) \leq \text{tw}(F(G)) - n \leq \text{pw}(F(G)) - n$, which shows the result. \blacktriangleleft

Lemma 7, together with the NP-completeness of VERTEX SEPARATION NUMBER [6], and the equivalence between the pathwidth and the vertex separation number (Theorem 3), leads to an alternative simple proof of NP-completeness of TREewidth in the class of co-bipartite graphs.

► **Corollary 8.** *TREewidth is NP-complete on co-bipartite graphs.*

One can obtain a proof of the NP-completeness of TREewidth on graphs with maximum degree five by combining the proof above with the technique of replacing a clique with a wall or grid (as in [4] or in the next section). Instead of this, we give in the next section a proof that reduces from CUTWIDTH and shows NP-completeness of TREewidth on graphs of degree three.

4 Cubic graphs

In this section, we give an NP-completeness proof for TREewidth on cubic graphs. The construction uses a few steps. The first step is a simplified version of the NP-completeness proof from Arnborg et al. [1]; the second step follows the idea of Bodlaender and Thilikos [4] to replace the cliques by grids or walls. After this step, we have a graph with maximum degree 7. In the third step, we replace vertices of degree more than 3 by trees of maximum degree 3, and show that this step does not change the treewidth (it actually can change the pathwidth). The fourth step makes the graph 3-regular by simply contracting over vertices of degree 2.

► **Theorem 9.** *TREewidth is NP-complete on regular graphs of degree 3.*

Proof. We use a transformation from CUTWIDTH on 3-regular graphs.

Let G be an n -vertex 3-regular graph and k an integer. Using a sequence of intermediate steps and intermediate graphs G_1, G_2, G_3 , we construct a 3-regular graph G_4 with the property that G has cutwidth at most k , if and only if G_4 has treewidth at most $3n + k + 2$.

Step 1: From Cutwidth to Treewidth. The first step is a streamlined version of the proof from Arnborg et al. [1]. For each vertex $v \in V(G)$, we take a set $A_v = \{v^1, v^2, v^3\}$ which has three copies of v .

For each edge $e \in E(G)$, we have a set $B_e = \{e^1, e^2\}$, which consists of two vertices that represent the edge.

Let $A = \bigcup_{v \in V(G)} A_v$, and $B = \bigcup_{e \in E(G)} B_e$. We create G_1 by taking $A \cup B$ as vertex set, turning A into a clique, turning B into a clique, and for each pair v, e with v an endpoint of e , adding edges between all vertices in A_v and all vertices in B_e .

▷ **Claim 10.** Let G and G_1 be as above. $\text{tw}(G_1) = \text{pw}(G_1) = \text{cw}(G) + 3n + 2$.

Proof. First, assume G has cutwidth k , and let f be a linear ordering of G of cutwidth k , and denote the i th vertex in the linear ordering as $v_i = f^{-1}(i)$.

Build a path decomposition (P, β) with P the path with nodes p_1, \dots, p_n . For $i \in [n]$, set

$$\begin{aligned} \beta(p_i) = & \{v_j^a \mid j \geq i \wedge a \in \{1, 2, 3\}\} \\ & \cup \{e^b \mid e = \{v_j, v_{j'}\} \in E(G) \wedge \min\{j, j'\} \leq i \wedge b \in [2]\}. \end{aligned}$$

That is, we take the representatives of the vertices v_i, v_{i+1}, \dots, v_n , and all vertices that represent an edge with at least one endpoint in $\{v_1, v_2, \dots, v_i\}$.

We can verify that (P, β) is a path decomposition of G_1 . From the construction, it directly follows that $A \subseteq \beta(p_1)$ and $B \subseteq \beta(p_n)$. For the second condition of path decompositions, it remains to look at edges in G_1 with one vertex of the form v_i^a and one vertex of the form e^b . Necessarily, v_i is an endpoint of e , and now we can note that both vertices are in bag $\beta(p_i)$. From the construction, it directly follows that the third condition of path decompositions is fulfilled.

To show that the width of this path decomposition is at most $k + 3n + 2$, we use an accounting system. Consider $\beta(p_i)$. Give each vertex $v \in V(G)$ three credits, except v_i , which gets six credits. Each edge that ‘‘crosses the cut’’, i.e. it belongs to the set $\{\{v, w\} \in E(G) \mid f(v) \leq i < f(w)\}$, gets one credit. All other edges get no credit. We handed out at most $k + 3n + 3$ credits. We now redistribute these credits to the vertices in $\beta(p_i)$. Each vertex v_j , $j \geq i$, gives one credit to each vertex of the form v_j^a , $a \in \{1, 2, 3\}$. For an edge $e = \{v_j, v_{j'}\}$, with $j < i$ and $j' < i$, the vertices e^1 and e^2 get, respectively, a credit from v_j and $v_{j'}$. For an edge $e = \{v_j, v_{j'}\}$, with $j \leq i < j'$, the vertices e^1 and e^2 get, respectively, a credit from v_j and a credit from e . Now, each vertex and edge precisely spends its credit: a vertex v_j with $j < i$ gives one credit to each of its incident edges, v_i gives one credit to each of its copies v_i^1, v_i^2, v_i^3 , and one credit to each of its incident edges, and v_j with $j > i$ gives one credit to each of its copies v_j^1, v_j^2, v_j^3 . Each vertex in the bag $\beta(p_i)$ gets one credit, so the size of the bag is at most $k + 3n + 3$. As this holds for each bag, the width of the path decomposition is at most $k + 3n + 2$.

Now, assume that we have a tree decomposition (T, γ) of G_1 of width ℓ . By Lemma 1(1), as A and B are cliques, there is a bag p_1 with $A \subseteq \gamma(p_1)$, and a bag p_r with $B \subseteq \gamma(p_r)$. As in the proof of Lemma 2, we can remove all bags not on the path from p_1 and p_r , and

7:8 Treewidth Is NP-Complete on Cubic Graphs

still keep a tree decomposition of G_1 . So, we can assume we have a path decomposition (P, γ) of width at most ℓ of G_1 , where P is a path with successive vertices p_1, p_2, \dots, p_r , and $\gamma(p_1) = A$ and $\gamma(p_r) = B$.

For each $v \in V(G)$, set $g(v)$ to the maximum i such that $\{v^1, v^2, v^3\} \subseteq \beta(p_i)$. (As $\{v^1, v^2, v^3\} \subseteq A \subseteq \beta(p_1)$, $g(v)$ is well defined and in $[r]$.)

Take a linear ordering f of G such that for all $v, w \in V(G)$, $g(v) < g(w) \Rightarrow f(v) < f(w)$. (That is, order the vertices with respect to increasing values of g , and arbitrarily break the ties when vertices have the same value $g(v)$.) We claim that f has cutwidth at most $\ell - 3n - 2$.

Consider a vertex $v \in V(G)$, and suppose $g(v) = i'$. Let e be an edge incident to v . The set $\{v^1, v^2, v^3, e^1, e^2\}$ is a clique in G_1 , so there is an i_e with $\{v^1, v^2, v^3, e^1, e^2\} \subseteq \beta(p_{i_e})$. From the definition of path decompositions and the construction of g , we have $i_e \leq i'$. As $\{e^1, e^2\} \subseteq \beta(p_{i_e}) \cap \beta(p_r)$, we have that $\{e^1, e^2\} \subseteq \beta(p_{i'})$.

Now, consider an $i \in [n]$. Let $v = f^{-1}(i)$ be the i th vertex of the ordering and $C = f^{-1}[i]$ be the first i vertices in the linear ordering. Let E^1 be the set of edges with exactly one endpoint in C , and let E^2 be the set of edges with both endpoints in C . Suppose $g(v) = i'$. We now examine which vertices belong to $\beta(p_{i'})$:

- By definition, v^1, v^2, v^3 .
- For each $w \in V(G) \setminus C$, there is an $i_w \geq i'$ with $\{w^1, w^2, w^3\} \subseteq \beta(p_{i_w})$, hence w^1, w^2 , and w^3 are in $\beta(p_{i'})$. (Use here that these vertices are in $\beta(p_1)$.) The number of such vertices is $3n - 3i$.
- For each edge $e \in E^1 \cup E^2$, from the discussion above it follows that there is an $i_e \leq i'$ with $e^1, e^2 \in \beta(p_{i_e})$, and, as these vertices are in $\beta(p_r)$, we have $\{e^1, e^2\} \subseteq \beta(p_{i'})$.

Thus, the size of $\beta(p_{i'})$ is at least $3n - 3i + 3 + 2 \cdot |E_1| + 2 \cdot |E_2|$. As each vertex in C is incident to exactly three edges, we have $3i = |E_1| + 2 \cdot |E_2|$. Now, $\ell \geq |\beta(p_{i'})| - 1 \geq 3n - 3i + 2 + 2 \cdot |E_1| + 2 \cdot |E_2| = 3n + 2 + |E_1|$. It follows that the size of the cut $\left| \{ \{x, y\} \in E(G) \mid f(x) \leq i < f(y) \} \right| = |E_1| \leq \ell - 3n - 2$. As this holds for each $i \in [n]$, the bound of $\ell - 3n - 2$ on the cutwidth of f follows.

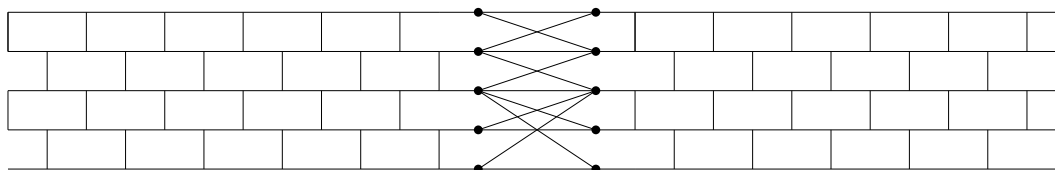
We have thus shown that $\text{pw}(G_1) \leq \text{cw}(G) + 3n + 2$ and that $\text{cw}(G_1) \leq \text{tw}(G_1) - 3n - 2$. Together with the inequality $\text{tw}(G_1) \leq \text{pw}(G_1)$, this proves the claim. \triangleleft

Step 2: The wall construction. In the second step, we use a technique from Bodlaender and Thilikos [4]. We construct a graph G_2 from the graph G_1 by removing the edges between vertices in A and the edges between vertices in B ; then, we add a wall with $3n$ rows and $24n$ columns, and add a matching from the vertices in the last column of the wall to the vertices in A . Similarly, we add another wall with $3n$ rows and $24n$ columns, and add a matching from the vertices in the first column of this wall to the vertices in B .

As applying the wall construction to a graph obtained from the first step would be unwieldy, the example in Figure 4 shows the wall construction applied to the graph from the previous section.

\triangleright **Claim 11.** $\text{tw}(G_1) = \text{pw}(G_1) = \text{tw}(G_2) = \text{pw}(G_2)$. Moreover, there is a path decomposition of G_2 of optimal width with a node x_A with $A \subseteq \beta(x_A)$ and a node x_B with $B \subseteq \beta(x_B)$.

Proof. Suppose we have a tree decomposition (T, β) of G_2 of optimal width k . By Lemma 1(3), there is a node x such that each connected component of $G_2 \setminus \beta(x)$ contains at most $36n^2$ vertices of the left wall. Note that $\beta(x)$ must contain a vertex of each row from the left wall. Suppose not. Each pair of two successive columns in the wall is connected; there are at least $12n - |\beta(x)|$ disjoint pairs of columns which do not contain a vertex from $\beta(x)$. All vertices on



■ **Figure 4** Illustration of the wall construction. Here, it is applied to the graphs from Figure 2, and the number of columns shown is smaller than that in the actual construction.

these columns are connected in $G_2 \setminus \beta(x)$ as they intersect the row without vertices in $\beta(x)$. As the number of vertices in these columns is larger than $36n^2$, since $k \leq |E(G)| = 3n/2$, we have a contradiction.

By Lemma 1(2), (T, β) is also a tree decomposition of the graph obtained from G_2 by adding edges between each pair of vertices in $\beta(x)$. Apply the same step to the right wall. We see that (T, β) is a tree decomposition of width k of a graph that for each pair of rows in the left wall contains an edge between a pair of vertices from these rows, and similarly for the right wall. Now, if we contract each row of the left wall to the neighbouring vertex in A , and contract each row of the right wall to the neighbouring vertex in B , we obtain G_1 as minor: G_1 is a minor of a graph of treewidth k , so has treewidth at most k .

By Lemma 2, $\text{tw}(G_1) = \text{pw}(G_1)$, and there is a path decomposition (P, γ) of G_1 of optimal width ℓ such that $A \subseteq \gamma(p_1)$ and $B \subseteq \gamma(p_q)$, where p_1 and p_q are the endpoints of P .

We can now build a path decomposition of G_2 of the same width ℓ as follows: first, take the successive bags of a path decomposition of the left wall, of width $3n$, where we can end with a bag that contains all vertices of A . Then, we take the bags of (P, γ) . Now, we add a path decomposition of the right wall, of width $3n$, that starts with a bag containing all vertices in B . ◁

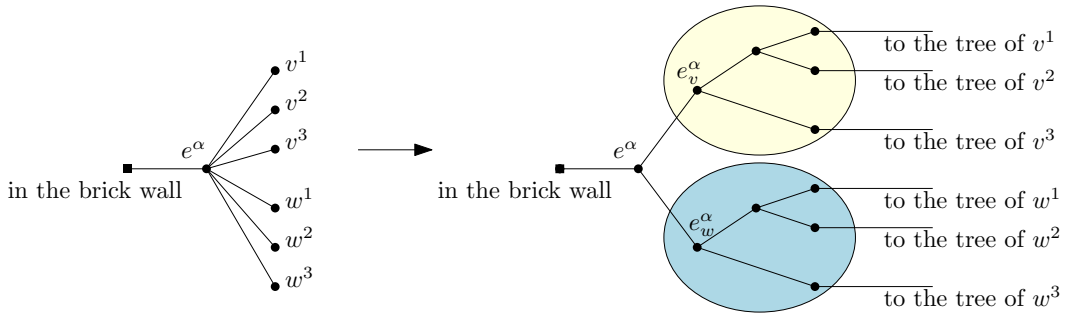
Step 3: Making the graph subcubic. Note that the maximum degree of a vertex in G_2 is seven. A vertex in A has one neighbour in the wall, and six neighbours in B (the vertex it represents has three incident edges, and each is represented by two vertices). Similarly, a vertex in B has degree seven: again, one neighbour in the wall, and six neighbours in A (each endpoint of the edge it represents is represented by three vertices). Vertices in the walls have degree at most three.

Given G_2 , we build a subcubic graph G_3 . We do this by replacing each vertex in A and in B by a tree, and replacing edges to vertices in A and B by edges to leaves or the root of these trees.

For vertices v^α in A (with $v \in V(G)$, $\alpha \in [3]$), we take an arbitrary tree with a root of degree 2, all other internal vertices of degree 3, and six leaves. The root (which we denote by the name of the original vertex v^α) is made adjacent to the neighbour of v^α in the wall.

Each vertex $e^\alpha \in B$ (with $e \in E(G)$, $\alpha \in [2]$) is also replaced by a tree with a root of degree 2, all other internal vertices of degree 3, and six leaves, but here we need to use a specific shape of the tree. Suppose e has endpoints v and w . Figure 5 shows this tree. In particular, note that the root is made adjacent to the neighbour of e^α in the wall, and the leaves that go to the subtrees that represent v are grouped together, and the leaves that go to the subtrees that represent w are grouped together.

Each edge between a vertex v^α in A and a vertex $e^{\alpha'}$ in B now becomes an edge from a leaf of the tree representing v^α , to a leaf of the tree representing $e^{\alpha'}$; $\alpha \in [3]$, $\alpha' \in [2]$. The roots of the trees are made adjacent to a vertex in the wall; this is the same vertex as the wall neighbour of the original vertex in G_2 .



■ **Figure 5** Replacing a vertex e^α from B by a tree; e is here the edge $\{v, w\}$.

▷ **Claim 12.** Suppose $\text{tw}(G_2) \geq 68$. Then $\text{tw}(G_2) = \text{pw}(G_2) = \text{tw}(G_3)$.

Proof. We have already established that $\text{tw}(G_2) = \text{pw}(G_2)$.

First, note that G_2 is a minor of G_3 : we obtain G_2 from G_3 by contracting each of the new trees to its original vertex. By Lemma 1(5), we have $\text{tw}(G_2) \leq \text{tw}(G_3)$.

Suppose we have a path decomposition (P, β) of G_2 of optimal width $\ell = \text{pw}(G_2) = \text{tw}(G_2)$. By Claim 11, we can also assume that there is a bag that contains all vertices in A , and that there is a bag that contains all vertices in B .

For each vertex $v \in V(G)$, we claim that there is a node p_{i_v} with $v^1, v^2, v^3 \in \beta(p_{i_v})$ and $e^1, e^2 \in \beta(p_{i_v})$ for each edge e incident to v . This can be shown as follows. The pair (P, β) is also a path decomposition of the graph $G + \text{clique}(A) + \text{clique}(B)$, obtained from G_2 by adding edges between each pair of vertices in A , and each pair of vertices in B (since there is a bag containing all vertices of A and a bag containing all vertices of B and by Lemma 1(2).) The claim now follows from Lemma 1(1) by observing that these nine vertices (v^1, v^2, v^3 , and e^1, e^2 for each edge incident to v) form a clique in $G + \text{clique}(A) + \text{clique}(B)$.

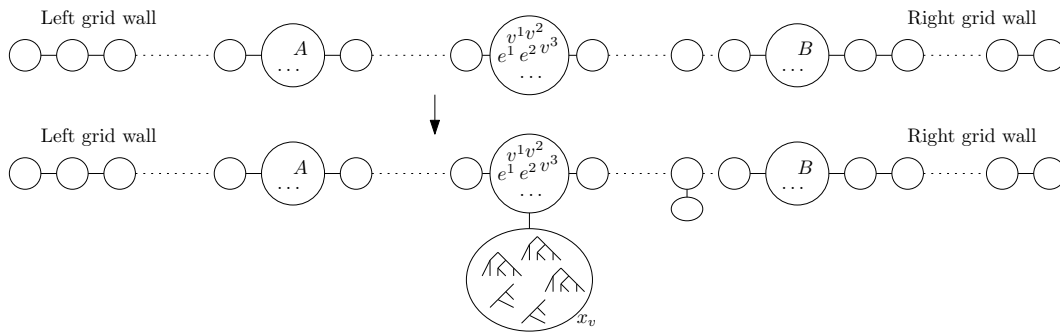
Now, we can construct a tree decomposition of G_3 as follows. Take (P, β) . Replace each vertex in A and each vertex in B by the root of the tree it represents. For each vertex $v \in V(G)$, we add one additional bag to the tree decomposition; this bag becomes a leaf of the tree decomposition. (Note that after this step, we no longer have a path decomposition.)

Consider a vertex $v \in V(G)$. Take a new node x_v , and make x_v adjacent to p_{i_v} in the tree. Let the bag of x_v contain the following vertices: all vertices in the subtrees that represent v^1, v^2, v^3 , for each edge e with v as endpoint the vertices e^1, e_v^1, e^2, e_v^2 , and the descendants of e_v^1 and e_v^2 in the respective subtrees (the vertices in the yellow area in Figure 5, assuming that $e = \{v, w\}$).

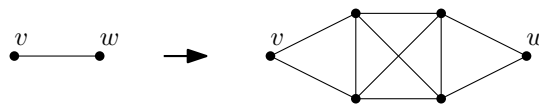
Each vertex in A is represented by a binary tree with a root of degree two and six leaves, so by eleven vertices. For each of the three edges incident to v , we have two subtrees of which we take six vertices each, so the total size of this new bag is $3 \cdot 11 + 3 \cdot 2 \cdot 6 = 69$. One easily verifies that we have a tree decomposition of G_3 , and as the original bags keep the same size when $\ell \geq 68$, we have a tree decomposition of G of width at most ℓ . ◁

By taking a sufficiently large n (e.g. $n \geq 22$ works), we can assume that $\ell \geq 68$.

Step 4: Making the graph 3-regular. The fourth step is simple. Note that when the treewidth of a graph is at least three, the treewidth does not change when we contract a vertex of degree at most two to a neighbour (see [2]), possibly removing parallel edges. We apply this step as long as possible, and let G_4 be the resulting graph. The graph G_4 is a 3-regular graph, and, when $n \geq 22$, its treewidth equals the treewidth of G_1 , which is $\text{cw}(G) + 3n + 2$. As we can construct G_4 in polynomial time, this completes the transformation, and we can conclude that TREEWIDTH is NP-complete on 3-regular graphs. ◀



■ **Figure 6** Illustration of the proof. The path decomposition before and after adding the new node x_v .



■ **Figure 7** Increasing the degree of two adjacent vertices by one.

5 Special cases

In this section, we give two NP-completeness proofs for TREEWIDTH on special graph classes, which follow from minor modifications of the proof of Theorem 9. We first observe that for any fixed $d \geq 4$, TREEWIDTH is NP-complete on d -regular graphs.

► **Proposition 13.** *For each $d \geq 3$, TREEWIDTH is NP-complete on d -regular graphs.*

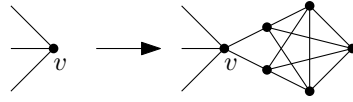
Proof. The result for $d = 3$ was given as Theorem 9.

A small modification of the proof of Theorem 9 gives the result for 4-regular graphs: instead of using a wall, use a grid. At the borders of this grid, we have vertices of degree less than 3. We can avoid these by first contracting vertices of degree 2, and then noting that there is a perfect matching with the vertices of degree 3 at the sides of the grid. Replace each edge in this matching by a small subgraph, as shown in Figure 7. Note that this step increases the degree of v and w by one, while, when the treewidth of G is at least 5, the step will not change the treewidth of the graph.

In the step where we change vertices of degree 7 to vertices of degree 3 by replacing a vertex by a small tree, we instead use a tree with the root having two children, each with three children. These roots are made adjacent to the grid. Now, the roots have degree 3, and we add an arbitrary perfect matching between these root vertices in A , and similarly for B . (Note that in the construction, there is a bag containing all roots for A , and similarly B ; these sets have even size.) This gives the result for $d = 4$.

Consider the following gadget. Take a clique with $d + 1$ vertices, and remove one edge, say $\{x, y\}$, from this clique. For a vertex v in a graph G , add an edge from x to v , and an edge from y to v . See Figure 8.

If G has treewidth at least d , then this step increases the degree of v by 2 without changing the treewidth. Now, if d is odd, we can take an instance of the hardness proof on 3-regular graphs, and add to each vertex of that instance $(d - 3)/2$ copies of this gadget. We obtain an equivalent instance that is d -regular. If d is even, we add $(d - 4)/2$ copies of the gadget to an instance of the hardness proof on 4-regular graphs. ◀



■ **Figure 8** Increasing the degree of a vertex: if $\text{tw}(G) \geq 4$, then the step increases the degree of v from 3 to 5, but does not change the treewidth.

A d -dimensional grid graph is a finite induced subgraph of the infinite d -dimensional grid. Observe that d -dimensional grid graphs have degree at most $2d$, and in particular the 3-dimensional grid graphs have degree at most 6. As a consequence of lowering the degree of hard TREEWIDTH instances from 9 to at most 6, we can show that computing the treewidth of 3-dimensional grid graphs is NP-complete. Since we lowered the degree of hard instances down to at most 3, we can even show the following.

► **Proposition 14.** *TREEWIDTH is NP-complete on subcubic 3-dimensional grid graphs.*

Proof. The argument is simply that every n -vertex (sub)cubic graph admits a subdivision of polynomial size that is a 3-dimensional grid graph. We give a simple such embedding.

We reduce from TREEWIDTH on cubic graphs, which is NP-hard by Theorem 9. Let G be any cubic graph, v_0, v_1, \dots, v_{n-1} its vertices, and $e_1, e_2, \dots, e_{3n/2}$ its edges. We build a subcubic induced subgraph H of the $(6n-1) \times (3n+1) \times 3$ grid that is a subdivision of G . In particular, $\text{tw}(H) = \text{tw}(G)$ and H has $O(n^2)$ vertices and edges, thus we can conclude.

For each $i \in [0, n-1]$, vertex v_i is encoded by the path made by the 5 vertices $(x, 0, 0)$ with $x \in [6i, 6i+4]$. We arbitrarily assign $(6i, 0, 0)$, $(6i+2, 0, 0)$, $(6i+4, 0, 0)$ each with a distinct neighbour of v_i in G , say $v_{i(0)}$, $v_{i(1)}$, $v_{i(2)}$, respectively.

Every edge $e_k = \{v_i, v_j\}$ of G with $i < j$ is encoded in the following way. Let $a, b \in [0, 2]$ be such that $i(a) = j$ and $j(b) = i$. We build a path from $(6i+2a, 0, 0)$ to $(6j+2b, 0, 0)$ with degree-2 vertices, by first adding all the vertices $(6i+2a, y, 0)$ and $(6j+2b, y, 0)$ for $y \in [2k]$, then bridging $(6i+2a, 2k, 0)$ and $(6j+2b, 2k, 0)$ by adding $(6i+2a, 2k, 1)(6i+2a, 2k, 2)(6i+2a+1, 2k, 2)(6i+2a+2, 2k, 2) \dots (6j+2b-1, 2k, 2)(6j+2b, 2k, 2)(6j+2b, 2k, 1)$.

This finishes the construction of H . All of its vertices have degree 2, except the vertices at $(6i+2, 0, 0)$, which have degree 3. It is easy to see that H is a subdivision of G (where each edge gets subdivided at most $12n+5$ times). ◀

We can easily adapt the previous proof to show hardness for finite subcubic (non-induced) subgraphs of the $\infty \times \infty \times 2$ grid.

6 Conclusions

In this paper, we gave a number of NP-completeness proofs for TREEWIDTH. The first proof is an elementary reduction from PATHWIDTH to TREEWIDTH on co-bipartite graphs; while the hardness result is long known, our new proof has the advantage of being very simple, and presentable in a matter of minutes. Our second main result is the NP-completeness proof for TREEWIDTH on cubic graphs, which improves upon the over 25-years-old bound of degree 9.

We end this paper with a few open problems. A long standing open problem is the complexity of TREEWIDTH on planar graphs. While the famous ratcatcher algorithm solves the related BRANCHWIDTH problem in polynomial time [7], it is still unknown whether TREEWIDTH on planar graphs is polynomial time solvable or whether it is NP-complete. Also, no NP-hardness proofs for TREEWIDTH on graphs of bounded genus, or H -minor free

graphs for some fixed H are known. An easier open problem might be the complexity of BRANCHWIDTH for graphs of bounded degree, and we conjecture that BRANCHWIDTH is NP-complete on cubic graphs.

While “our” reductions are simple, the NP-hardness of TREewidth is derived from the NP-hardness of PATHWIDTH or CUTWIDTH. Thus, it would be good to have simple NP-hardness proofs for PATHWIDTH and/or CUTWIDTH, preferably building upon “classic” NP-hard problems like SATISFIABILITY, elementary graph problems like CLIQUE, or BIN PACKING.

The reductions in our hardness proofs increase the parameter by a term linear in n , so shed no light on the parameterised complexity of TREewidth. Hence, it would be interesting to obtain parameterised reductions (i.e. reductions that change k to a value bounded by a function of k), and also aim at lower bounds (e.g. based on the (S)ETH) on the parameterised complexity of TREewidth. It is also not known whether one can obtain a time lower bound of $2^{\Omega(n)}$ for TREewidth.

References

- 1 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k -tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284, 1987. doi:10.1137/0608024.
- 2 Stefan Arnborg and Andrzej Proskurowski. Characterization and recognition of partial 3-trees. *SIAM Journal on Algebraic Discrete Methods*, 7(2):305–314, 1986. doi:10.1137/0607033.
- 3 Hans L. Bodlaender. A partial k -arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.
- 4 Hans L. Bodlaender and Dimitrios M. Thilikos. Treewidth for graphs with small chordality. *Discrete Applied Mathematics*, 79(1-3):45–61, 1997. doi:10.1016/S0166-218X(97)00031-0.
- 5 Nancy G. Kinnersley. The vertex separation number of a graph equals its path-width. *Information Processing Letters*, 42(6):345–350, 1992. doi:10.1016/0020-0190(92)90234-M.
- 6 B. Monien and I. H. Sudborough. Min cut is NP-complete for edge weighted trees. *Theoret. Comput. Sci.*, 58(1-3):209–229, 1988. doi:10.1016/0304-3975(88)90028-X.
- 7 Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994. doi:10.1007/BF01215352.

Stretch-Width

Édouard Bonnet   

Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Julien Duron  

Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Abstract

We introduce a new parameter, called stretch-width, that we show sits strictly between clique-width and twin-width. Unlike the reduced parameters [BKW '22], planar graphs and polynomial subdivisions do not have bounded stretch-width. This leaves open the possibility of efficient algorithms for a broad fragment of problems within Monadic Second-Order (MSO) logic on graphs of bounded stretch-width. In this direction, we prove that graphs of bounded maximum degree and bounded stretch-width have at most logarithmic treewidth. As a consequence, in classes of bounded stretch-width, MAXIMUM INDEPENDENT SET can be solved in subexponential time $2^{\tilde{O}(n^{8/9})}$ on n -vertex graphs, and, if further the maximum degree is bounded, Existential Counting Modal Logic [Pilipczuk '11] can be model-checked in polynomial time. We also give a polynomial-time $O(\text{OPT}^2)$ -approximation for the stretch-width of symmetric 0, 1-matrices or ordered graphs.

Somewhat unexpectedly, we prove that exponential subdivisions of bounded-degree graphs have bounded stretch-width. This allows to complement the logarithmic upper bound of treewidth with a matching lower bound. We leave as open the existence of an efficient approximation algorithm for the stretch-width of unordered graphs, if the exponential subdivisions of all graphs have bounded stretch-width, and if graphs of bounded stretch-width have logarithmic clique-width (or rank-width).

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Design and analysis of algorithms

Keywords and phrases Contraction sequences, twin-width, clique-width, algorithms, algorithmic metatheorems

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.8

Related Version *Full Version:* <https://arxiv.org/abs/2305.12023>

1 Introduction

Various graph classes have bounded *twin-width*¹ such as, for instance, bounded clique-width graphs, proper minor-closed classes, proper hereditary subclasses of permutation graphs, and some expander classes [11]. Low twin-width, together with the witnessing *contraction sequences*, enables parameterized algorithms (that are unlikely in general graphs) for testing if a graph satisfies a first-order sentence [11, 7], and improved approximation algorithms for highly inapproximable packing and coloring problems [7, 4].

However one should not expect a large gain, in the low twin-width regime, as far as (non-parameterized) exact algorithms are concerned. This is because every graph obtained by subdividing (at least) $2\lceil \log n \rceil$ times each edge of an n -vertex graph G has twin-width at most 4 [3]. It was already observed in the 70's that a problem like MAXIMUM INDEPENDENT SET (MIS, for short) remains NP-complete in $2t$ -subdivisions [23]. Furthermore, known reductions [16] combined with the Sparsification Lemma [20], imply that unless the

¹ We refer the reader to Section 2 for the relevant definitions.



© Édouard Bonnet and Julien Duron;

licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 8; pp. 8:1–8:15

Leibniz International Proceedings in Informatics



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

Exponential-Time Hypothesis² (ETH) fails [19] solving MIS in subcubic graphs requires time $2^{\Omega(n)}$. The previous remarks entail that, unless the ETH fails, solving MIS in subcubic graphs of twin-width at most 4 requires time $2^{\Omega(n/\log n)}$.

In contrast, on the significantly less general classes of bounded clique-width not only can MIS be solved in polynomial-time, but a fixed-parameter algorithm solving MSO_1 ³ model checking in time $f(w, |\varphi|)n^{O(1)}$ exists [14, 21], with w the clique-width of the input graph, φ the input sentence, and f some computable function.

In this paper, we start exploring the trade-off between class broadness and algorithmic generality in the zone delimited by bounded clique-width and bounded twin-width. It may seem like the *reduced parameters* [12], where a graph has *reduced* p at most k if it admits a contraction sequence in which all the red graphs have parameter p at most k , are exactly designed to tackle this endeavor. Indeed by definition, twin-width is *reduced* Δ , where Δ is the maximum degree, and it was shown that *reduced maximum connected component size* (under the name of *component twin-width*) is functionally equivalent to clique-width [10]. Between *maximum connected component size* and *maximum degree*, there are several parameters p , such as bandwidth, cutwidth, treewidth+ Δ , whose reduced parameters give rise to a strict [12] hierarchy between bounded clique-width and bounded twin-width. Unfortunately, even reduced bandwidth –the closest to clique-width among the above-mentioned reduced parameters– turns out to be too general in the following sense: the n -subdivision of any n -vertex graph has reduced bandwidth at most 2 [12]. This means, by the arguments of the second paragraph of this introduction, that solving MIS on graphs of bounded reduced bandwidth requires time $2^{\Omega(\sqrt{n})}$, unless the ETH fails, even among graphs of bounded degree. Actually, another fact leading to the same conclusion is that planar graphs have bounded reduced bandwidth [12].

We therefore introduce another parameter, that we call stretch-width⁴ and denote by stw , which, while inspired by reduced parameters, does not fully fit that framework. To a first approximation, *stretch-width* can be thought as *reduced bandwidth* where the bandwidth upper bound on the red graphs have to be witnessed by a single (and fixed) order on the vertex set. Observe indeed that the linear orders witnessing that all the red graphs of the sequence have low bandwidth can, in reduced bandwidth, be very different one from the other. We first show that the family of bounded stretch-width classes strictly contains the family of bounded clique-width classes. Using an upper bound of component twin-width by clique-width [2], we prove that:

► **Theorem 1** (\star). *The stretch-width of any graph is at most twice its clique-width.*

All the statements marked with a \star are only proved in the long version, in appendix.

Then we provide a separating class of bounded stretch-width and unbounded clique-width.

► **Theorem 2** (\star). *There is an infinite family of graphs G with bounded stretch-width and clique-width $\Omega(\log |V(G)|)$.*

As was done for twin-width [9], we give an effective characterization of bounded stretch-width for symmetric 0,1-matrices (or ordered graphs).

► **Theorem 3** (\star). *A class \mathcal{C} of symmetric 0,1-matrices has bounded stretch-width if and only if there is an integer k such that no matrix of \mathcal{C} has a k -wide division.*

² the assumption that there is a $\lambda > 1$ such that n -variable 3-SAT cannot be solved in time $\lambda^n n^{O(1)}$

³ Monadic Second-Order logic, when the second-order variables can only be vertex subsets

⁴ We refer a reader who would already want a formal definition to the start of Section 3.

The *k-wide division* (see definition in Section 4) is a scaled-down version of the *k-rich division* that analogously characterizes matrices of bounded twin-width [9]. Theorem 3 yields a polynomial-time approximation algorithm for the stretch-width of symmetric 0, 1-matrices. More precisely:

► **Theorem 4** (★). *Given an integer k and a symmetric $n \times n$ 0, 1-matrix M , there is an $n^{O(1)}$ -time algorithm that outputs a sequence witnessing that $stw(M) = O(k^3)$ or correctly reports that $stw(M) > k$.*

Compared to the approximation algorithm for the twin-width of a matrix, this is better both in terms of running time (polynomial vs fixed-parameter tractable) and approximation factor (quadratic vs exponential).

Conveniently for the sought algorithmic applications, planar graphs and n^c -subdivisions of n -vertex graphs (for any constant c) both have *unbounded* stretch-width (whereas they have bounded reduced bandwidth if $c \geq 1$). We indeed establish the following upper bound on treewidth, implying that graphs of bounded maximum degree and bounded stretch-width have at most logarithmic treewidth.

► **Theorem 5.** *There is a c such that for every graph G , $tw(G) \leq c\Delta(G)^2 stw(G)^4 \log |V(G)|$.*

We match Theorem 5 with a lower bound. There are graphs with bounded $\Delta + stw$ and treewidth growing as a logarithm of their number of vertices. This is because, as we prove, very long subdivisions of bounded-degree graphs have bounded stretch-width.

► **Theorem 6.** *Every $(\geq n^{2^m})$ -subdivision of every n -vertex m -edge graph G of maximum degree d has stretch-width at most $32(4d + 5)^3$.*

By $(\geq s)$ -subdivision of G , we mean every graph obtained by subdividing each edge of G at least s times. In particular, for every natural k , the n -vertex $k^2 2^{2k(k-1)}$ -subdivision of the $k \times k$ grid has bounded maximum degree (by 4) and stretch-width (by 296352), whereas it has treewidth $k = \Omega(\sqrt{\log n})$. A more careful argument and reexamination of Theorem 6 show that, for some constant c , the n -vertex 2^{ck} -subdivision of the $k \times k$ grid has bounded $\Delta + stw$, and treewidth $k = \Omega(\log n)$ matching the upper bound of Theorem 5.

The proofs of Theorems 5 and 6 involve the notion of *overlap graph* of a graph G whose vertex set is totally ordered by \prec , with one vertex per edge of G , and an edge between two “overlapping edges” of G , that is, two edges ab and cd such that $a \prec c \prec b \prec d$. Using Theorem 3, we show that finding a vertex ordering such that the overlap graph has no large biclique allows to bound the stretch-width.

► **Lemma 7.** *For every ordered graph (G, \prec) and every integer t , if the overlap graph of (G, \prec) has no $K_{t,t}$ subgraph then $stw(G) < 32(2t + 1)^3$.*

Theorem 6 is then derived by designing a long subdivision process that, for ordered graphs of maximum degree d , reduces the bicliques in the overlap graph to a size at most linear in d .

Theorem 5 has direct algorithmic implications for classes of bounded stretch-width.

► **Proposition 8.** *There is an algorithm that solves MAX INDEPENDENT SET in graphs of bounded stretch-width with running time $2^{\tilde{O}(n^{8/9})}$.*

Pilipczuk [22] showed that any problem expressible in Existential Counting Modal Logic (ECML) admits a single-exponential fixed-parameter algorithm in treewidth. In particular, ECML model checking can be solved in polynomial time in any class with logarithmic

treewidth. This logic allows existential quantifications over vertex and edge sets followed by an *arithmetic formula* and a *counting modal formula* that shall be satisfied from every vertex v . The arithmetic formula is a quantifier-free expression that may involve the cardinality of the vertex and edge sets, as well as integer parameters. Counting modal formulas enrich quantifier-free Boolean formulas with $\diamond^S \varphi$, whose semantics is that the current vertex v has a number of neighbors satisfying φ in a prescribed ultimately periodic set S of non-negative integers.

The logic ECML+C gives to ECML the power of also using in the arithmetic formula the number of connected components in subgraphs induced by some vertex or edge sets. There is a Monte-Carlo polynomial-time algorithm for ECML+C in graphs of treewidth at most a logarithm function in their number of vertices [22]. Most NP-hard graphs problems, such as MAXIMUM INDEPENDENT SET, MINIMUM DOMINATING SET, STEINER TREE, etc. are expressible in ECML+C; see [22, Appendix D] for the ECML+C formulation of several examples.

► **Corollary 9.** *Problems definable in ECML (resp. ECML+C) can be solved in polynomial time (resp. randomized polynomial time) in bounded-degree graphs of bounded stretch-width.*

Perspectives. Proposition 8 and Corollary 9 constitute some preliminary pieces of evidence of the algorithmic amenability of classes of bounded stretch-width. We ask several questions. How can the running time of Proposition 8 be improved? (As far as we know, there could be a polynomial-time algorithm for any problem defined in ECML on graphs of bounded stretch-width.) As for twin-width, an approximation algorithm for stretch-width of (unordered) graphs remains open. Lemma 7 gives some hope that this question might be easier than its twin-width counterpart, especially among *sparse* graphs.

Can we lift the bounded-degree requirement in Theorem 6, that is, is there a function f and a constant c , such that the stretch-width of any $(\geq f(n))$ -subdivision of any n -vertex graph is at most c ? Our separating example showing that bounded stretch-width is strictly more general than bounded clique-width (Theorem 2) yields graphs of essentially logarithmic clique-width. Is that true in general?

► **Conjecture 10.** *For every class \mathcal{C} of bounded stretch-width, there is a constant c such that for every n -vertex graph $G \in \mathcal{C}$ the clique-width of G is at most $c \log n$.*

We ask the same question with *rank-width* instead of *clique-width*, which would be more algorithmically helpful. One interpretation of Theorem 5 is that graphs of bounded maximum degree and bounded stretch-width have logarithmic treewidth. Whether the *bounded-degree* constraint can be relaxed to the mere absence of large bipartite complete subgraphs is related to Conjecture 10. A positive answer to Conjecture 10 would indeed imply this relaxation, as Gurski and Wanke have shown that graphs without $K_{t,t}$ subgraphs have treewidth at most their clique-width times $3t$ [18]. A natural future work would consist of using the witness of low stretch-width to get improved algorithms compared to those attained with a witness of low twin-width.

Related work. Our work is in line with twin-width [11], and the reduced parameters [12]. Theorem 1 closely follows a similar proof in the sixth paper of the twin-width series [10], while Theorem 3 is inspired by the fourth paper [9], and notably the so-called *rich divisions*.

Finding the right logic for a given width parameter, or the right width parameter for a given logic has been a common goal ever since Courcelle's and Courcelle-Makowsky-Rotics's theorems [13, 14] relating treewidth with MSO_2 , and clique-width with MSO_1 .

Recent developments (all from 2023) include an efficient model checking of the new logic A&C DN (an extension of Existential MSO₁) on classes of bounded mim-width [5], the new parameter flip-width [26], which could lead to an efficient first-order (FO) model checking in a very general class, and efficient model checking algorithms for FO extensions with disjoint-paths predicates in proper minor-closed classes [17], and in proper topological-minor-closed classes [24].

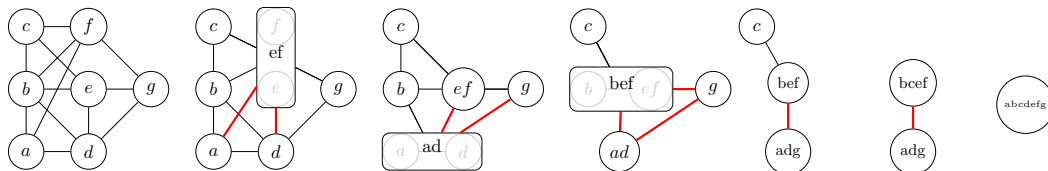
Classes with logarithmic treewidth, although not a priori defined as such, are somewhat rare. To our knowledge, the first such example is the class of triangle-free graphs with no theta (see [25] for the lower bound, and [1], for the upper bound). Another example consists of graphs without $K_{t,t}$ subgraph and bounded induced cycle packing number [6]. We add a new family: graphs of bounded maximum degree and bounded stretch-width. Note that these three families are all incomparable.

2 Preliminaries

For $i \leq j$ two integers, we denote the set of integers that are at least i and at most j by $[i, j]$, and $[i]$ is a short-hand for $[1, i]$. We use the standard graph-theoretic notations. In particular, for a graph G , we denote by $V(G)$ its set of vertices and by $E(G)$ its set of edges. If $S \subseteq V(G)$, the *subgraph of G induced by S* , denoted $G[S]$ is the graph obtained from G by removing the vertices not in S .

2.1 Contraction sequences and twin-width

Twin-width is a graph parameter introduced by Bonnet, Kim, Thomassé, and Watrigant [11]. A possible definition involves the notions of *trigraphs*, *red graphs*, and *contraction sequences*. A *trigraph* is a graph with two types of edges: black (regular) edges and red (error) edges. The *red graph* $\mathcal{R}(H)$ of a trigraph H consists of ignoring its black edges, and considering its red edges as being normal (black) edges. We may say *red neighbor* (or *red neighborhood*) to simply mean a neighbor (or neighborhood) in the red graph. A (vertex) *contraction* consists of merging two (non-necessarily adjacent) vertices, say, u, v into a vertex w , and keeping every edge wz black if and only if uz and vz were previously black edges. The other edges incident to w become red (if not already), and the rest of the trigraph remains the same. A *contraction sequence* of an n -vertex graph G is a sequence of trigraphs $G = G_n, \dots, G_1$ such that G_i is obtained from G_{i+1} by performing one contraction. A *d -sequence* is a contraction sequence in which every vertex of every trigraph has at most d red edges incident to it. In other words, every red graph of the sequence has maximum degree at most d . The *twin-width* of G , denoted by $\text{tw}(G)$, is then the minimum integer d such that G admits a d -sequence. Figure 1 gives an example of a graph with a 2-sequence, i.e., of twin-width at most 2.



■ **Figure 1** A 2-sequence witnessing that the initial graph has twin-width at most 2.

2.2 Partition sequences

Partition sequences yield an equivalent viewpoint to contraction sequences. Instead of dealing with a sequence of trigraphs $G = G_n, \dots, G_1$, we now have a sequence of partitions $\mathcal{P}_n, \dots, \mathcal{P}_1$ of $V(G)$, with $\mathcal{P}_n = \{\{v\} \mid v \in V(G)\}$ and for every $i \in [n-1]$, \mathcal{P}_i is obtained from \mathcal{P}_{i+1} by merging two parts $X, Y \in \mathcal{P}_{i+1}$ into one $(X \cup Y)$. In particular $\mathcal{P}_1 = \{V(G)\}$. Now one can obtain the red graph $\mathcal{R}(G_i)$ of G_i , as the graph whose vertices are the parts of \mathcal{P}_i , and whose edges link two parts $X \neq Y \in \mathcal{P}_i$ whenever there is $u, u' \in X$ and $v, v' \in Y$ such that $uv \in E(G)$ and $u'v' \notin E(G)$. We may call two such parts X, Y *inhomogeneous*. On the contrary, two parts X, Y are *homogeneous* in G when every vertex of X is adjacent to every vertex of Y , or no vertex of X is adjacent to a vertex of Y . We will also denote $\mathcal{R}(G_i)$ by $\mathcal{R}(\mathcal{P}_i)$.

2.3 Separation number

When dealing with treewidth in Section 7 it will more convenient to think of it in terms of the functionally equivalent *separation number*. A *separation* (A, B) of a graph G is such that $A \cup B = V(G)$ and there is no edge between $A \setminus B$ and $B \setminus A$. The *order* of the separation (A, B) is $|A \cap B|$. A separation (A, B) is *balanced* if $\max(|A \setminus B|, |B \setminus A|) \leq \frac{2}{3}|V(G)|$. The *separation number* $\text{sn}(G)$ of G is the smallest integer s such that every subgraph of G admits a balanced separation of order at most s . It is not difficult to show that for every graph G , $\text{sn}(G) \leq \text{tw}(G) + 1$. Dvorák and Norin showed the converse linear dependence:

► **Lemma 11** ([15]). *For every graph G , $\text{tw}(G) \leq 15\text{sn}(G)$.*

Note that if for some positive constant $c < 1$, every subgraph H of G has a separation (A, B) that is *c-balanced*, in the sense that $\max(|A \setminus B|, |B \setminus A|) \leq c|V(H)|$ of order at most s , then every subgraph of G has a balanced separation of order $\lceil \frac{\log c}{\log(2/3)} \rceil \cdot s$. In particular, by Lemma 11, $\text{tw}(G) = O(s)$.

3 Stretch-width

An ordered graph is a pair (G, \prec) where G is a graph and \prec a strict total order on $V(G)$. We write $u \preceq v$ whenever $u \prec v$ or $u = v$. Let (G, \prec) is an ordered graph, and $X \subseteq V(G)$. We now define some objects depending on \prec , but as the order will be clear from the context, we omit it from the corresponding notations.

The minimum and maximum of X along \prec are denoted by $\min(X)$ and $\max(X)$, respectively. The *convex closure* or *span* of X is $\text{conv}(X) := \{v \in V(G) \mid \min(X) \preceq v \preceq \max(X)\}$. Two sets $X, Y \subseteq V(G)$ are *in conflict*⁵, or *X conflicts with Y*, if $\text{conv}(X) \cap \text{conv}(Y) \neq \emptyset$. Note that this does not imply that X and Y themselves intersect, and indeed we will mostly use this notion for two disjoint sets X, Y .

Let now \mathcal{P} be a partition of $V(G)$, $\mathcal{R}(\mathcal{P})$ its red graph, and $X \in \mathcal{P}$. We say that $Y \in \mathcal{P} \setminus \{X\}$ *interferes* with X if Y conflicts with $N_{\mathcal{R}(\mathcal{P})}[X]$. Note that it may well be that Y interferes with X , but not vice versa. The *stretch* of the part $X \in \mathcal{P}$, denoted by $\text{str}(X)$, is then defined as the number of parts in \mathcal{P} interfering with X . In turn, the *stretch* of \mathcal{P} is the maximum over every part $Z \in \mathcal{P}$ of $\text{str}(Z)$. The *stretch-width* of the ordered graph (G, \prec) , denoted by $\text{stw}(G, \prec)$, is the minimum, taken among every partition sequence $\mathcal{P}_n, \dots, \mathcal{P}_1$ of G , of $\max_{i \in [n]} \text{str}(\mathcal{P}_i)$. Finally the *stretch-width* of G , denoted by $\text{stw}(G)$, is the minimum of $\text{stw}(G, \prec)$ taken among every total order \prec on $V(G)$.

⁵ In a similar context in [9], the verb *overlap* was also used. In this paper, we will reserve *overlap* for intersecting intervals (actually edges) that are not nested, notion which we will later use.

4 Matrix characterization

Let us first reinterpret the definition of stretch-width on symmetric (ordered) matrices. A (symmetric) *partition* of a (symmetric) matrix M is a pair $(\mathcal{R}, \mathcal{C})$ such that \mathcal{R} is a partition of the row set of M , $\text{rows}(M)$, \mathcal{C} is a partition of the column set, $\text{columns}(M)$, and \mathcal{C} is symmetric to \mathcal{R} , i.e., two rows r_i and r_j are in the same part if and only if the symmetric columns c_i and c_j are in the same part. Hence, each row part corresponds to a (unique) symmetric column part. A *division* of a (symmetric) matrix M is a partition of M every row (resp. column) part of which is on consecutive rows (resp. columns). Given a row part $R \in \mathcal{R}$, and a column part $C \in \mathcal{C}$, the *zone* $R \cap C$ of M is the submatrix of M with row set R and column set C . The *diagonal zone* of $R \in \mathcal{R}$ is the zone $R \cap C$ where C is the symmetric part of R in columns. A zone is *non-constant* if it contains two distinct entries. A zone of a division may be called *cell*. A *partition sequence* of an $n \times n$ 0,1-matrix M is a sequence $(\mathcal{R}_1, \mathcal{C}_1), \dots, (\mathcal{R}_{n-1}, \mathcal{C}_{n-1})$ where $(\mathcal{R}_1, \mathcal{C}_1)$ is the *finest partition* (with n row parts and n column parts), $(\mathcal{R}_{n-1}, \mathcal{C}_{n-1})$ is the *coarsest partition* (with one row part and one column part), and for every $i \in [n-2]$, $(\mathcal{R}_{i+1}, \mathcal{C}_{i+1})$ is obtained by merging together two row parts (and the symmetric two column parts) of $(\mathcal{R}_i, \mathcal{C}_i)$.

So far, we were following the definitions of [11, 8]. Instead of defining the *error value* which leads to the twin-width of a matrix, we introduce the *stretch value*. The *stretch value* of a row part R of a matrix partition $(\mathcal{R}, \mathcal{C})$ is the number of row parts conflicting with R plus the number column parts conflicting with the union of columns parts C such that $R \cap C$ is non-constant or $R \cap C$ is diagonal. The *stretch value* of a column part is defined symmetrically. The *stretch value* of a partition $(\mathcal{R}, \mathcal{C})$ is the maximum stretch value of a part of $(\mathcal{R}, \mathcal{C})$. Finally, one can define the *stretch-width* of a 0,1-matrix M as the minimum among every partition sequence \mathcal{S} of M of the maximum stretch value among partitions of \mathcal{S} . Observe that for any ordered graph (G, \prec) , the stretch-width of (G, \prec) is equal to the stretch-width of its adjacency matrix.

The following is the counterpart of the so-called *rich divisions* [9] tailored for stretch-width. If R is a set of rows and C a set of columns of a matrix M , we denote by $R \setminus C$ the zone $R \cap (\text{columns}(M) \setminus C)$, that is the submatrix formed by R deprived of the columns of C (and symmetrically for $C \setminus R$).

In a division $(\mathcal{R} = (R_1, \dots, R_n), \mathcal{C} = (C_1, \dots, C_m))$, a row part R_i is k -wide if for every k consecutive columns parts C_j, \dots, C_{j+k-1} containing the symmetric of R_i , $R_i \setminus \cup_{j \leq h \leq j+k-1} C_h$ contains at least k distinct rows. The k -wideness of column parts is defined symmetrically.

A division $(\mathcal{R}, \mathcal{C})$ is k -wide if all its row and column parts are k -wide. The division is k -diagonal if none of the row and column parts is k -wide. Given a set of rows (or columns) X of a matrix M , we keep the notation $\text{conv}(X)$ for the set of rows (or columns) of M with indices between the minimum and the maximum indices of X .

► **Theorem 12.** *For every integer k , if $\text{stw}(M) \leq k$, then M has no $9k$ -wide division.*

Proof. Let $\mathcal{D} = (\mathcal{R}, \mathcal{C})$ be a (symmetric) division of M . Let $(\mathcal{R}'_1, \mathcal{C}'_1), (\mathcal{R}'_2, \mathcal{C}'_2), \dots$ be a (symmetric) partition sequence of M with stretch value at most k (i.e., witnessing that the stretch-width of M is at most k). Let s be the smallest integer for which there is a row part $R' \in \mathcal{R}'_s$ such that $\text{conv}(R')$ contains a row part $R \in \mathcal{R}$ of the division \mathcal{D} (by symmetry, it happens at the same moment for a column part). We will prove that R is not $9k$ -wide.

Let C' be the symmetric of R' in columns. Set $\mathcal{S} := \{T \in \mathcal{R}'_s \mid \text{conv}(T) \cap R \neq \emptyset\}$. Note that \mathcal{S} is the set of row parts of \mathcal{R}'_s that conflicts with R , and that R' is necessarily in \mathcal{S} . As $\text{conv}(R) \subset \text{conv}(R')$, every part in \mathcal{S} conflicts with R' and it should hold that $|\mathcal{S}| \leq k$

because $(\mathcal{R}'_s, \mathcal{C}'_s)$ witnesses that $\text{stw}(M) \leq k$. For each T in \mathcal{S} , we define $C_T := \{c \in \text{columns}(M) \mid c \in \mathcal{C}, C \in \mathcal{C}'_s, \text{ and } C \cap T \text{ is non-constant or } C \text{ is the symmetric of } T\}$. By assumption on the stretch value of $(\mathcal{R}'_s, \mathcal{C}'_s)$, we know that for each $T \in \mathcal{S}$, C_T conflicts with at most k parts of \mathcal{C}'_s . Let C' be the symmetric of R' . As T conflicts with R' , the symmetric of T conflicts with C' . The symmetric of T being contained in C_T , C_T conflicts with C' which means that $\text{conv}(C') \cap \text{conv}(C_T)$ is not empty.

Let us consider $\bigcup_{T \in \mathcal{S}} C_T$. An element A of \mathcal{C}'_s conflicts with $\bigcup_{T \in \mathcal{S}} C_T$ if and only if $\text{conv}(A) \cap \text{conv}(\bigcup_{T \in \mathcal{S}} C_T)$ is non-empty. As for each T of \mathcal{S} , $\text{conv}(C_T) \cap \text{conv}(C')$ is non-empty, there is T_1, T_2 in \mathcal{S} such that the associated two parts C_{T_1} and C_{T_2} (informally the “leftmost” and the “rightmost”) verify

$$\text{conv}(C_{T_1}) \cup \text{conv}(C_{T_2}) \cup \text{conv}(C') = \text{conv}\left(\bigcup_{T \in \mathcal{S}} C_T\right).$$

Note that C_{T_i} can be equal to C' . As C_{T_1} and C_{T_2} conflicts with C' , $C' \cup C_{T_1} \cup C_{T_2}$ conflicts with at most $3k$ parts of \mathcal{C}'_s . Thus $\bigcup_{T \in \mathcal{S}} C_T$ conflicts with at most $3k$ parts of \mathcal{C}'_s .

Observe that, except for C' , every part in \mathcal{C}'_s is covered by the union of two consecutive parts of \mathcal{C} . Part C' itself is covered by the union of three consecutive parts of \mathcal{C} : $\text{conv}(C')$ cannot cover two parts of \mathcal{C} by minimality of s . Thus, overall, each part of \mathcal{C}'_s is covered by the union of at most three consecutive parts of \mathcal{C} . Hence, if $\bigcup_{T \in \mathcal{S}} C_T$ conflicts with $3k$ parts of \mathcal{C}'_s , it is contained in $9k$ consecutive parts of \mathcal{C} , say C_j, \dots, C_{j+9k-1} . Thus for any $T \in \mathcal{S}$, $T \setminus (C_j, \dots, C_{j+9k-1})$ is constant, and so $R \setminus (C_j, \dots, C_{j+9k-1})$ contains at most k different rows. ◀

► **Theorem 13** (\star). *For every integer k , if M does not have a k -wide division, then M admits a sequence $(\mathcal{R}_1, \mathcal{C}_1), (\mathcal{R}_2, \mathcal{C}_2), \dots$ every division of which is $2(k+1)$ -diagonal.*

► **Theorem 14** (\star). *If M admits a sequence of k -diagonal divisions, then $\text{stw}(M) \leq 4k^3$.*

► **Theorem 15**. *If a matrix M does not admit a k -wide division, then $\text{stw}(M) \leq 32(k+1)^3$.*

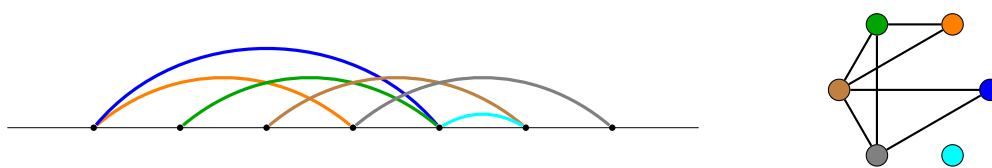
Proof. In fact, M admits a sequence of $2k$ -diagonal divisions by Theorem 13. Applying Theorem 14 on this sequence outputs a witness of stretch-width $4 \cdot (2(k+1))^3 = 32(k+1)^3$. ◀

► **Theorem 4** (\star). *Given an integer k and a symmetric $n \times n$ 0,1-matrix M , there is an $n^{O(1)}$ -time algorithm that outputs a sequence witnessing that $\text{stw}(M) = O(k^3)$ or correctly reports that $\text{stw}(M) > k$.*

5 Overlap graph

Consider an ordered graph (G, \prec) , and think of \prec as a *left-to-right* order (with the smallest vertex being the leftmost one). For any edge $e \in E(G)$, we denote by $L(e)$ (resp. $R(e)$) the left (resp. right) endpoint of e . Given two edges $e, f \in E(G)$, we say that e is *left of* f if $L(e) \preceq L(f)$, and e is *strictly left of* f if $L(e) \prec L(f)$. By extension, we say that $X \subset E(G)$ is *left of* (resp. *strictly left of*) $Y \subset E(G)$ if for every $e \in X$ and $f \in Y$, $L(e) \preceq L(f)$ (resp. $L(e) \prec L(f)$). If u, v are vertices of (G, \prec) , we denote by $[u, v]$ the set of vertices that are, in \prec , at least u and at most v . We also denote by $[\leftarrow, u]$ (resp. $[u, \rightarrow]$) the set of vertices that are at most u (resp. at least u).

We say that two edges e, f are *crossing* if $L(e) \prec L(f) \prec R(e) \prec R(f)$ (or symmetrically) and we denote $e \times f$ this relation. Observe that two edges sharing an endpoint are not crossing. The relation \times is symmetric and anti-reflexive, hence defines an undirected graph on $E(G)$. We denote by $\text{Ov}(G, \prec)$ the graph $(E(G), \times)$. $\text{Ov}(G, \prec)$ is called the *overlap graph* of (G, \prec) ; see Figure 2.



■ **Figure 2** An ordered graph (left) and its overlap graph (right).

We relate the structure of $\text{Ov}_{\prec}(G)$ and the stretch-width of G among bounded-degree graphs, by proving the following theorem:

► **Theorem 16.** *A class \mathcal{C} of ordered graphs of bounded degree has bounded stretch-width if and only if $\{\text{Ov}(G, \prec) \mid G \in \mathcal{C}\}$ does not admit $K_{t,t}$ subgraph, for some integer t .*

The next two lemmas prove the forward implication, by considering a special point in the partition sequence. The last lemma of this section proves the backward implication, using the matrix characterization of Section 4. We say that a $K_{t,t}$ subgraph of $\text{Ov}(G, \prec)$ is *clean* if the sides of the $K_{t,t}$ are $X, Y \subset E(G)$ such that X is strictly left of Y .

► **Lemma 17.** *For every ordered graph (G, \prec) , if $\text{Ov}(G, \prec)$ contains a $K_{t,t}$ as a subgraph, then $\text{Ov}(G, \prec)$ contains a clean $K_{\lfloor t/2 \rfloor, \lfloor t/2 \rfloor}$ subgraph.*

Proof. Assuming that $\text{Ov}(G, \prec)$ has a $K_{t,t}$ subgraph, there is two disjoint sets $X, Y \subset E(G)$ each of size t such that for every $x \in X$ and $y \in Y$, $x \times y$. Let $L(x_1) \prec L(x_2) \prec \dots \prec L(x_t)$ be the elements of X , and $L(y_1) \prec L(y_2) \prec \dots \prec L(y_t)$, the elements of Y . As $x_{\lfloor t/2 \rfloor}$ and $y_{\lfloor t/2 \rfloor}$ are crossing, either $L(x_{\lfloor t/2 \rfloor}) \prec L(y_{\lfloor t/2 \rfloor})$ or $L(y_{\lfloor t/2 \rfloor}) \prec L(x_{\lfloor t/2 \rfloor})$. The sides of the clean $K_{\lfloor t/2 \rfloor, \lfloor t/2 \rfloor}$ are $\{x_1, \dots, x_{\lfloor t/2 \rfloor}\}$ and $\{y_{\lfloor t/2 \rfloor}, \dots, y_t\}$ in the former case, and $\{y_1, \dots, y_{\lfloor t/2 \rfloor}\}$ and $\{x_{\lfloor t/2 \rfloor}, \dots, x_t\}$ in the latter. ◀

► **Lemma 18** (\star). *For any ordered graph (G, \prec) , if $\Delta(G) \leq d$ and $\text{stw}(G, \prec) \leq t$, then $\text{Ov}(G, \prec)$ does not contain $K_{N,N}$ with $N = 4td^2$ as a subgraph.*

► **Lemma 7.** *For every ordered graph (G, \prec) and positive integer N , if $\text{Ov}(G, \prec)$ does not contain $K_{N,N}$ as a subgraph, then $\text{stw}(G, \prec) \leq 32(2N + 1)^3$.*

Proof. Let (G, \prec) be an ordered graph such that $\text{Ov}(G, \prec)$ does not contain $K_{N,N}$ as a subgraph, and let M be the adjacency matrix of (G, \prec) . We prove that $\text{stw}(M) \leq 32(2N + 1)^3$.

Suppose, for the sake of contradiction, that $\text{stw}(M) > 32(2N + 1)^3$. By Theorem 15, there is a $2N$ -wide division $(\mathcal{R} = \{R_1, \dots, R_k\}, \mathcal{C} = \{C_1, \dots, C_k\})$ of M . In particular, for any row R_i , $R_i \setminus C_{i-N+1}, \dots, C_{i+N-1}$ contains more than $2N$ different rows. Let D be the union of the zones $R_i \cap C_j$ such that $|i - j| < N$, that is, the $2N - 1$ “longest” diagonals of zones of the division $(\mathcal{R}, \mathcal{C})$. As, for every $i \in [k]$, the number of distinct rows in $R_i \setminus D$ (resp. distinct columns in $C_i \setminus D$) is at least $2N$, $R_i \setminus D$ (resp. $C_i \setminus D$) contains at least $2N$ 1-entries.

To simplify the coming notations, let denote by $\|M'\|$ the number of 1-entries of any submatrix M' of M . For example, $\|R_i \setminus D\| \geq 2N$. Observe that R_i (resp. C_j) is split by D in at most two sets R_i^{\leftarrow} and R_i^{\rightarrow} (resp. C_j^{\uparrow} and C_j^{\downarrow}), namely, $R_i^{\leftarrow} = \bigcup_{j \leq i-N} R_i \cap C_j$ and $R_i^{\rightarrow} = \bigcup_{j \geq i+N} R_i \cap C_j$.

Observe that for every i, j such that $i + 1 \leq j < i + N$, each 1-entry of R_i^{\rightarrow} (resp. C_i^{\downarrow}) and 1-entry of C_j^{\uparrow} (resp. R_j^{\leftarrow}) correspond to crossing edges in (G, \prec) . As $\text{Ov}(G, \prec)$ does not contain any $K_{N,N}$ subgraph we have, for every i, j such that $i + 1 \leq j < i + N$:

1. $\min(\|R_i^{\rightarrow}\|, \|C_j^{\uparrow}\|) < N$, and
2. $\min(\|C_i^{\downarrow}\|, \|R_j^{\leftarrow}\|) < N$.

Indeed, if the first item does not hold, N 1-entries in R_i^{\rightarrow} and N 1-entries in C_j^{\uparrow} form the two sides of a $K_{N,N}$.

We finally prove by induction on i that, while $2i \leq k$, the property $\|R_{2i}^{\rightarrow}\| > N$, henceforth called (Q_i) , holds. Note that R_0^{\leftarrow} is empty. Thus $\|R_0^{\rightarrow}\| \geq 2N > N$, hence (Q_0) holds. Now assume that (Q_i) holds. By the first item, we have $\|C_{2i+1}^{\uparrow}\| < N$. Thus $\|C_{2i+1}^{\downarrow}\| > N$, since

$$C_{2i+1} \setminus D = C_{2i+1}^{\downarrow} \cup C_{2i+1}^{\uparrow} \text{ and } \|C_{2i+1} \setminus D\| \geq 2N.$$

Symmetrically, by the second item, $\|R_{2i+2}^{\leftarrow}\| < N$, and hence $\|R_{2i+2}^{\rightarrow}\| > N$. Thus (Q_{i+1}) holds. As R_{k-N+1}^{\rightarrow} is empty, (Q_i) can no longer be true when $2i \geq k - N + 1$, a contradiction. Therefore $\text{stw}(M) \leq 32(2N)^3$. ◀

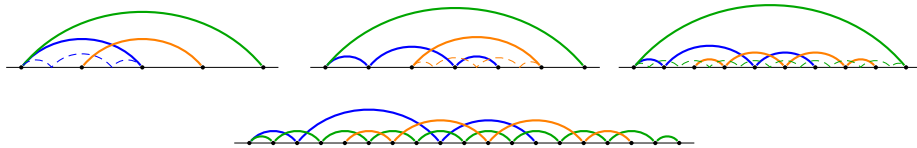
6 Subdivisions

When subdividing the edges of an ordered graph, there is a simple way of updating its vertex ordering without creating larger bicliques in its overlap graph.

► **Lemma 19.** *Let (G, \prec) be an ordered graph, and H be obtained by subdividing an edge of G . There is an order \prec' such that, for every integer t , if $\text{Ov}(G, \prec)$ has no $K_{t,t}$ subgraph, then $\text{Ov}(H, \prec')$ has no $K_{t,t}$ subgraph.*

Proof. Let $e = uv$ be the edge of G subdivided to form H , and let $w \in V(H)$ be the new vertex resulting from this subdivision. The total order \prec' is obtained from \prec , by adding w next to u , say, just to its right. This way $\text{Ov}(H, \prec')$ is simply $\text{Ov}(G, \prec)$ plus an isolated vertex. Indeed the edge $uw \in E(H)$ is an isolated vertex in $\text{Ov}(H, \prec')$, since u and w are consecutive along \prec' , whereas $wv \in E(H)$ crosses the same edges as uv was crossing. ◀

We now define a long subdivision process that is actually “erasing” large bicliques in the overlap graph of a bounded-degree graph. Let uv be an edge of an ordered graph (G, \prec) , with h vertices between u and v , say, $u \prec u_1 \prec u_2 \prec \dots \prec u_h \prec v$. We describe an $h + 1$ -subdivision of uv in (G, \prec) that we call *flattening of uv* . We delete uv , and create $h + 1$ new vertices w_1, \dots, w_{h+1} such that $u \prec w_1 \prec u_1 \prec w_2 \prec u_2 \prec \dots \prec w_h \prec u_h \prec w_{h+1} \prec v$. We then create the edges $uw_1, w_i w_{i+1}$ for every $i \in [h]$, and $w_{h+1}v$. We may say that these *edges stem* from uv . An iterated subdivision of (G, \prec) chooses a total order on the edges of G , and iteratively flattens the edges of G in this order (note that the created edges are *not* flattened themselves); see Figure 3.



■ **Figure 3** An iterated subdivision. Created edges have the color of the edge they stem from.

► **Lemma 20.** *Any iterated subdivision (G', \prec') of an ordered graph (G, \prec) of maximum degree d , is such that $\text{Ov}(G', \prec')$ has no $K_{2d+2, 2d+2}$ subgraph.*

Proof. Assume for the sake of contradiction that $\text{Ov}(G', \prec')$ has a $K_{2d+2, 2d+2}$ subgraph. Then by Lemma 17, $\text{Ov}(G', \prec')$ has a clean $K_{d+1, d+1}$ subgraph. Let X, Y be the two sides of this clean biclique, where X is left of Y . As every vertex of G' (like G) is incident to at most d edges, there is $\{x_1, x_2\} \subseteq X$ and $\{y_1, y_2\} \subseteq Y$ such that $L(x_1) \preceq L(x_2) \prec L(y_1) \prec L(y_2) \prec R(x_i) \prec R(x_{3-i}) \prec R(y_j) \preceq R(y_{3-j})$ with $i, j \in [2]$.

As x_1 and x_2 cross y_1 and y_2 , there is no $i, j \in [2]$ such that x_i and y_j stem from the same edge of G . We can thus assume without loss of generality that the last edge among x_1, x_2, y_1, y_2 to be created is in X (since the argument is symmetric if this happens in Y), i.e., x_i for some $i \in [2]$. When x_i is created, the vertices $L(y_1)$ and $L(y_2)$ already exist and form a non-trivial interval since $L(y_1) \prec L(y_2)$. This contradicts the construction of the iterated subdivision, since x_i jumps over $[L(y_1), L(y_2)]$, when it should have at least created an intermediate vertex in $[L(y_1), L(y_2)]$. ◀

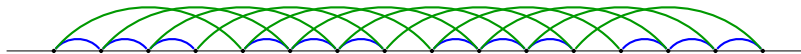
► **Theorem 6.** Every $(\geq n2^m)$ -subdivision of every n -vertex m -edge graph G of maximum degree d has stretch-width at most $32(4d + 5)^3$.

Proof. Let G be any graph of \mathcal{C} with n vertices and m edges, and let G'' be any $(\geq n2^m)$ -subdivision of G . Choose an arbitrary order \prec of $V(G)$. Let (G', \prec') be the iterated subdivision of (G, \prec) , choosing an arbitrary order on the edges G . By Lemma 20, $\text{Ov}(G', \prec')$ has no $K_{2d+2, 2d+2}$ subgraph. Every edge of G is subdivided at most $n2^m$ times by the process of iterated subdivision. By Lemma 19, the edges of G' can be further subdivided to obtain G'' such that $\text{Ov}(G'', \prec'')$ has no $K_{2d+2, 2d+2}$ subgraph, for some vertex ordering \prec'' . Therefore, by Section 5, $\text{stw}(G'', \prec'') \leq 32(4d + 5)^3$, and in particular, $\text{stw}(G'') \leq 32(4d + 5)^3$. ◀

► **Corollary 21.** There are graph classes with bounded stretch-width and maximum degree, and yet unbounded treewidth.

Proof. Consider the family $\Gamma_1, \Gamma_2, \dots$, where Γ_k is the $k^2 2^{2k(k-1)}$ -subdivision of the $k \times k$ -grid, for every positive integer k . The graphs from this family have degree at most 4, and stretch-width at most 296352, but unbounded treewidth since $\text{tw}(\Gamma_k) = k$. ◀

The above argument gives an example of n -vertex graphs with bounded degree and stretch-width, and treewidth $\Omega(\sqrt{\log n})$. We can do better by picking the vertex ordering \prec , and the order on the edges (for the iterated subdivision) more carefully. We simply order



■ **Figure 4** The 4×4 grid ordered row by row, with the horizontal edges in blue, and vertical edges in green.

the grid row by row, and from left to right within each row; see Figure 4. We perform the iterated subdivision of this ordered grid, with the following edge ordering. First we flatten every horizontal edge (in blue), in any order. When this is done, the total number of vertices has less than doubled. Then we flatten every vertical edge (in green) from left to right. It can be observed that, starting from the $k \times k$ grid, we now obtain an iterated subdivision with less than 2^{ck} vertices, for some constant c . Thus, there are n -vertex graphs with bounded degree and stretch-width, and treewidth $\Omega(\log n)$.

7

 Classes with bounded $\Delta + \text{stw}$ have logarithmic treewidth

For any edge e of an ordered graph (G, \prec) , we denote $e^i =]L(e), R(e)[$, the *interior* of e , and $e^o := [\leftarrow, L(e)[\cup]R(e), \rightarrow]$, the *exterior* of e ; note that $L(e)$ and $R(e)$ are neither part of e^i nor of e^o . The *length* of e according to \prec is $\ell(e, \prec) = R(e) - L(e)$. When F is a set of edges we define $\ell(F, \prec)$ to be the maximum length of an edge of F . We say that a set C of vertices is a *c-balanced separator* of G when there is a c -balanced separation (A, B) of G such that $C = A \cap B$. In an ordered graph (G, \prec) a set of vertices C is a *left/right c-balanced separator* if there exists a c -balanced separation (A, B) where A contains the initial interval of length $c \cdot n$, B contains the final interval of length $c \cdot n$ and $C = A \cap B$.

To simplify the notations, if the vertices of (G, \prec) are $v_1 \prec \dots \prec v_n$, we will write $G\langle i, j \rangle$ instead of $G[[v_i, v_j]]$ and $\langle i, j \rangle$ instead of $[v_i, v_j]$.

► **Lemma 22** (\star). *For any ordered n -vertex graph (G, \prec) , if $\Delta(\text{Ov}(G, \prec)) \leq d$ and $\ell(E(G), \prec) \leq \lambda n$, then G admits a left/right $(1/2 - \lambda)$ -balanced separator of size at most $d + 2$.*

► **Lemma 23** (\star). *For every ordered graph (G, \prec) , if $\Delta(\text{Ov}(G, \prec)) \leq d$ then G admits a $1/6$ -balanced separator of size $2d + 4$.*

We say that a set S of edges of (G, \prec) is a *rainbow* if for every pair e, f of S , $e^i \subset f^i$ or $f^i \subset e^i$. Notice that a rainbow induces an independent set in $\text{Ov}(G, \prec)$. When S contains t edges we say that S is a *t-rainbow*, or a rainbow of order t . The following is an application of Dilworth's theorem on permutation graphs. It can be found in [27].

► **Lemma 24** ([27]). *Let (G, \prec) be an ordered graph, such that $\text{Ov}(G, \prec)$ does not contain some K_t . Then, for every vertex v of $V(G)$, for every set F of edges from $[\leftarrow, v[$ to $]v, \rightarrow]$ we have $|F| \leq kt$ where k is the maximum order of a rainbow of F .*

For any rainbow S , we denote by $S^i := \bigcup_{e \in S} e^i$. A *rainbow over v* is a rainbow S contained in the set of edges from $[\leftarrow, v[$ to $]v, \rightarrow]$. A *maximum rainbow over v* is a rainbow of maximum cardinality among the rainbows over v .

► **Lemma 25** (\star). *Let (G, \prec) be an ordered graph such that $\text{Ov}(G, \prec)$ does not contain a $K_{t,t}$ subgraph. Then, if S is a maximum rainbow over $v \in V(G)$, there is a vertex $x \in S^i$ and a set U that separates $[\leftarrow, x[$ from $]x, \rightarrow]$ with U of size $\leq g(t)(\log(\ell(S, \prec)) + 1)$, for a function g such that $g(t) = O(t^4)$.*

► **Theorem 26**. *For any ordered graph (G, \prec) , if $\text{Ov}(G, \prec)$ does not contain any $K_{t,t}$, then G contains a $1/12$ -balanced separator of order at most γt^4 , for some constant γ .*

Proof. Let (G, \prec) be an ordered graph on n vertices, such that $\text{Ov}(G, \prec)$ does not contain any $K_{t,t}$. Let F be the set of edges over a vertex $v \in V(G)$, and let S be a maximum rainbow over v . Denote by e_1, \dots, e_k the edges in S with $e_{j+1}^i \subsetneq e_j^i$. We build $X_{a,b}$ and Y_a in the same way as in the proof of Lemma 25.

Suppose that there are $3t + 2$ edges of S with length in $[n/12, 11n/12]$. We denote these edges by e_i, \dots, e_j , with $j = i + 3t + 2$. We consider the set of edges Z , which is the union of the Y_a and $X_{a,b}$ for $a \in [i, j]$ and $b \in [a - t, a + t]$, and erase it by removing $L_Z = \{L(z) \mid z \in Z\}$. As in the proof of Lemma 25, this operation removes $O(t^4)$ vertices of G . Let $G' = G - L_Z$ be the obtained graph.

Let x, y two vertices, respectively in $]L(e_{i+t}), L(e_{i+2t}[$ and $]R(e_{i+2t}), R(e_{i+t}[$. If one of these sets is empty, say $]L(e_{i+t}), L(e_{i+2t}[$, then any path going from $[\leftarrow, L(e_{i+t})[\cup]R(e_{i+t}), \rightarrow]$ to $]L(e_{i+2t}), R(e_{i+2t}]$ has a vertex in $]R(e_{i+2t}), R(e_{i+t}[$. Then, we only need to consider y in the following (if y also does not exist, the graph is already separated).

By construction, any edge over x is contained in $[L(e_i), L(e_j)]$, or is going from $[\leftarrow, L(e_i)]$ to $[R(e_i), \rightarrow]$. Thus, by applying Lemma 25 on $G_x = G'[\leftarrow, R(e_i)]$ over x , we find a set U_x separating $[\leftarrow, u_x]$ from $[u_x, \rightarrow]$, where u_x is in $[L(e_i), L(e_j)]$. We do the same on the right side, considering $G_y = G[L(e_i), \rightarrow]$ and finding U_y and u_y such that $u_y \in [R(e_j), R(e_i)]$. Set $U_y \cup U_x$ separates $]u_x, u_y[$ from $]u_x, u_y]^c$. As $u_y - u_x$ is between $n/11$ and $n/12$, we found a $n/12$ -balanced separator of G of size at most $1 + 2g(t) \cdot \log \ell(S, \prec)$.

Hence, assume now that for any vertex v of G , the number of edges of length in $[n/12, 11n/12]$ in a maximum rainbow over v is bounded by $3t + 1$. Thus, for every v , the set M_v of edges over v going from $[n/12, v - n/12]$ to $[v + n/12, 11n/12]$ is of size at most $2t(3t + 1)$ by Lemma 24.

Set $u = v_{n/3}$ and $v = v_{2n/3}$. By deleting the set $A = \{L(e) \mid e \in M_u \cup M_v\}$, we get a new ordered graph $H = (G - A, \prec)$. Consider $H_u = H[\leftarrow, 11n/12]$, and $H_v = H[n/12, \rightarrow]$. Then the length of a maximum rainbow over u (resp. v) in H_u (resp. H_v) is at most $n/12$.

Hence we can apply Lemma 25 on H_u over u , and on H_v over v . This yields two sets W_u, W_v of size $O(t^4 \log n)$ and two vertices w_u, w_v such that $|w_u - u|$ (resp. $|w_v - v|$) is at most $n/12$, and such that W_u separates $[\leftarrow, w_u]$ from $[w_u, \rightarrow]$ in H_u (and resp. for v). The set $W_u \cup W_v$ is then separating $]w_u, w_v[$ from $[w_u, w_v]^c$. ◀

► **Theorem 27.** *Let G be any graph on n vertices, such that $\Delta(G) \leq d$ and $\text{stw}(G) \leq t$. Then G contains a $1/12$ -balanced separator of size at most $\gamma(d^2 t)^4 \log n$, for a constant γ .*

Proof. Let G a graph on n vertices, such that $\text{stw}(G) \leq t$ and $\Delta(G) \leq d$. Let \prec be an order such that (G, \prec) has stretch-width t . By Lemma 18, $\text{Ov}(G, \prec)$ does not admit any $K_{4td^2, 4td^2}$ as a subgraph. Hence the Theorem 26 ensures the existence of a $1/12$ -balanced separator of (G, \prec) (hence of G) of size $\gamma(4td^2)^4 \log n$. ◀

By Lemma 11 and Theorem 27, we obtain the bound on the treewidth of a graph of bounded degree and bounded stretch-width.

► **Theorem 5.** *There is a c such that for every graph G , $\text{tw}(G) \leq c\Delta(G)^8 \text{stw}(G)^4 \log |V(G)|$.*


References

- 1 Tara Abrishami, Maria Chudnovsky, Sepehr Hajebi, and Sophie Spirkl. Induced subgraphs and tree-decompositions III. Three-path-configurations and logarithmic tree-width. *Advances in Combinatorics*, 2022.
- 2 Ambroise Baril, Miguel Couceiro, and Victor Lagerkvist. Linear bounds between cliquewidth and component twin-width and applications, 2023. URL: <https://ramics20.lis-lab.fr/slides/slidesAmbroise.pdf>.
- 3 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is NP-complete. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 18:1–18:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICALP.2022.18.
- 4 Pierre Bergé, Édouard Bonnet, Hugues Déprés, and Rémi Watrigant. Approximating highly inapproximable problems on graphs of bounded twin-width. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPICs*, pages 10:1–10:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.STACS.2023.10.

- 5 Benjamin Bergougnoux, Jan Dreier, and Lars Jaffke. A logic-based algorithmic meta-theorem for mim-width. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3282–3304. SIAM, 2023. doi:10.1137/1.9781611977554.ch125.
- 6 Marthe Bonamy, Edouard Bonnet, Hugues Déprés, Louis Esperet, Colin Geniet, Claire Hilaire, Stéphan Thomassé, and Alexandra Wesolek. Sparse graphs with bounded induced cycle packing number have logarithmic treewidth. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3006–3028. SIAM, 2023. doi:10.1137/1.9781611977554.ch116.
- 7 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: Max Independent Set, Min Dominating Set, and Coloring. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPICs*, pages 35:1–35:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.35.
- 8 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. *Combinatorial Theory*, 2(2), 2022. doi:10.5070/C62257876.
- 9 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Toruńczyk. Twin-width IV: ordered graphs and matrices. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 924–937. ACM, 2022. doi:10.1145/3519935.3520037.
- 10 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, and Stéphan Thomassé. Twin-width VI: the lens of contraction sequences. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1036–1056. SIAM, 2022.
- 11 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.
- 12 Édouard Bonnet, O-joung Kwon, and David R. Wood. Reduced bandwidth: a qualitative strengthening of twin-width in minor-closed classes (and beyond). *CoRR*, abs/2202.11858, 2022. arXiv:2202.11858.
- 13 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.
- 14 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.
- 15 Zdenek Dvorák and Sergey Norin. Treewidth of graphs with balanced separations. *J. Comb. Theory, Ser. B*, 137:137–144, 2019. doi:10.1016/j.jctb.2018.12.007.
- 16 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 17 Petr A. Golovach, Giannos Stamoulis, and Dimitrios M. Thilikos. Model-checking for first-order logic with disjoint paths predicates in proper minor-closed graph classes. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3684–3699. SIAM, 2023. doi:10.1137/1.9781611977554.ch141.
- 18 Frank Gurski and Egon Wanke. The tree-width of clique-width bounded graphs without K_n , n . In Ulrik Brandes and Dorothea Wagner, editors, *Graph-Theoretic Concepts in Computer Science, 26th International Workshop, WG 2000, Konstanz, Germany, June 15-17, 2000, Proceedings*, volume 1928 of *Lecture Notes in Computer Science*, pages 196–205. Springer, 2000. doi:10.1007/3-540-40064-8_19.
- 19 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.

- 20 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.
- 21 Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):10:1–10:20, 2008. doi:10.1145/1435375.1435385.
- 22 Michal Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 520–531. Springer, 2011. doi:10.1007/978-3-642-22993-0_47.
- 23 Svatopluk Poljak. A note on stable sets and colorings of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 15(2):307–309, 1974.
- 24 Nicole Schirrmacher, Sebastian Siebertz, Giannos Stamoulis, Dimitrios M. Thilikos, and Alexandre Vigny. Model checking disjoint-paths logic on topological-minor-free graph classes. *CoRR*, abs/2302.07033, 2023. doi:10.48550/arXiv.2302.07033.
- 25 Ni Luh Dewi Sintiar and Nicolas Trotignon. (θ , triangle)-free and (even hole, K_4)-free graphs - part 1: Layered wheels. *J. Graph Theory*, 97(4):475–509, 2021. doi:10.1002/jgt.22666.
- 26 Szymon Toruńczyk. Flip-width: Cops and robber on dense graphs. *CoRR*, abs/2302.00352, 2023. doi:10.48550/arXiv.2302.00352.
- 27 Jakub Černý. Coloring circle graphs. *Electronic Notes in Discrete Mathematics*, 29:457–461, 2007. European Conference on Combinatorics, Graph Theory and Applications. doi:10.1016/j.endm.2007.07.072.

Minimum Separator Reconfiguration

Guilherme C. M. Gomes ✉ 

Department of Computer Science, Federal
University of Minas Gerais, Belo Horizonte, Brazil

Reem Mahmoud ✉

Virginia Commonwealth University, Richmond,
VA, USA

Yoshio Okamoto ✉  

Graduate School of Informatics and Engineer-
ing, The University of Electro-Communications,
Chofu, Japan


Tom C. van der Zanden ✉  

Department of Data Analytics and Digitalisation,
Maastricht University, The Netherlands

Clément Legrand-Duchesne¹

✉  

LaBRI, CNRS, Université de Bordeaux, France

Amer E. Mouawad ✉  

Department of Computer Science, American
University of Beirut, Beirut, Lebanon

Vinicius F. dos Santos ✉  

Department of Computer Science, Federal
University of Minas Gerais, Belo Horizonte, Brazil

Abstract

We study the problem of reconfiguring one minimum s - t -separator A into another minimum s - t -separator B in some n -vertex graph G containing two non-adjacent vertices s and t . We consider several variants of the problem as we focus on both the token sliding and token jumping models. Our first contribution is a polynomial-time algorithm that computes (if one exists) a minimum-length sequence of slides transforming A into B . We additionally establish that the existence of a sequence of jumps (which need not be of minimum length) can be decided in polynomial time (by an algorithm that also outputs a witnessing sequence when one exists). In contrast, and somewhat surprisingly, we show that deciding if a sequence of at most ℓ jumps can transform A into B is an NP-complete problem. To complement this negative result, we investigate the parameterized complexity of what we believe to be the two most natural parameterized counterparts of the latter problem; in particular, we study the problem of computing a minimum-length sequence of jumps when parameterized by the size k of the minimum s - t -separators and when parameterized by the number ℓ of jumps. For the first parameterization, we show that the problem is fixed-parameter tractable, but does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$. We complete the picture by designing a kernel with $\mathcal{O}(\ell^2)$ vertices and edges for the length ℓ of the sequence as a parameter.

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

Keywords and phrases minimum separators, combinatorial reconfiguration, parameterized complexity, kernelization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.9

Related Version *Full Version:* <https://arxiv.org/abs/2307.07782>

Funding *Yoshio Okamoto:* JSPS KAKENHI Grant Numbers JP20K11670, JP20H05795, JP23K10982. *Vinicius F. dos Santos:* FAPEMIG grant APQ-01707-21 and CNPq grants 406036/2021-7 and 312069/2021-9.

Acknowledgements This work started during the Combinatorial Reconfiguration Workshop (CoRe 2022) which was hosted at the Banff International Research Station for Mathematical Innovation and Discovery (BIRS), Alberta, Canada, from May 8–13, 2022. We would like to thank everyone who made this collaboration possible and, in particular, the organizers Daniel Cranston, Marthe Bonamy, Moritz Mühlenthaler, Naomi Nishimura, Nicolas Bousquet, Ryuhei Uehara, and Takehiro Ito for their continuous support of the combinatorial reconfiguration community in general.

¹ Corresponding author



1 Introduction

We study the problem of computing reconfiguration sequences between minimum s - t -separators². A set S of vertices in a graph G is an s - t -separator if vertices s and t are separated in $G - S$, i.e. s and t belong to different components of $G - S$. A *minimum s - t -separator* is an s - t -separator of minimum size. We always let k denote the size of a minimum s - t -separator in G . The token³ jumping (TJ-) (resp. token sliding (TS-)) MINIMUM SEPARATOR RECONFIGURATION (MSR) problem is defined as follows. Given a graph G and minimum s - t -separators A and B , the goal is to determine if there exists a sequence of sets $A = S_1, S_2, \dots, S_r = B$, such that S_i is a minimum s - t -separator, $S_i = (S_{i-1} \setminus \{v\}) \cup \{u\}$ for some $v \in S_{i-1}$, and $u \in V(G) \setminus S_{i-1}$ (resp. $u \in N_{G-S_{i-1}}(v)$) for every $i \in [r] \setminus \{1\}$.

Motivation. Reconfiguration problems arise in various applications and, as a result, have gained considerable attention in recent literature [1, 2, 9, 18]. They appear in power supply problems, such as operating switches in a network to transform between different arrangements of power supply from stations to homes without causing a blackout [17]. They also show up in evolutionary biology, such as in the transformation of genomes via mutations [22]. Moreover, reconfiguration problems contribute to numerous fields of study, such as computational geometry with polygon reconfiguration [5], or statistical physics with the transformation of a particle’s spin system [6]. At the same time, vertex separators are useful in the factorization of sparse matrices [23], as well as, partitioning hypergraphs [19]. They also lend themselves to problems in cyber security and telecommunication [20], bioinformatics and computational biology [15], and many divide-and-conquer graph algorithms [12]. Given the importance of vertex separators, we believe that it is a natural question to study the problem of reconfiguration between different vertex separators.

Related work. Gomes, Nogueira, and dos Santos [16] initiated the study of the problem of computing reconfiguration sequences between s - t -separators, A and B , without restricting the size of the separators (to minimum). We call the corresponding problem VERTEX SEPARATOR RECONFIGURATION (VSR). They show that for token sliding, checking if A can be transformed to B , i.e., VERTEX SEPARATOR RECONFIGURATION, is a PSPACE-complete problem even on bipartite graphs. In contrast, under the token jumping model the problem becomes NP-complete for bipartite graphs.

Our results. Unlike the VSR problem, the requirement in the MSR problem that the separators in the reconfiguration sequence must be minimum introduces a lot of structure. In particular, we can rely on the duality between minimum separators and disjoint paths, observing that tokens are always constrained to move on a set of disjoint s - t -paths, which we call *canonical paths*. Using this property, we prove that, in an (optimal) solution, tokens always move “forward” towards their target locations and we never need to take a step back. This immediately prevents the problems from being PSPACE-complete since this gives a (polynomial) bound on the length of a solution. In fact, the “always-forward” property immediately implies a greedy algorithm that decides whether we can reconfigure one s - t -separator into another or not for both the token sliding and token jumping models. We then

² It is important to note that graphs may have an exponential number of minimum s - t -separators as otherwise the problem is trivial.

³ The notion of tokens is intentionally kept abstract as tokens can represent any type of agents.

turn our attention to finding shortest reconfiguration sequences. While TS-MSR is still solvable in polynomial-time, finding an optimal solution for the TJ-MSR problem is shown to be NP-complete by a reduction from VERTEX COVER; finding the largest set of vertices that can be “skipped” by jumping over them is “similar” to finding a minimum vertex cover.

We give a complete characterization of the (parameterized) complexity of the TJ-MSR problem for its natural parameterizations. In particular, we complement our NP-hardness result by showing that the problem of finding a shortest sequence of token jumps is fixed-parameter tractable when parameterized by k , the size of a minimum separator; this is accomplished by further exploiting the structure imposed by the separators’ minimality as yes-instances have pathwidth bounded by $\mathcal{O}(k)$. Unfortunately, unless $\text{NP} \subseteq \text{coNP/poly}$, the problem admits no polynomial kernel under this parameterization. Finally, we show that if we parameterize the problem by the length of the reconfiguration sequence, ℓ , then we obtain a kernel with $\mathcal{O}(\ell^2)$ vertices and edges.

2 Preliminaries

We denote the set of natural numbers by \mathbb{N} and, for $n \in \mathbb{N}$, we let $[n] = \{1, 2, \dots, n\}$. We only deal with finite simple undirected graphs. We let $V(G)$ and $E(G)$ denote the vertex set and edge set of graph G , respectively. The *open neighborhood* of a vertex v is denoted by $N_G(v) = \{u \in V(G) \mid \{u, v\} \in E(G)\}$ and the *closed neighborhood* by $N_G[v] = N_G(v) \cup \{v\}$. For a set $S \subseteq V(G)$ of vertices, we define $N_G(S) = \bigcup_{v \in S} N_G(v) \setminus S$ and $N_G[S] = N_G(S) \cup S$; if the context is clear, we omit the subscript G . The subgraph of G *induced* by S is denoted by $G[S]$, where $G[S]$ has vertex set S and edge set $\{\{u, v\} \in E(G) \mid u, v \in S\}$; we also define $G - S = G[V(G) \setminus S]$. A *walk* of length q from v_0 to v_q in G is a vertex sequence v_0, \dots, v_q such that $\{v_i, v_{i+1}\} \in E(G)$ for all $i \in \{0, \dots, q-1\}$. It is a *path* if all vertices are distinct. An *s-t-path* is one with endpoints s and t . Two *s-t-paths* P_1 and P_2 are (*internally*) *disjoint* if $V(P_1) \cap V(P_2) = \{s, t\}$. The following is a celebrated theorem attributed to Menger [21] and later generalized and made algorithmic by Ford and Fulkerson [13].

► **Theorem 1.** *The size of a minimum s-t-separator is equal to the maximum number of pairwise internally disjoint s-t-paths, which can be computed in polynomial time.*

Parameterized complexity. A *parameterized problem* Q is a subset of $\Sigma^* \times \mathbb{N}$, where the second component denotes the *parameter*. A parameterized problem is *fixed-parameter tractable* with respect to a parameter κ , FPT for short, if there exists an algorithm to decide whether $(x, \kappa) \in Q$ in time $f(\kappa) \cdot |x|^{\mathcal{O}(1)}$, where f is a computable function. We say that two instances are *equivalent* if they are both yes-instances or both no-instances. A *kernelization algorithm*, or a *kernelization* for short, is a polynomial-time algorithm that reduces an input instance (x, κ) into an equivalent instance (x', κ') such that $|x'|, \kappa' \leq f(\kappa)$, for some computable function f . Such x' is called a *kernel*. Every fixed-parameter tractable problem admits a kernel, however, possibly of exponential or worse size. For efficient algorithms it is therefore most desirable to obtain kernels of polynomial, or even linear size. We refer to the textbooks [7, 10] for extensive background on parameterized complexity.

3 Preprocessing and general observations

Let (G, s, t, A, B) denote an instance of MINIMUM SEPARATOR RECONFIGURATION, where A and B are minimum *s-t-separators* of size k . The reconfiguration model, i.e., jumping vs. sliding, will be clear from context. We begin by making some general observations (that

9:4 Minimum Separator Reconfiguration

hold for both models) about the structure of sequences of minimum s - t -separators, which we make extensive use of. We also describe some preprocessing operations: we assume they have been applied on every instance in the rest of the paper. We begin by introducing the notion of canonical paths, which describe the possible locations for each token.

► **Definition 2** (Canonical paths). *Let A and B denote the starting and target separators, respectively. By Theorem 1, we begin by fixing a maximum set of pairwise internally disjoint s - t -paths, which has size $k = |A| = |B|$ (since A, B are minimum separators). We may assume that all paths are chordless, i.e., if two vertices of the same path are adjacent the edge connecting them is part of the path; otherwise, we could decrease the length of the path by shortcutting along the chord. We repeat this procedure until all paths are chordless; it terminates since in each iteration the total number of vertices involved in the paths decreases. We call these k chordless pairwise internally disjoint s - t -paths P_1, \dots, P_k the canonical paths of our instance.*

Note that canonical paths may not be uniquely defined; however, we can fix any set of k internally vertex disjoint paths as canonical.

► **Lemma 3.** *Let S be a minimum s - t -separator. Then, S contains exactly one vertex of each canonical path.*

Proof. The set S has to contain at least one vertex of each path since otherwise there would be an s - t -path in $G - S$. Since the number of paths is equal to the size of a minimum separator and the paths are disjoint, it has to be exactly one in each (Theorem 1). ◀

The next observation follows immediately from Lemma 3.

► **Observation 4.** *For both token sliding and token jumping, each token is confined to its respective canonical path and in the case of sliding, a token can only slide to either one of its two neighbours along its canonical path.*

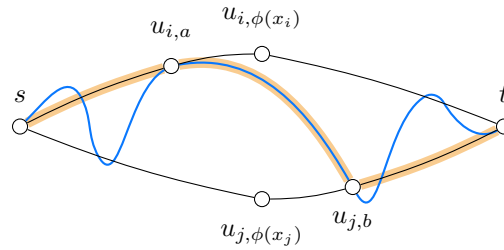
Thus, our view of the problem is that we are sliding (resp. jumping) tokens along a set of k paths and that each token is confined to its respective path. We now show that we can always slide (resp. jump) a token in the direction of the target separator B and never have to do a “backward” move.

Let $L(i)$ denote the number of vertices on the canonical path P_i , including s and t . Let $u_{i,1}, \dots, u_{i,L(i)}$ denote the vertices on the canonical path P_i in the order in which they appear on it, with $u_{i,1} = s$ and $u_{i,L(i)} = t$. Let a_i and b_i denote the indices such that $V(P_i) \cap A = \{u_{i,a_i}\}$ and $V(P_i) \cap B = \{u_{i,b_i}\}$, i.e., a_i is the index of the starting vertex of the token on P_i and b_i the index of the goal vertex for this token. Let $l_i = \min(a_i, b_i)$ and $r_i = \max(a_i, b_i)$. We first show that, in any (shortest) reconfiguration sequence, we only need to consider configurations of tokens in which, for all i , the token on the path P_i remains between (or on) u_{i,l_i} and u_{i,r_i} .

► **Lemma 5.** *For all $i \in [k]$, let ϕ_i be the function such that for all $1 < a < L(i)$,*

$$\phi_i(a) := \begin{cases} l_i & \text{if } a < l_i, \\ r_i & \text{if } a > r_i, \\ a & \text{otherwise.} \end{cases}$$

Let $f(u_{i,a}) := u_{i,\phi_i(a)}$. If X is a minimum s - t -separator, then so is $f(X)$.



■ **Figure 1** P is in blue and P' is highlighted in orange.

Proof. Given a set $X = \{u_{i,x_i} \mid i \in [k]\}$, let $\text{Left}(X) = \bigcup_i \{u_{i,a} \mid a < x_i\}$.

Let $X = \{u_{i,x_i} \mid i \in [k]\}$ be an s - t -separator. Assume that $Y = f(X)$ is not an s - t -separator. Let P be an s - t -path in $G - Y$. Let $u_{i,a}$ be the last vertex of P belonging to $\text{Left}(Y)$ (possibly $u_{i,a} = s$). Let $u_{j,b}$ be the first vertex of P lying on a canonical path after $u_{i,a}$ (possibly $u_{j,b} = t$). We have $u_{j,b} \notin \text{Left}(Y)$ and thus $b > \phi_j(x_j)$. Note that $a \neq \phi_i(x_i)$ and $b \neq \phi_j(x_j)$ by the definition of P . Let P' be the path in G going from s to $u_{i,a}$ via P_i , then to $u_{j,b}$ via P and finally to t via P_j ; see Figure 1. We have $i \neq j$ since otherwise P' would be an s - t -path in $G - A$ (respectively $G - B$ or $G - X$) if $\phi_i(x_i) = a_i$ (respectively $\phi_i(x_i) = b_i$ or $\phi_i(x_i) = x_i$).

We cannot have $a < x_i$ and $x_j < b$ at the same time since otherwise P' would be an s - t -path in $G - X$. Without loss of generality, assume that $x_i \leq a$. As a result, $x_i \leq a < \phi_i(x_i) = l_i$. We must have $b < l_j$, or otherwise P' would be an s - t -path in $G - A$ or $G - B$. Thus, $b < l_j \leq \phi_j(x_j)$ which contradicts the aforementioned property that $b > \phi_j(x_j)$ and proves that $f(X)$ is an s - t -separator. ◀

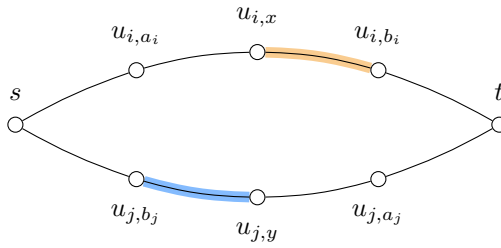
► **Corollary 6.** *In both the token jumping and token sliding models, if there exists a reconfiguration sequence from A to B , then there exists a shortest sequence such that, for any i , the i^{th} token remains between l_i and r_i at all times. As a result, deleting all vertices on canonical paths that are not between l_i and r_i and replacing them by the edges $\{\{s, l_i\} \mid 1 \leq i \leq k\} \cup \{\{r_i, t\} \mid 1 \leq i \leq k\}$ yields an equivalent instance.*

Proof. Two s - t -separators differing only by a token jump (resp. slide) are mapped by f to two s - t -separators that are either equal or differing by only a token jump (resp. slide). Thus, applying f to all separators in the reconfiguration sequence, we get a reconfiguration sequence with the claimed property. ◀

Given a vertex $u_{i,x}$ on a canonical path P_i , we define the set $F(u_{i,x})$ as the set of vertices of P_i between $u_{i,x}$ and u_{i,b_i} (see Figure 2). We say that a jump (resp. slide) from $u_{i,a}$ to $u_{i,b}$ is *forward* if $u_{i,b} \in F(u_{i,a})$. Intuitively, this means that a jump (resp. slide) is forward if it moves the token closer to its target location (along the canonical path) without going past it.

► **Lemma 7 (Forward-moving lemma).** *If there exists a reconfiguration sequence from A to B , then there exists a shortest sequence \mathcal{S} of jumps (resp. slides) going from A to B containing only forward jumps (resp. slides).*

Proof. We proceed by induction on the length of a shortest sequence. If $A = B$, there is nothing to prove. Otherwise, let \mathcal{S} be a shortest sequence of moves going from A to B . By Corollary 6, we can assume that the first move in \mathcal{S} is a forward move. Let A' be the separator obtained after this move. The tail of \mathcal{S} is a shortest sequence of move between A' and B and by induction, it may be replaced with a shortest sequence that only contains forward jumps (resp. slides). ◀



■ **Figure 2** $F(u_{i,x})$ is highlighted in orange and $F(u_{j,y})$ in blue.

4 Polynomial-time algorithms

The forward-moving lemma immediately implies that several problems can be solved in polynomial time by a greedy algorithm.

► **Theorem 8.** *A minimum-length sequence of token slides reconfiguring one minimum s - t -separator to another can be computed in polynomial time.*

Proof. Since slides are reversible, doing any slide can never turn a yes-instance into a no-instance. Thus, we can greedily apply moves that slide a token forward. That is, we iteratively find any token that can slide forward (as long as possible) and execute the slide. Since we never need to do a backward slide (Lemma 7), this always finds a solution if one exists. Moreover, this is optimal; the paths are chordless, so any slide can only advance a token one step closer towards its target position, and since there are no backward slides, the solution is optimal. Clearly, checking if a token can slide forward requires polynomial time and given that an optimal solution has polynomial length we get the claimed running time. ◀

► **Theorem 9.** *A (feasible, but not necessarily minimum-length) sequence of token jumps reconfiguring one minimum s - t -separator to another can be computed in polynomial time.*

Proof. Jumps are also reversible, so doing a jump can never turn a yes-instance into a no-instance. Again, we can greedily apply forward jumps. Since a solution never needs to contain a backward jump, this yields a feasible solution if one exists. ◀

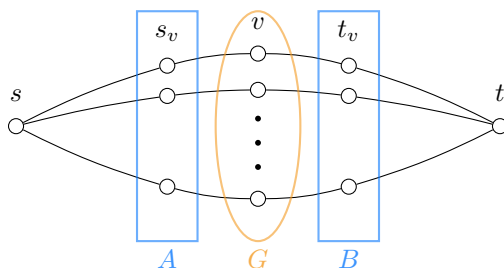
In the case of token jumping, the solution produced by the greedy algorithm is not necessarily optimal (not guaranteed to be a shortest sequence of jumps); by choosing a different order for the jumps, it might be possible to make “longer” jumps, i.e., jumping over more vertices. In fact, we show that deciding whether a sequence of at most ℓ jumps can transform one minimum s - t -separator into another is an NP-complete problem.

5 Hardness of finding short sequences of jumps

First, we note that the problem of deciding whether a sequence of at most ℓ jumps between two minimum s - t -separators exists is in NP. Indeed, Lemma 7 implies that the length of a reconfiguration sequence cannot exceed $|V(G)|$; we can therefore directly use a reconfiguration sequence as a certificate.

We show NP-hardness by reducing the VERTEX COVER problem. Given a graph $G = (V, E)$ and the size κ of a desired vertex cover, we construct our instance (G', s, t, A, B) as follows. We first create a copy of the graph G , and for every $v \in V(G)$ we add two additional

vertices s_v, t_v and edges $\{s_v, v\}, \{v, t_v\}$. We further add vertices s, t and for all $v \in V(G)$, edges $\{s, s_v\}, \{t_v, t\}$; see Figure 3. We ask whether we can reconfigure the s - t -separator $A = \{s_v \mid v \in V(G)\}$ to the separator $B = \{t_v \mid v \in V(G)\}$ using at most $|V(G)| + \kappa$ token jumps. Note that the canonical paths are of the form $\{s, s_v, v, t_v, t \mid v \in V(G)\}$.



■ **Figure 3** The graph G' formed from G (highlighted in orange), along with the initial and target separators A and B (highlighted in blue), respectively.

► **Theorem 10.** *Deciding whether a sequence of at most ℓ token jumps can transform one minimum s - t -separator into another is an NP-complete problem.*

6 Preprocessing for token jumping

In this section, we describe some preprocessing rules that can be applied to the token jumping variant of MINIMUM SEPARATOR RECONFIGURATION. In the remainder of this paper, we assume that all instances are preprocessed according to these rules. We first show that we can reduce the graph so that it contains no vertices that are not on the canonical s - t -paths.

► **Lemma 11.** *Given an instance (G, s, t, A, B) of MINIMUM SEPARATOR RECONFIGURATION in the token jumping model, it is possible to compute in polynomial time an equivalent instance (G', s, t, A, B) in which all vertices are on the (chordless) canonical paths and both the minimum s - t -separator size and the length of minimum reconfiguration sequences are preserved. Moreover, all vertices in $A \cup B$ are adjacent to either s or t .*

Going forward, we assume that all graphs are preprocessed according to these rules. We next observe that to ensure that a configuration of tokens forms a valid s - t -separator it suffices to check the existence of relatively simple s - t -paths.

► **Lemma 12.** *To check whether a configuration of tokens that assigns exactly one token to each canonical path forms an s - t -separator, it suffices to check whether there exists an s - t -path that from s , follows one of the canonical paths, then follows one edge (a crossing edge) from that canonical path to another, and then follows that canonical path to t .*

We now show that it suffices to consider instances in which all vertices have degree at least 3 and at most $2k$, where $k \geq 3$ is the size of the minimum separator (for $k \leq 2$, the problem can be solved in polynomial time by a simple algorithm that computes a shortest path in an auxiliary graph H having one vertex for each of the at most n^2 s - t -separators and where two vertices of H share an edge whenever the corresponding s - t -separators are one reconfiguration step away from each other).

► **Lemma 13.** *Given an instance (G, s, t, A, B) of MINIMUM SEPARATOR RECONFIGURATION in the token jumping model, it is possible to compute in polynomial time an equivalent instance (G', s, t, A, B) in which the length of minimum reconfiguration sequences is preserved and all vertices have degree at least 3 and at most $2k$, where $k = |A| = |B|$ (assuming $k \geq 3$). In particular, every vertex can have at most two neighbors on each canonical path.*

We proceed by showing that we can always assume that the source and target minimum s - t -separators, i.e., A and B , are disjoint.

► **Lemma 14.** *Given an instance (G, s, t, A, B) of MINIMUM SEPARATOR RECONFIGURATION in the token jumping model, it is possible to compute in polynomial time an equivalent instance (G', s, t, A', B') in which $A' \cap B' = \emptyset$ and such that the length of minimum reconfiguration sequences is preserved.*

We conclude this section by formalizing the notion of unskippable vertices; a notion that will be useful in many of our subsequent results. Given an instance (G, s, t, A, B) , we say that $v \in V(G) \setminus (A \cup B \cup \{s, t\})$ is *unskippable* if for every sequence (if any exist) of minimum s - t -separators that transforms A to B there exists at least one s - t -separator S in the sequence such that $v \in S$. In other words, there is no transformation from A to B that can skip over v and not jump a token onto v at some point. Similarly, we say that a set $U \subseteq V(G) \setminus (A \cup B \cup \{s, t\})$ of vertices is *unskippable* whenever there exists at least one s - t -separator S in every reconfiguration sequence (from A to B) such that $|U \cap S| \geq 1$.

► **Lemma 15.** *Let (G, s, t, A, B) be an instance of MINIMUM SEPARATOR RECONFIGURATION in the token jumping model. A vertex $v \in V(G) \setminus (A \cup B \cup \{s, t\})$ having two neighbors on a canonical path other than its own is unskippable. If $u, v \in V(G) \setminus (A \cup B \cup \{s, t\})$, u, v belong to two different canonical paths, and $\{u, v\} \in E(G)$, then $\{u, v\}$ is unskippable.*

7 Parameterizing by the size of the separators

In this section, we study the natural parameterization by k : the number of tokens, or the size of minimum separators. We first prove that there exists an FPT algorithm for this parameterization, then show that no polynomial kernel exists unless $\text{NP} \subseteq \text{coNP/poly}$.

7.1 FPT algorithm

In this section, we show that finding a shortest sequence for the token jumping variant of MINIMUM SEPARATOR RECONFIGURATION is fixed-parameter tractable with respect to k , the size of the minimum s - t -separators. We do so by constructing a path decomposition of the graph from an arbitrary sequence of token jumps, and then proceed by designing a dynamic programming algorithm that operates on that path decomposition. We shall work towards proving the following.

► **Lemma 16.** *Given a graph G preprocessed by our reduction rules and two minimum s - t -separators A and B of G with $|A| = k$, if A and B can be reconfigured into each other, then $G \setminus \{s, t\}$ has pathwidth at most k .*

Let $\mathcal{S} = A, S_1, \dots, S_\ell, B$ be a reconfiguration sequence of minimum separators. Note that the symmetric difference between two elements of \mathcal{S} are two vertices $u_{j,p} \in S_i \setminus S_{i+1}$ and $u_{j,r} \in S_{i+1} \setminus S_i$ belonging to the *same* canonical path. As such, we can construct a width- k path decomposition where (i) each $S_i \in \mathcal{S}$ is placed in the order they appear in \mathcal{S} and (ii) between S_i and S_{i+1} we have the bags $X \cup \{u_{j,q}\}$, where $X = S_i \cap S_{i+1}$ and $p < q < r$. Using the previous statement and dynamic programming, we prove our main result.

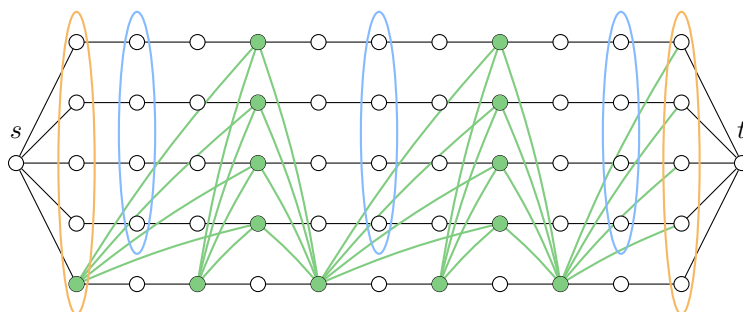
► **Theorem 17.** *The optimization version of MINIMUM SEPARATOR RECONFIGURATION under token jumping parameterized by the size of a minimum separator is in FPT.*

7.2 No polynomial kernel for parameter k

We use the cross-composition framework developed by the work of Drucker [11], Bodlaender et al. [3, 4], Dell and van Melkebeek [8], and Fortnow and Santhanam [14]. Roughly speaking, in this framework, a problem Π *and-cross-composes* into a parameterized problem Γ if, given instances $\{I_1, \dots, I_r\}$ of Π , we can construct an instance (O, k) of Γ such that: O is a yes-instance if and only if I_i is a yes-instance for all $i \in [r]$, $|O|$ is polynomial on $\sum_i |I_i| + r$, and $k \leq \text{poly}(\max_i |I_i| + \log r)$.

For MINIMUM SEPARATOR RECONFIGURATION under token jumping, we and-cross-compose VERTEX COVER, employing a construction very similar to the one used in Theorem 10. In fact, for each of the r vertex cover instances (G_i, κ) , we repeat the construction of Theorem 10 and serialize all the graphs in a “linear fashion” using synchronization gadgets, as depicted in Figure 4. We ask for a reconfiguration sequence between the leftmost and rightmost sets (each of size $k = \mu + 1$) of length $\mathcal{O}(rk + \sum(|V(G_i)|))$. As the name suggests, the purpose of the synchronization gadgets is to guarantee that all tokens jump over each graph G_i before moving on to the next graph (allowing us to accurately calculate the total number of required jumps). Formally, our result can be stated as follows:

► **Theorem 18.** *There exists an and-cross-composition from VERTEX COVER into TJ MINIMUM SEPARATOR RECONFIGURATION, parameterized by the minimum size k of an s - t -separator. Consequently, when parameterized by k , TJ MINIMUM SEPARATOR RECONFIGURATION does not admit a polynomial kernel unless $NP \subseteq \text{coNP}/\text{poly}$.*



■ **Figure 4** An overview of the and-cross-composition with $r = 3$ and $k = 5$. The orange blobs are A and B , the blue blobs represent the graphs G_i , and the green vertices and edges represent the synchronization gadgets.

8 Polynomial kernel for parameter ℓ

Observe that the construction employed in Theorem 18 heavily relies on the fact that the length ℓ of the reconfiguration sequence is linearly proportional to the number of instances. We show that this dependence cannot be broken. That is, when parameterizing by ℓ , we prove that MINIMUM SEPARATOR RECONFIGURATION admits a quadratic kernel.

Recall that, by Lemma 11, we can preprocess the graph so that all vertices are on the canonical paths and the vertices in A and B are adjacent to (at least one of) s or t . Similarly, by Lemma 13, each vertex in G will have degree at least 3 and at most $2k$ (with at most two neighbors on each canonical path) and, by Lemma 14, we know that $A \cap B = \emptyset$. We refer to an instance satisfying all of the above as a reduced instance.

► **Lemma 19.** *In a reduced yes-instance where ℓ is the parameter, all the following properties must be satisfied: (i) $A \cap B = \emptyset$ and $|A| = |B| \leq \ell$; (ii) all vertices are on the (at most ℓ) canonical paths; (iii) vertices in A and B are adjacent to (at least one of) s or t ; and (iv) each vertex in G has degree at least 3, degree at most 2ℓ , and at most two neighbors on each canonical path.*

Proof. The lemma follows immediately from the fact that if after obtaining a reduced instance we have $|A| = |B| > \ell$, then we have a no-instance; as at least $\ell + 1$ jumps are needed. Consequently, the minimum separator size will be at most ℓ and all the remaining properties follow from Lemma 11, Lemma 13, and Lemma 14. ◀

► **Lemma 20.** *Assume that in a reduced instance one of the canonical paths contains more than $4(\ell + 1)^2 + 4$ vertices. Then, the instance is a no-instance.*

Proof. Let P denote such a canonical path. We claim that at least $\ell + 1$ jumps are required for the token on P . We assume otherwise, i.e., that ℓ jumps or fewer are enough, and work towards a contradiction.

First, recall that if a vertex v on a canonical path is adjacent to two distinct vertices on another canonical path, then v can never be jumped over, i.e., v is unskippable (Lemma 15). We decompose P into $\ell + 1$ subpaths each consisting of at least $4\ell + 4$ vertices (excluding s , t , and the initial and target vertices of A and B). For the token on P to reach its final position in at most ℓ jumps, it must (at least once) jump over $2\ell + 1$ (consecutive) vertices of P or more (landing on the vertex $2\ell + 2$ away or more). Let us denote those vertices that are jumped over by Q . Moreover, let S_i denote the s - t -separator preceding the jump and let S_{i+1} denote the resulting s - t -separator after the jump.

If Q contains a vertex having two distinct neighbors on another canonical path, then the vertex is unskippable and we get a contradiction. Hence, every vertex v of Q (which has degree 3 or more) can have at most one neighbor on every canonical path $P' \neq P$ (and we know that v must have at least one neighbor not in P). Since S_i and S_{i+1} are s - t -separators, every vertex $v \in Q$ can only be adjacent to vertices in $S_i \setminus V(P) = S_{i+1} \setminus V(P)$; otherwise an s - t -path can be easily constructed, contradicting the fact that S_i and S_{i+1} are s - t -separators. Now, given that $|Q| \geq 2\ell + 1$, we know that there exists at least 3 distinct vertices in Q all having the same neighbor in $S_{i+1} \setminus V(P)$. This contradicts the fact that after our reductions each vertex in G can have at most two distinct neighbors on any canonical path. ◀

Lemmas 19 and 20 immediately imply a kernel with $\mathcal{O}(\ell^3)$ vertices: a yes-instance consists of at most ℓ canonical paths each having $\mathcal{O}(\ell^2)$ vertices. We obtain an improved bound by refining our analysis slightly and strengthening the result of Lemma 20 in Theorem 21.

► **Theorem 21.** *The optimization version of TJ MINIMUM SEPARATOR RECONFIGURATION admits a kernel with $\mathcal{O}(\ell^2)$ vertices and edges when parameterized by the length ℓ of a reconfiguration sequence.*

9 Concluding Remarks

We studied the minimum s - t -separator reconfiguration problem through several lenses. First, we considered the token sliding and token jumping reconfiguration models, showing that the reachability question is answerable in polynomial time in both cases. Afterwards, we considered the task of finding a shortest reconfiguration sequence; we proved that it is easy under the first model but NP-complete under the second. To tackle this hardness, we studied the parameterized complexity of the token jumping version for the natural parameterizations

k (the number of tokens) and ℓ (the length of the sequence). In this context, we designed an FPT algorithm for parameter k , a quadratic kernel when parameterized by ℓ , and showed that no polynomial kernel exists for k unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

In terms of future work on minimum s - t -separator reconfiguration itself, we are interested in understanding shortest token jumping sequences for some graph classes, in particular for planar graphs. Other possibilities include the study of structural parameterizations, such as treewidth, feedback edge set, and vertex deletion distance metrics (e.g., distance to cluster, clique, etc.). In a different spirit, studying the connectivity of the minimum s - t -separator reconfiguration graph might yield different insights than the ones presented in this paper. Beyond minimum separators, the work of Gomes, Nogueira, and dos Santos [16] investigated arbitrary s - t -separator reconfiguration, but no research has yet been done on bounded size separators. As such, we believe work on the complexity of reconfiguration of minimum $+r$ s - t -separators might be of independent interest.

References

- 1 Kira Adaricheva, Chassidy Bozeman, Nancy E. Clarke, Ruth Haas, Margaret-Ellen Messinger, Karen Seyffarth, and Heather C. Smith. Reconfiguration graphs for dominating sets. In *Research Trends in Graph Theory and Applications*, volume 25 of *Assoc. Women Math. Ser.*, pages 119–135. Springer, Cham, 2021.
- 2 Akanksha Agrawal, Soumita Hait, and Amer E. Mouawad. On finding short reconfiguration sequences between independent sets. In *33rd International Symposium on Algorithms and Computation*, volume 248 of *LIPICs. Leibniz Int. Proc. Inform.*, pages Paper No. 39, 14. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2022.
- 3 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.
- 4 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discret. Math.*, 28(1):277–305, 2014. doi:10.1137/120880240.
- 5 Robert Connelly, Erik D. Demaine, and Günter Rote. Blowing up polygonal linkages. *Discrete Comput. Geom.*, 30:205–239, 2003.
- 6 Daniel W. Cranston and Reem Mahmoud. In most 6-regular toroidal graphs all 5-colorings are Kempe equivalent. *European J. Combin.*, 104:Paper No. 103532, 21, 2022.
- 7 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 8 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.
- 9 Quentin Deschamps, Carl Feghali, František Kardoš, Clément Legrand-Duchesne, and Théo Pierron. Strengthening a theorem of Meyniel, 2022.
- 10 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.
- 11 Andrew Drucker. New limits to classical and quantum instance compression. *SIAM J. Comput.*, 44(5):1443–1479, 2015. doi:10.1137/130927115.
- 12 Cem Evrendilek. Vertex separators for partitioning a graph. *Sensors*, 8(2):635–657, 2008.
- 13 Lester R. Ford and Delbert R. Fulkerson. *Flows in Networks*. Princeton University Press, USA, 2010.
- 14 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011. doi:10.1016/j.jcss.2010.06.007.
- 15 Bin Fu and Zhixiang Chen. Sublinear time width-bounded separators and their application to the protein side-chain packing problem. *J. Comb. Optim.*, 15(4):387–407, 2008.

9:12 Minimum Separator Reconfiguration

- 16 Guilherme C. M. Gomes, Sérgio H. Nogueira, and Vinícius F. dos Santos. Some results on vertex separator reconfiguration, 2020.
- 17 Takehiro Ito, Erik D. Demaine, Xiao Zhou, and Takao Nishizeki. Approximability of partitioning graphs with supply and demand. *J. Discrete Algorithms*, 6(4):627–650, 2008.
- 18 Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto. Shortest reconfiguration of perfect matchings via alternating cycles. *SIAM J. Discrete Math.*, 36(2):1102–1123, 2022.
- 19 Enver Kayaaslan, Ali Pinar, Ümit Çatalyürek, and Cevdet Aykanat. Partitioning hypergraphs in scientific computing applications through vertex separators on graphs. *SIAM J. Sci. Comput.*, 34(2):A970–A992, 2012.
- 20 Charles E. Leiserson. Area-efficient graph layouts. In *21st Annual Symposium on Foundations of Computer Science*, pages 270–281, 1980.
- 21 Karl Menger. Zur allgemeinen Kurventheorie. *Fundamenta Mathematicae*, 10:96–115, 1927. URL: <http://eudml.org/doc/211191>.
- 22 István Miklós and Heather Smith. Sampling and counting genome rearrangement scenarios. *BMC Bioinformatics*, 16, 2015.
- 23 Xie Xian-fen, Gu Wan-rong, He Yi-chen, and Mao Yi-jun. Matrix transformation and factorization based on graph partitioning by vertex separator for recommendation. *Computer Science*, 49(6):272–279, 2022.

Kernels for the Disjoint Paths Problem on Subclasses of Chordal Graphs

Juhi Chaudhary   

Ben-Gurion University of the Negev, Beersheba, Israel

Harmender Gahlawat   

Ben-Gurion University of the Negev, Beersheba, Israel

Michał Włodarczyk   

University of Warsaw, Poland

Meirav Zehavi   

Ben-Gurion University of the Negev, Beersheba, Israel

Abstract

Given an undirected graph G and a multiset of k terminal pairs \mathcal{X} , the VERTEX-DISJOINT PATHS (VDP) and EDGE-DISJOINT PATHS (EDP) problems ask whether G has k pairwise internally vertex-disjoint paths and k pairwise edge-disjoint paths, respectively, connecting every terminal pair in \mathcal{X} . In this paper, we study the kernelization complexity of VDP and EDP on subclasses of chordal graphs. For VDP, we design a $4k$ vertex kernel on split graphs and an $\mathcal{O}(k^2)$ vertex kernel on well-partitioned chordal graphs. We also show that the problem becomes polynomial-time solvable on threshold graphs. For EDP, we first prove that the problem is NP-complete on complete graphs. Then, we design an $\mathcal{O}(k^{2.75})$ vertex kernel for EDP on split graphs, and improve it to a $7k + 1$ vertex kernel on threshold graphs. Lastly, we provide an $\mathcal{O}(k^2)$ vertex kernel for EDP on block graphs and a $2k + 1$ vertex kernel for clique paths. Our contributions improve upon several results in the literature, as well as resolve an open question by Heggenes et al. [Theory Comput. Syst., 2015].

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Mathematics of computing \rightarrow Graph algorithms

Keywords and phrases Kernelization, Parameterized Complexity, Vertex-Disjoint Paths Problem, Edge-Disjoint Paths Problem

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.10

Related Version *Full Version:* <https://doi.org/10.48550/arXiv.2309.16892> [5]

Funding This research was supported by the European Research Council (ERC) grant titled PARAPATH.

1 Introduction

The VERTEX-DISJOINT PATHS (VDP) and EDGE-DISJOINT PATHS (EDP) problems are fundamental routing problems, having applications in VLSI design and virtual circuit routing [20, 39, 44, 45]. Notably, they have been a cornerstone of the groundbreaking Graph Minors project of Robertson and Seymour [42], and several important techniques, including the *irrelevant vertex technique*, originated in the process of solving disjoint paths [42]. In VDP (respectively, EDP), the input is an undirected graph G and a multiset of terminal pairs $\mathcal{X} = \{(s_1, t_1), \dots, (s_k, t_k)\}$, and the goal is to find k pairwise internally vertex-disjoint (respectively, edge-disjoint) paths P_1, \dots, P_k such that P_i is a path with endpoints s_i and t_i .

Both VDP and EDP are studied extensively, and have been at the center of numerous results in algorithmic graph theory [6, 15, 18, 22, 31, 34, 36, 48]. Karp [30] proved that VDP is NP-complete (attributing the result to Knuth), and a year later, Even, Itai, and



© Juhi Chaudhary, Harmender Gahlawat, Michał Włodarczyk, and Meirav Zehavi; licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 10; pp. 10:1–10:22

Leibniz International Proceedings in Informatics



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

■ **Table 1** Summary of the kernelization results of VDP and EDP parameterized by the number of occurrences of terminal pairs (k) on the subclasses of chordal graphs studied in this paper.

<u>Graph Class</u>	<u>VDP</u>	<u>EDP</u>
Well-partitioned Chordal	$\mathcal{O}(k^2)$ vertex kernel [Theorem 7]	OPEN
Split	$4k$ vertex kernel [Theorem 6]	$\mathcal{O}(k^{2.75})$ vertex kernel [Theorem 2]
Threshold	P [Theorem 9]	$7k + 1$ vertex kernel [Theorem 3]
Block	P [Observation 8]	$4k^2 - 2k$ vertex kernel [Theorem 4]
Clique Path	P [Observation 8]	$2k + 1$ vertex kernel [Theorem 5]

Shamir [16] proved the same for EDP. When k is *fixed* (i.e., treated as a constant), Robertson and Seymour [37, 42] gave an $\mathcal{O}(|V(G)|^3)$ time algorithm as a part of their famous Graph Minors project. This algorithm is a *fixed-parameter tractable* (FPT) algorithm parameterized by k . Later, the power 3 was reduced to 2 by Kawarabayashi, Kobayashi, and Reed [32].

In Parameterized Complexity, each problem instance is associated with an integer parameter k . We study both VDP and EDP through the lens of *kernelization* under the parameterization by k . A *kernelization algorithm* is a polynomial-time algorithm that takes as input an instance (I, k) of a problem and outputs an *equivalent instance* (I', k') of the same problem such that the size of (I', k') is bounded by some computable function $f(k)$. The problem is said to admit an $f(k)$ sized kernel, and if $f(k)$ is polynomial, then the problem is said to admit a polynomial kernel. It is known that a problem is FPT if and only if it admits a kernel. Due to its profound impact, kernelization has been termed “*the lost continent of polynomial time*” [17]. For more details on kernelization, we refer to books [12, 19].

Bodlaender et al. [4] proved that, unless $\text{NP} \subseteq \text{coNP/poly}$, VDP does not admit a polynomial kernel (on general graphs). On the positive side, Heggernes et al. [27] extended this study to show that VDP and EDP admit polynomial kernels on split graphs with $\mathcal{O}(k^2)$ and $\mathcal{O}(k^3)$ vertices, respectively. Yang et al. [47] further showed that a restricted version of VDP, where each vertex can appear in at most one terminal pair, admits a $4k$ vertex kernel. Recently, Ahn et al. [2] introduced so-called *well-partitioned chordal graphs* (being a generalization of split graphs), and showed that VDP on these graphs admits an $\mathcal{O}(k^3)$ vertex kernel. In this paper, we extend the study of kernelization of EDP and VDP on these (and other) subclasses of chordal graphs. We provide an extended survey in the Appendix.

1.1 Our Contribution

Proofs of the results marked with (*) are deferred to the full version [5] to respect the space constraints. An overview of our results is given in Table 1. We begin by discussing the results about EDP. First, we observe that the problem remains NP-hard even on inputs with a trivial graph structure given by a clique, unlike VDP. This extends the known hardness results for split graphs [27] and graphs of cliquewidth at most 6 [25]. We prove this theorem in the Appendix.

► **Theorem 1.** *EDP is NP-hard on complete graphs.*

Every graph class treated in this paper includes cliques, so EDP is NP-hard on each of them. This motivates the study of kernelization algorithms. From now on, we always use k to denote the number of occurrences of terminal pairs in an instance.

We present an $\mathcal{O}(k^{2.75})$ vertex kernel for EDP on split graphs, improving upon the $\mathcal{O}(k^3)$ vertex kernel given by Heggernes et al. [27]. Our main technical contribution is a lemma stating that the length of each path in a minimum-size solution is bounded by $\mathcal{O}(\sqrt{k})$. This allows us to obtain the following.

► **Theorem 2.** *EDP on split graphs admits a kernel with at most $\mathcal{O}(k^{2.75})$ vertices.*

In the quest to achieve better bounds, we consider a subclass of split graphs. Specifically, we prove that EDP on threshold graphs admits a kernel with at most $7k + 1$ vertices. Here, we exploit the known vertex ordering of threshold graphs that exhibits an inclusion relation concerning the neighborhoods of the vertices.

► **Theorem 3 (*)**. *EDP on threshold graphs admits a kernel with at most $7k + 1$ vertices.*

Another important subclass of chordal graphs is the class of block graphs. For this case, we present a kernel with at most $4k^2 - 2k$ vertices. Our kernelization algorithm constructs an equivalent instance where the number of blocks can be at most $4k - 2$, and each block contains at most k vertices. Thus, we have the following theorem.

► **Theorem 4.** *EDP on block graphs admits a kernel with at most $4k^2 - 2k$ vertices.*

Whenever a block has more than two cut vertices, decreasing the size of that block below $\mathcal{O}(k)$ becomes trickier. However, if we restrict our block graph to have at most two cut vertices per block – i.e., if we deal with *clique paths* – then this can be done. The key point in designing our linear kernel in clique paths is that, in the reduced instance, for each block B , the number of vertices in B is dictated by a linear function of the number of terminal pairs having at least one terminal vertex in B . So, we obtain a $2k + 1$ vertex kernel for this class.

► **Theorem 5.** *EDP on clique paths admits a kernel with at most $2k + 1$ vertices.*

Now, we switch our attention to kernelization algorithms for VDP. First, we give a $4k$ vertex kernel for VDP on split graphs. This resolves an open question by Heggenes et al. [27], who asked whether this problem admits a linear vertex kernel. For this purpose, we use the result by Yang et al. [47], who gave a $4k$ vertex kernel for a restricted variant of VDP, called VDP-UNIQUE by us, where each vertex can participate in at most one terminal pair.

In order to obtain a linear vertex kernel for VDP, we give a parameter-preserving reduction to VDP-UNIQUE. Our reduction relies on a non-trivial matching-based argument.

In this way, we improve upon the $4k^2$ vertex kernel given by Heggenes et al. [27] as well as generalize the result given by Yang et al. [47]. Specifically, we have the following theorem.

► **Theorem 6.** *VDP on split graphs admits a kernel with at most $4k$ vertices.*

Next, we give an $\mathcal{O}(k^2)$ vertex kernel for VDP on well-partitioned chordal graphs (see Definition 64). Ahn et al. [2] showed that VDP admits an $\mathcal{O}(k^3)$ vertex kernel on this class. We improve their bound by giving a marking procedure that marks a set of at most $\mathcal{O}(k^2)$ vertices in G , which “covers” some solution (if it exists). As a result, we arrive at the following theorem.

► **Theorem 7 (*)**. *VDP on well-partitioned chordal graphs admits a kernel with $\mathcal{O}(k^2)$ vertices.*

Unlike EDP, the VDP problem turns out easier on the remaining graph classes. In block graphs, for every terminal pair with terminals in different blocks, there is a unique induced path connecting these terminals (all internal vertices of this path are the cut vertices). After adding these paths to the solution, we end up with a union of clique instances, where VDP is solvable in polynomial time. This leads to the following observation about block graphs and its subclass, clique paths.

► **Observation 8.** *VDP on block graphs (and, in particular, on clique paths) is solvable in polynomial time.*

Finally, we identify a less restricted graph class on which VDP is polynomial-time solvable, namely the class of threshold graphs. This yields a sharp separation between split graphs and its subclass – threshold graphs – in terms of VDP.

► **Theorem 9 (*)**. *VDP on thresholds graphs is solvable in polynomial time.*

1.2 Organization of the paper

We begin with formal preliminaries, where we gather information about the studied graph classes and the basic algorithmic tools. In Section 3, we prove the kernelization theorems for EDP, which are followed by the NP-hardness proof for EDP on cliques in Appendix C. Next, we cover the kernelization results for VDP in Section 4. We conclude in Section 5.

2 Preliminaries

For a positive integer ℓ , let $[\ell]$ denote the set $\{1, \dots, \ell\}$. We provide the preliminaries concerning parameterized complexity and graph classes considered in the Appendix.

Graph Notations. All graphs considered in this paper are simple, undirected, and connected unless stated otherwise. Standard graph-theoretic terms not explicitly defined here can be found in [13]. For a graph G , let $V(G)$ denote its vertex set, and $E(G)$ denote its edge set. For a graph G , the subgraph of G induced by $S \subseteq V(G)$ is denoted by $G[S]$, where $G[S] = (S, E_S)$ and $E_S = \{xy \in E(G) \mid x, y \in S\}$. For two sets $X, Y \subseteq V(G)$, we denote by $G[X, Y]$ the subgraph of G with vertex set $X \cup Y$ and edge set $\{xy \in E(G) : x \in X, y \in Y\}$. The *open neighborhood* of a vertex v in G is $N_G(v) = \{u \in V(G) : uv \in E(G)\}$. The *degree* of a vertex v is $|N_G(v)|$, and it is denoted by $d_G(v)$. When there is no ambiguity, we do not use the subscript G in $N_G(v)$ and $d_G(v)$. A vertex v with $d(v) = 1$ is a *pendant vertex*. The *distance* between two vertices in a graph G is the number of edges in the shortest path between them. We use the notation $d(u, v)$ to represent the distance between two vertices u and v in a graph G (when G is clear from the context). For a graph G and a set $X \subseteq V(G)$, we use $G - X$ to denote $G[V(G) \setminus X]$, that is, the graph obtained from G by deleting X . In a graph G , two vertices u and v are *twins* if $N_G[u] = N_G[v]$.

An *independent set* of a graph G is a subset of $V(G)$ such that no two vertices in the subset have an edge between them in G . A *clique* is a subset of $V(G)$ such that every two distinct vertices in the subset are adjacent in G . Given a graph G , a *matching* M is a subset of edges of G that do not share an endpoint. The edges in M are called *matched edges*, and the remaining edges are called *unmatched edges*. Given a matching M , a vertex $v \in V(G)$ is *saturated* by M if v is incident on an edge of M , that is, v is an end vertex of some edge of M . Given a graph G , MAX MATCHING is to find a matching of maximum cardinality in G .

► **Proposition 10** ([28]). *For a bipartite graph G , MAX MATCHING can be solved in $\mathcal{O}(\sqrt{|V(G)|} \cdot |E(G)|)$ time.*

A path $P = (v_1, \dots, v_n)$ is an *M -alternating path* if the edges in P are matched and unmatched alternatively with respect to M . If both the end vertices of an alternating path are unsaturated, then it is an *M -augmenting path*.

► **Proposition 11** ([3]). *A matching M is maximum if and only if there is no M -augmenting path in G .*

A path $P = (v_1, v_2, \dots, v_n)$ on n vertices is a (v_1, v_n) -path and $\{v_2, \dots, v_{n-1}\}$ are the *internal vertices* of P . Moreover, for a path $P = (v_1, v_2, \dots, v_n)$, we say that P *visits* the vertices $\{v_1, v_2, \dots, v_n\}$. Throughout this paper, let P_{uv} denote the path containing only the edge uv . Let P_1 be an (s_1, t_1) -path and P_2 be an (s_2, t_2) -path. Then, P_1 and P_2 are *vertex-disjoint* if $V(P_1) \cap V(P_2) = \emptyset$. Moreover, P_1 and P_2 are *internally vertex-disjoint* if $(V(P_1) \setminus \{s_1, t_1\}) \cap V(P_2) = \emptyset$ and $(V(P_2) \setminus \{s_2, t_2\}) \cap V(P_1) = \emptyset$, that is, no internal vertex of one path is used as a vertex on the other path, and vice versa. Two paths are said to be *edge-disjoint* if they do not have any edge in common. Note that two internally vertex-disjoint paths are edge-disjoint, but the converse may not be true. A path P is *induced* if $G[V(P)]$ is the same as P . For a path $P = (v_1, \dots, v_n)$ on n vertices and vertices $v_i, v_j \in V(P)$, let (v_i, v_j) -subpath of P denote the subpath of P with endpoints v_i and v_j .

Problem Statements. Given a graph G and a set (or, more generally, an ordered multiset) \mathcal{X} of pairs of distinct vertices in G , we refer to the pairs in \mathcal{X} as *terminal pairs*. A vertex in G is a *terminal vertex* if it appears in at least one terminal pair in \mathcal{X} (when \mathcal{X} is clear from context); else, it is a *non-terminal vertex*. For example, if G is a graph with $V(G) = \{v_1, v_2, \dots, v_6\}$, and $\mathcal{X} = \{(v_1, v_3), (v_2, v_3), (v_3, v_6), (v_1, v_6)\}$ is a set of terminal pairs, then $\{v_1, v_2, v_3, v_6\}$ are terminal vertices in G and $\{v_4, v_5\}$ are non-terminal vertices in G . Formally, the definitions of VDP and EDP are given below.

VERTEX-DISJOINT PATHS (VDP):
Input: A graph G and an ordered multiset $\mathcal{X} = \{(s_1, t_1), \dots, (s_k, t_k)\}$ of k terminal pairs.
Question: Does G contain k distinct and pairwise internally vertex-disjoint paths P_1, \dots, P_k such that for all $i \in [k]$, P_i is an (s_i, t_i) -path?

EDGE-DISJOINT PATHS (EDP):
Input: A graph G and an ordered multiset $\mathcal{X} = \{(s_1, t_1), \dots, (s_k, t_k)\}$ of k terminal pairs.
Question: Does G contain k pairwise edge-disjoint paths P_1, \dots, P_k such that for all $i \in [k]$, P_i is an (s_i, t_i) -path?

► **Remark 12.** Note that in both problems (VDP and EDP), we allow different terminal pairs to intersect, that is, it may happen that for $i \neq j$, $\{s_i, t_i\} \cap \{s_j, t_j\} \neq \emptyset$.

If there are two identical pairs $\{s_i, t_i\} = \{s_j, t_j\} = \{x, y\}$ in \mathcal{X} and the edge xy is present in G then only one of the paths P_i, P_j can use the edge xy if we require them to be edge-disjoint. However, setting $P_i = P_j = (x, y)$ does not violate the condition of being internally vertex-disjoint. It is natural though and also consistent with the existing literature to impose the additional condition that all paths in a solution have to be pairwise distinct.

► **Remark 13.** Throughout this paper, we assume that the degree of every terminal vertex is at least the number of terminal pairs in which it appears. Else, it is trivially a No-instance.

Following the notation introduced in [2] and [27], we have the following definitions.

► **Definition 14.** An edge $xy \in E(G)$ is *heavy* if for some $w \geq 2$, there exist pairwise distinct indices i_1, \dots, i_w such that for each $j \in [w]$, $\{x, y\} = \{s_{i_j}, t_{i_j}\}$. We call a terminal pair (s_i, t_i) *heavy* if $s_i t_i$ is a heavy edge; else, we call it *light*. Note that calling a terminal pair heavy or light only makes sense when the terminals in the pair have an edge between them.

► **Definition 15 (Minimum Solution).** Let (G, \mathcal{X}, k) be a Yes-instance of VDP or EDP. A solution $\mathcal{P} = \{P_1, \dots, P_k\}$ for the instance (G, \mathcal{X}, k) is *minimum* if there is no solution $\mathcal{Q} = \{Q_1, \dots, Q_k\}$ for (G, \mathcal{X}, k) such that $\sum_{i=1}^k |E(Q_i)| < \sum_{i=1}^k |E(P_i)|$.

Since we deal with subclasses of chordal graphs, the following proposition is crucial for us.

► **Proposition 16** ([27]). *Let (G, \mathcal{X}, k) be a Yes-instance of VDP such that G is a chordal graph, and let $\mathcal{P} = \{P_1, \dots, P_k\}$ be a minimum solution of (G, \mathcal{X}, k) . Then, for every path $P_i \in \mathcal{P}$, either P_i is an induced path or P_i is a path of length 2, and there exists a path $P_j \in \mathcal{P}$ of length 1 whose endpoints are the same as the endpoints of P_i .*

The next observation follows from Proposition 16.

► **Observation 17.** *Let (G, \mathcal{X}, k) be a Yes-instance of VDP. If there is a terminal pair $(s, t) \in \mathcal{X}$ such that $st \in E(G)$, then P_{st} belongs to every minimum solution of (G, \mathcal{X}, k) .*

3 **Kernelization Results on EDP**

We begin with the analysis of the simplest scenario where the input graph is a clique. In this setting, EDP is still NP-hard (see Appendix C), but we show below that whenever the size of the clique is larger than the parameter k , then we always obtain a Yes-instance. This improves the bound in [27, Lemma 7] by a factor of 2, which will play a role in optimizing the constants in our kernels (particularly, the linear ones).

► **Lemma 18** (*). *Let (G, \mathcal{X}, k) be an instance of EDP such that G is a clique. If $|V(G)| > k$, then (G, \mathcal{X}, k) is a Yes-instance.*

The bound above is tight as one can construct a No-instance (G, \mathcal{X}, k) where G is a clique and $|V(G)| = k$. Consider \mathcal{X} comprising just k copies of some pair $\{u, v\}$. Since the degree of u is $k - 1$, there cannot be k edge-disjoint paths having u as their common endpoint.

If G is a split graph with more than k vertices in the clique and the degree of each terminal vertex is at least the number of terminals on it, then we can reduce such an instance to the setting of Lemma 18 by replacing each terminal v in the independent set with an arbitrary neighbor of v . As a consequence, we obtain the following corollary, being a quantitative improvement over [27, Lemma 8].

► **Corollary 19.** *Let (G, \mathcal{X}, k) be an instance of EDP such that G is a split graph with split partition (C, I) . If $|C| > k$ and the degree of each terminal vertex is at least the number of terminals on it, then (G, \mathcal{X}, k) is a Yes-instance.*

3.1 **A Subcubic Vertex Kernel for Split Graphs**

In this section, we show that EDP on split graphs admits a kernel with $\mathcal{O}(k^{2.75})$ vertices. Let (G, \mathcal{X}, k) be an instance of EDP where G is a split graph. Note that given a split graph G , we can compute (in linear time) a partition (C, I) of $V(G)$ such that C is a clique and I is an independent set [26]. We partition the set I into two sets, say, I_T and I_N , where I_T and I_N denote the set of terminal vertices and the set of non-terminal vertices in I , respectively.

To ease the presentation of mathematical calculations, for this section (Section 3.1), we assume that $k^{\frac{1}{4}}$ is a natural number. If this is not the case, then we can easily get a new equivalent instance that satisfies this condition in the following manner. Let $d = (\lceil k^{\frac{1}{4}} \rceil)^4 - k$ and $v \in C$. Now, we add d terminal pairs $\{(s_{i_1}, t_{i_1}), \dots, (s_{i_d}, t_{i_d})\}$ and attach each of these terminals to v . Observe that this does not affect the size of our kernel ($\mathcal{O}(k^{2.75})$ vertices) since $(\lceil k^{\frac{1}{4}} \rceil)^4 = \mathcal{O}(k)$. Moreover, we assume that $k > 8$, as otherwise, we can use the FPT algorithm for EDP [42] to solve it in polynomial time.

Overview. Heggernes et al. [27] gave an $\mathcal{O}(k^3)$ vertex kernel for EDP on split graphs. In our kernelization algorithm (in this section), we use their algorithm as a preprocessing step. After the preprocessing step, the size of C and I_T gets bounded by $2k$ each, and the size of I_N gets bounded by $\mathcal{O}(k^3)$. Therefore, we know that the real challenge in designing an improved kernel for EDP on split graphs lies in giving a better upper bound on $|I_N|$.

Our kernelization algorithm makes a non-trivial use of a lemma (Lemma 25), which establishes that the length of each path in any minimum solution (of EDP on a split graph G) is bounded by $\mathcal{O}(\sqrt{k})$. This, in turn, implies that a minimum solution of EDP for split graphs contains $\mathcal{O}(k^{1.5})$ edges. Note that during the preprocessing step (i.e., the kernelization algorithm by Heggernes et al. [27]), for every pair of vertices in C , at most $4k + 1$ vertices are reserved in I_N , giving a cubic vertex kernel. In our algorithm, we characterized those vertices (called *rich* by us) in C for which we need to reserve only $\mathcal{O}(k^{1.5})$ vertices in I_N . Informally speaking, a vertex $v \in C$ is *rich* if there are $\Omega(k^{0.75})$ vertices in C that are “reachable” from v , even if we delete all the edges used by a “small” solution (containing $\mathcal{O}(k^{1.5})$ edges). Then, we show that if two vertices are rich, then even if they do not have any common neighbors in I_N , there exist “many” ($\Omega(k^{1.5})$) edge-disjoint paths between them even after removing any $\mathcal{O}(k^{1.5})$ edges of G . Hence, for every rich vertex, we keep only those vertices in I_N that are necessary to make the vertex rich, that is, we keep $\mathcal{O}(k^{1.5})$ vertices in I_N for every rich vertex. Thus, all rich vertices in C contribute a total of $\mathcal{O}(k^{2.5})$ vertices in I_N . The vertices in C that are not rich are termed as *poor*. Finally, we establish that a poor vertex cannot have too many neighbors in I_N . More specifically, a poor vertex can have only $\mathcal{O}(k^{1.75})$ neighbors in I_N . So, even if we keep all their neighbors in I_N , we store a total of $\mathcal{O}(k^{2.75})$ vertices in I_N for the poor vertices. This leads us to the desired kernel.

3.1.1 A Bound on the Length of the Paths in a Minimum Solution

In this section, we prove that for a minimum solution \mathcal{P} of an instance (G, \mathcal{X}, k) of EDP where G is a split graph, each path $P \in \mathcal{P}$ has length at most $4\sqrt{k} + 3$. We prove this bound by establishing that if there is a path of length $4\sqrt{k} + 4$ in \mathcal{P} , then \mathcal{P} contains at least $k + 1$ paths, a contradiction. To this end, we need the concept of *intersecting edges* (see Definition 21) and *non-compatible edges* (see Definition 22). Now, consider the following remark.

► **Remark 20.** For ease of exposition, throughout this section (Section 3.1.1), we assume (without mentioning explicitly it every time) that (G, \mathcal{X}, k) is a Yes-instance of EDP, where G is a split graph. Moreover, \mathcal{P} denotes a (arbitrary but fixed) minimum solution of (G, \mathcal{X}, k) , and $P \in \mathcal{P}$ is a path such that P contains ℓ vertices, say, v_1, \dots, v_ℓ , from clique C . Moreover, without loss of generality, let v_1, \dots, v_ℓ be the order in which these vertices appear in the path P from some terminal to the other. Note that if a path, say, P' , in a split graph has length p (i.e. $|E(P')| = p$), then it contains at least $\lceil \frac{p}{2} \rceil$ vertices from C . Therefore, to bound the length of P by $\mathcal{O}(\sqrt{k})$, it suffices to bound the number of vertices of C in P by $\mathcal{O}(\sqrt{k})$.

Assuming the ordering v_1, \dots, v_ℓ of the vertices in $V(P) \cap C$ along the path P , we have the following definitions.

► **Definition 21 (Intersecting Edges).** Consider two edges $e_i = v_i v_{i'}$ and $e_j = v_j v_{j'}$ such that $i, i', j, j' \in [\ell]$, and without loss of generality, assume that $i < i', j < j'$, and $i \leq j$. Then, e_i and e_j are non-intersecting if $j \geq i'$; otherwise, they are intersecting.

► **Definition 22 (Non-compatible Edges).** Two edges $e_1, e_2 \in E(G)$ are non-compatible if there does not exist a path $P' \in \mathcal{P} \setminus \{P\}$ (given $P \in \mathcal{P}$) such that $\{e_1, e_2\} \subseteq E(P')$. Moreover, a set of edges $\mathcal{S} = \{e_1, \dots, e_p\} \subseteq E(G)$ is non-compatible if every distinct $e_i, e_j \in \mathcal{S}$ are non-compatible.

Next, we show that, since P is a path in a minimum solution \mathcal{P} , most of the edges with both endpoints in $\{v_1, \dots, v_\ell\}$ are used by paths in \mathcal{P} (otherwise, we get a contradiction to the fact that \mathcal{P} is a minimum solution). In particular, we have the following lemma.

► **Lemma 23** (*). *Each edge of the form $v_i v_j$, where $i, j \in [\ell]$ and $j \geq i + 2$, is used by some path in $\mathcal{P} \setminus \{P\}$.*

Now, we show that if two edges are intersecting edges, then they are non-compatible.

► **Lemma 24** (*). *Let $e_i = v_i v_{i'}$ and $e_j = v_j v_{j'}$ be two (distinct) intersecting edges. Then, e_i and e_j are non-compatible.*

Now, we present the main lemma of this section.

► **Lemma 25** (*). *Let (G, \mathcal{X}, k) be a Yes-instance of EDP where G is a split graph. Moreover, let \mathcal{P} be a minimum solution of (G, \mathcal{X}, k) . Then, for every path $P \in \mathcal{P}$, $|E(P)| < 4\sqrt{k} + 4$.*

We have the following corollary as a consequence of Lemma 25 (since $k \geq 9$).

► **Corollary 26**. *Let \mathcal{P} be a minimum solution of an instance (G, \mathcal{X}, k) of EDP where G is a split graph. Then, $\sum_{P \in \mathcal{P}} |E(P)| \leq 5k^{1.5}$.*

3.1.2 An $\mathcal{O}(k^{2.75})$ Vertex Kernel for Split Graphs

In this section, we use Corollary 26 stating that there can be at most $5k^{1.5}$ edges in any minimum solution to design a subcubic ($\mathcal{O}(k^{2.75})$) vertex kernel for EDP on split graphs. We start with the following preprocessing step, which we apply only once to our input instance.

Preprocessing Step. First, we use the kernelization for EDP on split graphs provided by Heggernes et al. [27] as a preprocessing step. In their kernel, if $|C| \geq 2k$, then they report a Yes-instance (due to [27, Lemma 8]), and hence, assume that $|C| < 2k$. Due to Corollary 19, if $|C| > k$, then we have a Yes-instance, and hence we assume that $|C| \leq k$. Moreover, in their kernel, for any two vertices $u, w \in C$, $|N(u) \cap N(w) \cap I_N| \leq 4k + 1$ (i.e., u and w have at most $4k + 1$ common neighbors in I_N). Furthermore, there are no pendant vertices in I_N .

Next, we define a MARKING PROCEDURE, where we label the vertices in C as *rich* or *poor*. Furthermore, we partition the vertices in I_N into two sets, denoted U (read *unmarked*) and M (read *marked*), in the following manner.

Marking Procedure. Let (G, \mathcal{X}, k) be an instance of EDP where G is a split graph.

1. $M \Leftarrow \emptyset$ and $U \Leftarrow I_N$. (Initially, all vertices in I_N is unmarked.) Moreover, fix an ordering $v_1, \dots, v_{|C|}$ of the vertices of C .
2. For $1 \leq i \leq |C|$:
 - 2.1. $A_{v_i} \Leftarrow \emptyset$, $M_{v_i} \Leftarrow \emptyset$ (read *marked for v_i*), and $U_T = U$ (read *unmarked temporary*).
 - 2.2. For $1 \leq j \leq |C|$ such that $i \neq j$ and $|A_{v_i}| < 100k^{0.75}$:
 - 2.2.1. If $|N(v_i) \cap N(v_j) \cap U_T| \geq k^{0.75}$, then $A_{v_i} \Leftarrow A_{v_i} \cup \{v_j\}$. Moreover, select some (arbitrary) subset $M_{v_i, v_j} \subseteq N(v_i) \cap N(v_j) \cap U_T$ such that $|M_{v_i, v_j}| = k^{0.75}$. Then, $M_{v_i} \Leftarrow M_{v_i} \cup M_{v_i, v_j}$ and $U_T \Leftarrow U_T \setminus M_{v_i, v_j}$.
 - 2.3. If $|A_{v_i}| = 100k^{0.75}$, then label v_i as *rich*. Moreover, $M \Leftarrow M \cup M_{v_i}$ and $U \Leftarrow U_T$.
 - 2.4. If $|A_{v_i}| < 100k^{0.75}$, then label v_i as *poor*.

This completes our MARKING PROCEDURE.

► **Remark 27.** Note that the definition of *rich* and *poor* depends on the order in which our MARKING PROCEDURE picks and marks the vertices (i.e., being rich or poor is not an intrinsic property of the vertex itself). A different execution of the above procedure can label different vertices as rich and poor. Moreover, note that if $M_{v,x}$ exists (i.e., v is rich and $x \in A_v$) and $M_{x,v}$ exists (i.e., x is rich and $v \in A_x$), then $M_{x,v} \cap M_{v,x} = \emptyset$.

We have the following observation regarding the vertices marked rich by an execution of MARKING PROCEDURE on an instance (G, \mathcal{X}, k) of EDP where G is a split graph.

► **Observation 28 (*)**. Consider an execution of MARKING PROCEDURE on an instance (G, \mathcal{X}, k) of EDP where G is a split graph. Then for a rich vertex v , $|M_v| = 100k^{1.5}$ (i.e., the number of vertices marked in I_N for v are $100k^{1.5}$).

► **Definition 29 (Reachable Vertices)**. Consider an execution of MARKING PROCEDURE on an instance (G, \mathcal{X}, k) of EDP where G is a split graph. Moreover, let \mathcal{P} be a solution of (G, \mathcal{X}, k) . Then, for a rich vertex $v \in C$, let $R_v \subseteq A_v$ (read reachable from v) denote the set of vertices such that for each vertex $x \in R_v$, there is a vertex $u \in M_{v,x}$ such that u is not used by any path in \mathcal{P} .

Notice that, in Definition 29, a path of the form (v, u, x) is edge-disjoint from every path in \mathcal{P} . Furthermore, we briefly remark that R_v is defined with respect to the execution of MARKING PROCEDURE and a solution \mathcal{P} of (G, \mathcal{X}, k) , which we will always fix before we use R_v . Let \mathcal{P} be a solution to an instance (G, \mathcal{X}, k) of EDP. Informally speaking, in the following lemma, we show that if \mathcal{P} uses at most $6k^{1.5}$ edges, then for a rich vertex v , $R_v = \Omega(k^{0.75})$ (i.e., there are “many reachable” vertices in A_v from v using paths that are edge-disjoint from every path in \mathcal{P}). In particular, we have the following lemma.

► **Lemma 30 (*)**. Consider an execution of MARKING PROCEDURE on an instance (G, \mathcal{X}, k) of EDP where G is a split graph. Moreover, let \mathcal{P} be a solution of (G, \mathcal{X}, k) (not necessarily minimum) such that the total number of edges used in \mathcal{P} is at most $6k^{1.5}$. Then, for any rich vertex $v \in C$, $|R_v| \geq 94k^{0.75}$.

Next, we provide the following reduction rule.

► **Reduction Rule 1 (RR1)**. Let (G, \mathcal{X}, k) be an instance of EDP where G is a split graph. Let U be the set of unmarked vertices we get after an execution of MARKING PROCEDURE on (G, \mathcal{X}, k) . Moreover, let $U' \subseteq U$ be the set of vertices in U that do not have a poor neighbor. If $U' \neq \emptyset$, then $G' \Leftarrow G - U'$ and $\mathcal{X}' \Leftarrow \mathcal{X}$.

The following two lemmas (Lemmas 31 and 32) are essential to prove the safeness of RR1.

► **Lemma 31 (*)**. Let (G, \mathcal{X}, k) be an instance of EDP where G is a split graph. Consider an execution of MARKING PROCEDURE on (G, \mathcal{X}, k) . Moreover, let \mathcal{P} be a solution of (G, \mathcal{X}, k) (not necessarily minimum) such that the total number of edges used in \mathcal{P} is ℓ , where $\ell \leq 6k^{1.5}$. Furthermore, let $u \in U$ be an unmarked vertex such that u does not have any poor neighbor. If there is a path $P \in \mathcal{P}$ such that $u \in V(P)$, then there exists a solution $\mathcal{P}' = (\mathcal{P} \setminus \{P\}) \cup \{P'\}$ of (G, \mathcal{X}, k) such that $u \notin V(P')$, and the total number of edges in \mathcal{P}' is at most $\ell + 3$.

► **Lemma 32 (*)**. Let (G, \mathcal{X}, k) be an instance of EDP where G is a split graph. Let U be the set of unmarked vertices we get after an execution of MARKING PROCEDURE on (G, \mathcal{X}, k) , and let $u \in U$ be a vertex such that u does not have any poor neighbor. Let $G' = G - \{u\}$. Then, (G, \mathcal{X}, k) is a Yes-instance if and only if (G', \mathcal{X}, k) is a Yes-instance.

Since G' is a split graph (due to Remark 67), Lemma 32 implies the following.

► **Lemma 33.** *RR1 is safe.*

Next, we show that an exhaustive application of RR1 provides a subcubic vertex kernel.

► **Lemma 34** (*). *Let (G, \mathcal{X}, k) be an instance of EDP where G is a split graph. If we cannot apply RR1 on (G, \mathcal{X}, k) , then $|V(G)| = \mathcal{O}(k^{2.75})$.*

Finally, due to PREPROCESSING STEP, RR1, Lemmas 33 and 34, and the observation that RR1, MARKING PROCEDURE, and PREPROCESSING STEP can be executed in polynomial time (and do not increase our parameter k) we have the following theorem.

► **Theorem 2.** *EDP on split graphs admits a kernel with at most $\mathcal{O}(k^{2.75})$ vertices.*

3.2 A Quadratic Vertex Kernel for Block Graphs

In this section, we show that EDP on block graphs admits a kernel with at most $4k^2 - 2k$ vertices. Let us first discuss the overall idea leading us to this result.

Overview. Let (G, \mathcal{X}, k) be an instance of EDP where G is a block graph. First, we aim to construct a reduced instance where the number of blocks can be at most $4k - 2$. We begin by showing that if there is an end block that does not contain any terminal, then we can delete this block from the graph (in RR2), while preserving all solutions. Next, we argue that if there is a block, say, B , with at most two cut vertices that do not contain any terminal, then we can either *contract* (defined in Definition 37) B to a single vertex, or answer negatively (in RR4). Thus, each block with at most two cut vertices in the (reduced) graph contains at least one terminal. This bounds the number of blocks with at most two cut vertices to be at most $2k$ (as k terminal pairs yield at most $2k$ terminals). First, we observe the following.

► **Observation 35** (*). *Let ℓ be the number of end blocks in a block graph G . Then, the number of blocks with at least three cut vertices is at most $\ell - 2$.*

Observation 35, along with the fact that the number of end blocks is at most $2k$, establishes that the number of blocks with at least three cut vertices in the (reduced) graph is at most $2k - 2$. Therefore, we have at most $4k - 2$ blocks in the (reduced) graph. Finally, due to Lemma 18 and the properties of block graphs, we show that if a block, say, B , is big enough (i.e., $|V(B)| > k$), then we can contract B to a single vertex while preserving the solutions (in RR3). Hence, each block contains at most k vertices, and thus, the total number of vertices in the (reduced) graph is at most $4k^2 - 2k$.

Our kernelization algorithm is based on the application of three reduction rules (RR2-RR4), discussed below, to an instance (G, \mathcal{X}, k) of EDP where G is a block graph.

► **Reduction Rule 2** (RR2). *If B is an end block of G with cut vertex v such that B does not contain any terminal, then $G' \Leftarrow G[V(G) \setminus (V(B) \setminus \{v\})]$ and $\mathcal{X}' \Leftarrow \mathcal{X}$.*

We have the following lemma to establish that RR2 is safe.

► **Lemma 36** (*). *RR2 is safe.*

The following definitions (Definitions 37 and 38) are crucial to proceed further in this section. Informally speaking, we contract a block B by replacing it with a (new) vertex v such that “edge relations” are preserved. We have the following formal definition.

► **Definition 37** (Contraction of a Block). *Let (G, \mathcal{X}, k) be an instance of EDP where G is a block graph, and let B be a block of G . The contraction of B in (G, \mathcal{X}, k) yields another instance (G', \mathcal{X}', k') of EDP as follows. First, $V(G') = (V(G) \setminus V(B)) \cup \{v\}$ (i.e., delete $V(B)$ and add a new vertex v). Moreover, define $f : V(G) \rightarrow V(G')$ such that $f(x) = x$ if $x \in V(G) \setminus V(B)$, and $f(x) = v$ if $x \in V(B)$. Second, $E(G') = \{f(x)f(y) : xy \in E(G), f(x) \neq f(y)\}$. Similarly, $\mathcal{X}' = \{(f(s), f(t)) : (s, t) \in \mathcal{X}, f(s) \neq f(t)\}$. Finally, let $k' = |\mathcal{X}'|$. Note that k' might be smaller than k (in case B contains a terminal pair). Moreover, $\bigcup_{u \in V(B)} (N_G(u) \setminus B) \subseteq N_{G'}(v)$.*

We will exploit the properties of block graphs to show that if a block B has at least $k + 1$ vertices, then we can contract B to a single vertex “safely”. To this end, we define the instance (G, \mathcal{X}, k) of EDP restricted to a block B of the block graph G as follows.

► **Definition 38** (Restriction of an Instance (G, \mathcal{X}, k) to a Block). *Consider a block B whose set of cut vertices is U . For each cut vertex $u \in U$, let $C_{B,u}$ denote the (connected) component of $G[V(G) \setminus (V(B) \setminus \{u\})]$ containing u . Now, define $h : V(G) \rightarrow V(B)$ such that $h(x) = x$ if $x \in V(B)$, and $h(x) = u$ if $x \in V(C_{B,u})$. Then, the restriction of (G, \mathcal{X}, k) to B , denoted by $(B, \mathcal{X}_B, |\mathcal{X}_B|)$, is defined as follows: $\mathcal{X}_B = \{(h(s), h(t)) : (s, t) \in \mathcal{X}, h(s) \neq h(t)\}$.*

We have the following trivial observation that we will use for our proofs.

► **Observation 39**. *Let \mathcal{P} be a set of edge-disjoint paths. Consider a set of paths \mathcal{P}^* constructed in the following manner. For each path $P \in \mathcal{P}$, add a path P^* to \mathcal{P}^* such that $E(P^*) \subseteq E(P)$. Then, \mathcal{P}^* is also a set of edge-disjoint paths.*

We have the following lemma.

► **Lemma 40** (*). *Let (G, \mathcal{X}, k) be an instance of EDP on a block graph G , and let B be a block of G . If (G, \mathcal{X}, k) is a Yes-instance, then $(B, \mathcal{X}_B, |\mathcal{X}_B|)$ is a Yes-instance.*

Next, we will show that if for a block B , $(B, \mathcal{X}_B, |\mathcal{X}_B|)$ is a Yes-instance, then we can contract B “safely”. In particular, we have the following lemma.

► **Lemma 41** (*). *Let (G, \mathcal{X}, k) be an instance of EDP on a block graph G , and let B be a block of G whose set of cut vertices is U . Moreover, let the contraction of B in (G, \mathcal{X}, k) yield (G', \mathcal{X}', k') . Given that $(B, \mathcal{X}_B, |\mathcal{X}_B|)$ is a Yes-instance, (G, \mathcal{X}, k) is a Yes-instance if and only if (G', \mathcal{X}', k') is a Yes-instance.*

Next, we have the following reduction rule.

► **Reduction Rule 3** (RR3). *If G has a block B such that $|V(B)| > k$, then contract B in (G, \mathcal{X}, k) to get (G', \mathcal{X}', k') .*

The safeness of RR3 is implied by Lemma 41 and Lemma 18.

► **Corollary 42**. *RR3 is safe.*

Finally, we have the following reduction rule.

► **Reduction Rule 4** (RR4). *Let B be a block of G that has exactly two cut vertices, say, u and w , and there is no terminal vertex in $V(B) \setminus \{u, w\}$. Consider the instance $(B, \mathcal{X}_B, |\mathcal{X}_B|)$ restricted to the block B . If $|V(B)| > |\mathcal{X}_B|$, then contract B in (G, \mathcal{X}, k) to get the instance (G', \mathcal{X}', k') . Else, answer negatively.*

We have the following lemma to establish that RR4 is safe.

► **Lemma 43** (*). *RR4 is safe.*

Next, we establish that once we cannot apply RR2-RR4 anymore, the size of the reduced graph is bounded by a quadratic function of k . In particular, we have the following lemma.

► **Lemma 44** (*). *Let (G, \mathcal{X}, k) be an instance of EDP where we cannot apply reduction rules RR2-RR4 anymore. Then, G contains at most $4k - 2$ blocks and $4k^2 - 2k$ vertices.*

Observe that RR2-RR4 can be implemented in polynomial time and do not increase our initial parameter k . Hence, Lemmas 36, 43, 44, and Corollary 42 imply the following.

► **Theorem 4**. *EDP on block graphs admits a kernel with at most $4k^2 - 2k$ vertices.*

3.3 A Linear Vertex Kernel for Clique Paths

A *clique path* is a block graph where each block has at most two cut vertices. (In an informal manner, we can think that the blocks are arranged in the form of a path.) In this section, we present a linear vertex kernel for EDP on clique paths. First, we present an overview of the overall idea leading to this result.

Overview. Let (G, \mathcal{X}, k) be an instance of EDP where G is a clique path. Our kernelization algorithm is based on the application of RR2-RR4 (defined in Section 3.2) along with three new reduction rules (RR5-RR7). In our kernel for block graph in Section 3.2, we established that for a block B , if $|V(B)| \geq k + 1$, then we can contract B (see RR3). Moreover, we showed that the total number of blocks in a reduced instance can be at most $4k - 2$, thus giving us an $\mathcal{O}(k^2)$ vertex kernel.

Here, we use the property of clique paths that each block can have at most two cut vertices to improve the kernel size. Since there is no block with more than two cut vertices, each block must contain a terminal after an exhaustive application of RR2-RR4. Let B be a block of G with cut vertices u and w . Consider the instance $(B, \mathcal{X}_B, |\mathcal{X}_B|)$, that is, the instance (G, \mathcal{X}, k) restricted to the block B (see Definition 38). Any terminal pair $(s, t) \in \mathcal{X}_B$ is of one of the following types: (i) Type-A: $s, t \in \{u, w\}$, (ii) Type-B: $s = u$ and $t \in V(B) \setminus \{u, w\}$, or vice versa, (iii) Type-C: $s = w$ and $t \in V(B) \setminus \{u, w\}$, or vice versa, and (iv) Type-D: $s, t \in V(B) \setminus \{u, w\}$. Let a, b, c , and d denote the cardinality of Type-A, Type-B, Type-C, and Type-D occurrences of terminal pairs in \mathcal{X}_B , respectively. We show that if $|V(B)| > d + 2 + \max\{b + c - 1, 0\}$, then we can either contract B to a single vertex “safely” (when $|V(B)| > \max\{a + b, a + c\}$) or report a No-instance. Summing these numbers over all blocks yields an upper bound on the total size of the reduced instance. The Type-A pairs are irrelevant now, each pair of Type-B or Type-C contributes to two blocks, while each Type-D pair appears in only a single block. By grouping the summands in an appropriate way, we are able to attain a bound of $2k + 1$. We have the following reduction rules.

► **Reduction Rule 5** (RR5). *Let (G, \mathcal{X}, k) be an instance of EDP where G is a clique path. Moreover, let B be a block of G such that B has two cut vertices, say, u and w . Consider the instance $(B, \mathcal{X}_B, |\mathcal{X}_B|)$. If $|V(B)| \leq \max\{a + b, a + c\}$, then report a No-instance.*

We have the following lemma to prove that RR5 is safe.

► **Lemma 45** (*). *RR5 is safe.*

Next, we have the following reduction rule.

► **Reduction Rule 6 (RR6).** *Let (G, \mathcal{X}, k) be an instance of EDP where G is a clique path. Moreover, let B be a block of G that has two cut vertices, say, u and w . Consider the instance $(B, \mathcal{X}_B, |\mathcal{X}_B|)$. If $|V(B)| > d + 2 + \max\{b + c - 1, 0\}$, then contract B in (G, \mathcal{X}, k) to obtain the instance (G', \mathcal{X}', k') .*

We have the following lemma to establish that RR6 is safe.

► **Lemma 46 (*)**. *RR6 is safe.*

In RR6, we showed that in a reduced instance, the size of each block with two cut vertices is bounded by a linear function of the number of times its vertices appear in some occurrences of terminal pairs. In the next reduction rule (RR7), we consider the end blocks (i.e., blocks with exactly one cut vertex). So, consider an end block B of G with cut vertex u , and let $(B, \mathcal{X}_B, |\mathcal{X}_B|)$ be the restriction of (G, \mathcal{X}, k) to B . Any terminal pair $(s, t) \in \mathcal{X}_B$ is one of the following types: (i) Type-B': $s = u$ and $t \in V(B) \setminus \{u\}$, or vice versa, and (ii) Type-D': $s, t \in V(B) \setminus \{u\}$. Let b' and d' denote the number of occurrences of Type-B' and Type-D' terminal pairs in \mathcal{X}_B , respectively. We have the following reduction rule.

► **Reduction Rule 7 (RR7).** *Let (G, \mathcal{X}, k) be an instance of EDP where G is a clique path. Let B be an end block of G with cut vertex u . Consider the instance $(B, \mathcal{X}_B, |\mathcal{X}_B|)$. If $|V(B)| > b' + d'$, then contract B in (G, \mathcal{X}, k) to obtain (G', \mathcal{X}', k') .*

We have the following lemma to prove that RR7 is safe.

► **Lemma 47 (*)**. *RR7 is safe.*

Next, we establish that once we cannot apply RR2-RR7, the number of vertices of the reduced graph is bounded by a linear function of k in the following lemma.

► **Lemma 48 (*)**. *Let (G, \mathcal{X}, k) be an instance of EDP where G is a clique path. If we cannot apply RR2-RR7 on this instance, then G contains at most $2k + 1$ vertices.*

Now, we have the following observation.

► **Observation 49.** *RR5-RR7 can be applied in polynomial time. Moreover, during the application of RR5-RR7, we never increase the initial k .*

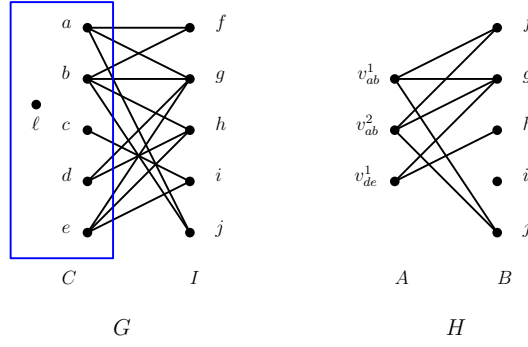
Finally, due to RR2-RR7, along with Lemmas 36, 43, 46, 47, and 48, Corollary 42, and Observation 49, we have the following theorem.

► **Theorem 5.** *EDP on clique paths admits a kernel with at most $2k + 1$ vertices.*

4 Kernelization Results on VDP on Split Graphs

Let (G, \mathcal{X}, k) be an instance of VDP where G is a split graph. Note that given a split graph G , we can compute (in linear time) a partition (C, I) of $V(G)$ such that C is a clique and I is an independent set [26]. We partition the set I into two sets, say, I_T and I_N , where I_T denotes the set of terminal vertices in I and I_N denotes the set of non-terminal vertices in I . Observe that a terminal pair in a split graph can only be of one of the following types: (i) Type-I: One of the terminal vertices belongs to C , and the other belongs to I , (ii) Type-II: Both terminal vertices belong to C , and Type-III: Both terminal vertices belong to I .

Our kernelization algorithm is based on a preprocessing step and the application of three reduction rules (RR8-RR10). We begin by defining the CLEAN-UP operation, which is a preprocessing step of our kernelization algorithm. Therefore, given an instance (G, \mathcal{X}, k) of VDP where G is a split graph, we apply this operation before applying RR8-RR10.



■ **Figure 1** Let (G, \mathcal{X}, k) be an instance of VDP where G is a split graph with a partition (C, I) and $\mathcal{X} = \{(a, b), (a, b), (a, b), (d, e), (d, e), (c, d), (b, d)\}$. The vertices in C form a clique; to keep the picture clean, we do not show edges in C . Observe that the terminal pairs (a, b) and (d, e) are heavy.

Clean-Up. First, consider the following construction (Construction \mathcal{A}), which is crucial to define the CLEAN-UP operation. This construction was also used by Heggenes et al. [27] who used it to remove a subset of I_N (safely). However, in the CLEAN-UP operation, we will remove the entire set I_N (safely). In order to do so, we need a few more technical arguments than the ones present in [27].

Construction \mathcal{A} . Given an instance (G, \mathcal{X}, k) of VDP where G is a split graph, we construct a bipartite graph, say, H , with bipartition (A, B) as follows: For every terminal pair (s, t) of Type-II such that st is a heavy edge with weight $w \geq 2$, we introduce $w - 1$ vertices, say, $v_{st}^1, \dots, v_{st}^{w-1}$, to A . The set B consists of all the vertices from set I_N . For each $v \in I_N$, introduce an edge from v to vertices $v_{st}^1, \dots, v_{st}^{w-1}$ if and only if v is adjacent to both s and t in G . See Figure 1 for an illustration of the construction of H from (G, \mathcal{X}, k) .

Due to Proposition 10, we compute a maximum matching, say, M , in H in polynomial time. Let $\hat{\mathcal{X}} \subseteq \mathcal{X}$ be the multiset of all terminal pairs whose corresponding vertices in H are saturated by M . For example, in Figure 1, if $M = \{v_{ab}^1 f, v_{ab}^2 g, v_{de}^1 h\}$ is a maximum matching in H , then $\hat{\mathcal{X}} = \{(a, b), (a, b), (d, e)\}$. This ends Construction \mathcal{A} .

► **Definition 50** (CLEAN-UP). *Given an instance (G, \mathcal{X}, k) of VDP where G is a split graph, construct the bipartite graph H and find a maximum matching M in H , as described in Construction \mathcal{A} . If $I_N \neq \emptyset$ or $\hat{\mathcal{X}} \neq \emptyset$, then $G' \leftarrow G - I_N$, $\mathcal{X}' \leftarrow \mathcal{X} \setminus \hat{\mathcal{X}}$, $k' \leftarrow k - |\hat{\mathcal{X}}|$.*

Note that due to Remark 67, G' is a split graph. Now, with the help of Definition 52, Observation 53, we will establish that (G, \mathcal{X}, k) and (G', \mathcal{X}', k') , as described in Definition 50, are equivalent (see Lemma 54).

Before that, consider the next observation which follows trivially from Proposition 16.

► **Observation 51.** *Let (G, \mathcal{X}, k) be a Yes-instance of VDP where G is a split graph. Moreover, let \mathcal{P} be a minimum solution of (G, \mathcal{X}, k) . If there exists a path $P \in \mathcal{P}$ that visits a vertex $x \in I_N$, then P must be of the form (u', x, v') , where $u', v' \in C$ are terminal vertices such that $P_{u'v'} \in \mathcal{P}$.*

► **Definition 52.** *Let (G, \mathcal{X}, k) be a Yes-instance of VDP where G is a split graph. Moreover, let M and H be as described in Construction \mathcal{A} . Let \mathcal{P} be a minimum solution of (G, \mathcal{X}, k) . Then, $M' \subseteq E(H)$ is said to be induced by \mathcal{P} in H if it is constructed as follows:*

1. Initialize $M' \leftarrow \emptyset$.
2. For every path $P \in \mathcal{P}$ that visits a vertex $x \in I_N$: By Observation 51, P must be of the form (u', x, v') , where $u', v' \in C$ are terminal vertices such that $P_{u'v'} \in \mathcal{P}$. This further implies that $u'v'$ is a heavy edge. Let $w \geq 2$ be the weight of $u'v'$, and consider the vertices $v_{u'v'}^1, \dots, v_{u'v'}^{w-1}$ in graph H . By Construction \mathcal{A} , x is adjacent to every vertex in the set $\{v_{u'v'}^1, \dots, v_{u'v'}^{w-1}\}$. So, we can arbitrarily choose an edge $xv_{u'v'}^j$ of H , for some $j \in [w-1]$ such that $xv_{u'v'}^j$ does not appear in M'^1 , and add it to M' .

► **Observation 53** (*). Let (G, \mathcal{X}, k) be a Yes-instance of VDP where G is a split graph. Moreover, let M, A, B , and H be as described in Construction \mathcal{A} . Let \mathcal{P} be a minimum solution of (G, \mathcal{X}, k) . Let M' be induced by \mathcal{P} in H as described in Definition 52. Then, M' is a matching.

For an illustrative example, consider G, H , and \mathcal{X} given in Figure 1. Let $\mathcal{P} = \{(a, f, b), (a, \ell, b), P_{ab}, (d, g, e), P_{de}, P_{cd}, P_{bd}\}$ be a solution of $(G, \mathcal{X}, 7)$. Then, $\{v_{ab}^1 f, v_{de}^1 g\}$ and $\{v_{ab}^2 f, v_{de}^2 g\}$ are two possible choices for M' .

► **Lemma 54** (*). Let (G, \mathcal{X}, k) be an instance of VDP where G is a split graph. Moreover, let $\hat{\mathcal{X}}$ be as described in Construction \mathcal{A} . Furthermore, let (G', \mathcal{X}', k') be the output of CLEAN-UP on (G, \mathcal{X}, k) . Then, if (G', \mathcal{X}', k') is a Yes-instance of VDP, then (G, \mathcal{X}, k) is also a Yes-instance of VDP.

Now, consider the following lemma.

► **Lemma 55** (*). Let (G, \mathcal{X}, k) be an instance of VDP obtained by applying CLEAN-UP. Let $(s, t) \in \mathcal{X}$ be a heavy terminal pair of Type-II in G of weight $w \geq 2$. Then, any minimum solution of (G, \mathcal{X}, k) must contain the following internally vertex-disjoint paths: $\{P_{st}\} \cup \{(s, u^1, t), \dots, (s, u^{w-1}, t)\}$, where $\{u^1, \dots, u^{w-1}\}$ is a set of non-terminals in C .

Reduction Rules. Let us start by defining our first reduction rule (RR8).

► **Reduction Rule 8** (RR8). If there is a terminal pair $(s, t) \in \mathcal{X}$ of Type-I such that $st \in E(G)$, then $V(G') \leftarrow V(G)$, $E(G') \leftarrow E(G) \setminus \{st\}$, $\mathcal{X}' \leftarrow \mathcal{X} \setminus \{(s, t)\}$, $k' \leftarrow k - 1$. Furthermore, for every $x \in \{s, t\}$ that does not appear as a terminal in any terminal pair in \mathcal{X}' , update $V(G') \leftarrow V(G') \setminus \{x\}$.

We have the following lemma to establish that RR8 is safe.

► **Lemma 56** (*). RR8 is safe.

► **Observation 57.** After applying RR8 exhaustively on G , no Type-I terminal pair in G has an edge between its terminals. Moreover, RR8 can be applied in polynomial time.

Let $\{(s, t) \times (w)\}$ denote w copies of (s, t) .

► **Reduction Rule 9** (RR9). If there is a heavy terminal pair $(s, t) \in \mathcal{X}$ of Type-II of weight $w \geq 2$, then $V(G') \leftarrow V(G) \cup \{s^1, \dots, s^{w-1}, t^1, \dots, t^{w-1}\}$, $E(G') \leftarrow E(G) \cup \{s^i v, t^i v : v \in C, i \in [w-1]\}$, $\mathcal{X}' \leftarrow (\mathcal{X} \setminus \bar{\mathcal{X}}) \cup \{(s^1, t^1), \dots, (s^{w-1}, t^{w-1})\}$, where $\bar{\mathcal{X}} = \{(s, t) \times (w-1)\} \subseteq \mathcal{X}$.

We have the following lemma to establish that RR9 is safe.

¹ The existence of such a j follows from the choice of w , and since $P_{u'v'} \in \mathcal{P}$

10:16 Kernels for the Disjoint Paths Problem on Subclasses of Chordal Graphs

► **Lemma 58** (*). *After CLEAN-UP and the exhaustive application of RR8, RR9 is safe.*

► **Observation 59**. *After applying RR9 exhaustively, there do not exist any heavy Type-II terminal pairs. Moreover, RR9 can be applied in polynomial time.*

The next reduction rule is applied for every terminal participating in more than one terminal pair.

► **Reduction Rule 10** (RR10). *If $v \in V(G)$ belongs to $x \geq 2$ terminal pairs $(v, a_1), \dots, (v, a_x)$, then $V(G') \Leftarrow (V(G) \setminus \{v\}) \cup \{v_1, \dots, v_x\}$, $E(G') \Leftarrow E(G) \cup \{v_i u : u \in N(v), i \in [x]\}$, $\mathcal{X}' \Leftarrow (\mathcal{X} \setminus \{(v, a_1), \dots, (v, a_x)\}) \cup \{(v_1, a_1), \dots, (v_x, a_x)\}$. Moreover, if $v \in C$, then $E(G') = E(G) \cup \{v_i v_j : i \neq j \in [x]\}$.*

We have the following lemma to establish that RR10 is safe.

► **Lemma 60** (*). *After CLEAN-UP and an exhaustive application of RR8 and RR9, RR10 is safe.*

By Lemma 55 and Observations 57 and 59, we have the following lemma.

► **Observation 61**. *After applying reduction rules RR8-RR10 exhaustively, no terminal participates in more than one terminal pair.*

Before concluding this section, we need the following result by Yang et al. [47].

► **Proposition 62** ([47]). *VDP-UNIQUE on split graphs admits a kernel with at most $4k$ vertices, where k is the number of occurrences of terminal pairs.*

By Observations 61, we note that every instance (G, \mathcal{X}, k) of VDP where G is a split graph, can be converted to an instance (G', \mathcal{X}', k') of VDP-UNIQUE in polynomial time. Due to Observations 57 and 59 and Lemmas 54, 56, 58, and 60, (G, \mathcal{X}, k) and (G', \mathcal{X}', k) are equivalent. Furthermore, our initial parameter k does not increase during the application of the CLEAN-UP operation and rules RR8-RR10. Therefore, using Proposition 62, we have the following theorem.

► **Theorem 6**. *VDP on split graphs admits a kernel with at most $4k$ vertices.*

5 Conclusion

In this paper, we studied VDP and EDP, two disjoint paths problems in the realm of Parameterized Complexity. We analyzed these problems with respect to the natural parameter “the number of (occurrences of) terminal pairs”. We gave several improved kernelization results as well as new kernelization results for these problems on subclasses of chordal graphs.

For VDP, we provided a $4k$ vertex kernel on split graphs and an $\mathcal{O}(k^2)$ vertex kernel on well-partitioned chordal graphs. We also show that VDP becomes polynomial-time solvable on threshold graphs. For EDP, we first proved NP-hardness on complete graphs. Second, we provided an $\mathcal{O}(k^{2.75})$ vertex kernel on split graphs, a $7k + 1$ vertex kernel on threshold graphs, an $\mathcal{O}(k^2)$ vertex kernel for EDP on block graphs, and a $2k + 1$ vertex kernel on clique paths.

Apart from the obvious open questions to improve the sizes of the kernels we designed, it is interesting to determine if VDP or/and EDP admit polynomial kernels for chordal graphs. It is worth noting here that Golovach et al. [24] proved that it is unlikely for SET-RESTRICTED DISJOINT PATHS, a generalization of VDP where each terminal pair has to find its path from

a predefined set of vertices, to admit a polynomial kernel even on interval graphs. However, as noted by them, their reduction is heavily dependent on the sets designed for terminal pairs and thus cannot be directly generalized to VDP. Moreover, recently Włodarczyk and Zehavi [46] established that both VDP and EDP are unlikely to admit polynomial compression even when restricted to planar graphs. They also suggested to investigate the existence of polynomial kernels for chordal graphs for VDP and EDP.

Another interesting open problem is to study the kernelization complexity of EDP on well-partitioned chordal graphs. Note that EDP on well-partitioned chordal graphs is more subtle than VDP on the same. The reason is that, in VDP, a path between a non-adjacent terminal pair must be induced due to Proposition 16. This paves the way to define valid paths in the partition tree of the given well-partitioned chordal graph. However, the concept of valid paths (and, thus, the used techniques) fails for EDP, as a path in the solution can visit the bags of the partition tree in any weird manner. Furthermore, the approach for EDP on block graphs does not generalize to well-partitioned chordal graphs because the intersection of adjacent bags can be large (in well-partitioned chordal graphs), whereas, for a block graph G , there always exists a partition tree such that for any two consecutive bags, the corresponding boundary of one of the bags has size one.

References

- 1 I. Adler, S. G. Kolliopoulos, P. K. Krause, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Irrelevant vertices for the planar disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 122:815–843, 2017.
- 2 J. Ahn, L. Jaffke, O. J. Kwon, and P. T. Lima. Well-partitioned chordal graphs. *Discrete Mathematics*, 345(10):112985, 2022.
- 3 Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, 43(9):842–844, 1957.
- 4 H. L. Bodlaender, S. Thomassé, and A. Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theoretical Computer Science*, 412(35):4570–4578, 2011.
- 5 J. Chaudhary, H. Gahlawat, M. Włodarczyk, and M. Zehavi. Kernels for the disjoint paths problem on subclasses of chordal graphs. *arXiv:2309.16892*, 2023.
- 6 C. Chekuri, S. Khanna, and F. B. Shepherd. An $\mathcal{O}(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow. *Theory of Computing*, 2(7):137–146, 2006.
- 7 J. Chuzhoy and D. H. K. Kim. On approximating node-disjoint paths in grids. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015*, pages 187–211. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
- 8 J. Chuzhoy, D. H. K. Kim, and S. Li. Improved approximation for node-disjoint paths in planar graphs. In *Proceedings of the 48th annual ACM symposium on Theory of Computing, STOC 2016*, pages 556–569, 2016.
- 9 J. Chuzhoy, D. H. K. Kim, and R. Nimavat. Improved approximation for node-disjoint paths in grids with sources on the boundary. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, pages 38:1–38:14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- 10 J. Chuzhoy, D. H. K. Kim, and R. Nimavat. Almost polynomial hardness of node-disjoint paths in grids. *Theory of Computing*, 17(6):1–57, 2021. doi:10.4086/toc.2021.v017a006.
- 11 J. Chuzhoy, D. H. K. Kim, and R. Nimavat. New hardness results for routing on disjoint paths. *SIAM Journal on Computing*, 51(2):189–237, 2022.
- 12 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

- 13 R. Diestel. Graph theory 3rd ed. *Graduate texts in mathematics*, 173(33):12, 2005.
- 14 Z. Dvořák, D. Král', and R. Thomas. Coloring triangle-free graphs on surfaces. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 120–129. SIAM, 2009.
- 15 A. Ene, M. Mnich, M. Pilipczuk, and A. Risteski. On routing disjoint paths in bounded treewidth graphs. In *Proceedings of the 15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016*, pages 1–15. Schloss Dagstuhl, 2016.
- 16 S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.
- 17 Michael R Fellows. The lost continent of polynomial time: Preprocessing and kernelization. In *International Workshop on Parameterized and Exact Computation*, pages 276–277. Springer, 2006.
- 18 K. Fleszar, M. Mnich, and J. Spoerhase. New algorithms for maximum disjoint paths based on tree-likeness. In *Proceedings of the European Symposium on Algorithms, ESA 2016*, pages 1–17. Schloss Dagstuhl, 2016.
- 19 F. V. Fomin, D. Lokshtanov, S. Saurabh, and M. Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.
- 20 A. Frank. Packing paths, circuits, and cuts—a survey. *Paths, flows, and VLSI-layout*, 49:100, 1990.
- 21 R. Ganian and S. Ordyniak. The power of cut-based parameters for computing edge-disjoint paths. *Algorithmica*, 83:726–752, 2021.
- 22 N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.
- 23 P. A. Golovach, F. Panolan, A. Rai, and S. Saurabh. New algorithms for maximum disjoint paths based on tree-likeness. In *Proceedings of the European Symposium on Algorithms, ESA 2016*, pages 1–17. Schloss Dagstuhl, 2016.
- 24 P. A. Golovach, F. Panolan, A. Rai, and S. Saurabh. Parameterized complexity of set-restricted disjoint paths on chordal graphs. In *Proceedings of the 17th International Computer Science Symposium in Russia, CSR 2022, Virtual Event, June 29–July 1, 2022, Proceedings*, pages 152–169. Springer, 2022.
- 25 F. Gurski and E. Wanke. Vertex disjoint paths on clique-width bounded graphs. *Theoretical Computer Science*, 359(1–3):188–199, 2006.
- 26 P. L. Hammer and B. Simeone. The splittance of a graph. *Combinatorica*, 1:275–284, 1981.
- 27 P. Heggenes, P. V. Hof, E. J. V. Leeuwen, and R. Saei. Finding disjoint paths in split graphs. *Theory of Computing Systems*, 57(1):140–159, 2015.
- 28 J. E. Hopcroft and R. M. Karp. An $n^{\frac{5}{2}}$ algorithm for maximum matching in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.
- 29 F. Kammer, T. Tholey, O. J. Kwon, and P. T. Lima. The k-disjoint paths problem on chordal graphs. In *Proceedings of the 35th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2009*, pages 190–201. Springer, Berlin, 2009. doi:10.1007/978-3-642-11409-0_17.
- 30 R. M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5:45–68, 1975.
- 31 K. Kawarabayashi, Y. Kobayashi, and S. Kreuzer. An excluded half-integral grid theorem for digraphs and the directed disjoint paths problem. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing, STOC 2014*, pages 70–78, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591876.
- 32 K. Kawarabayashi, Y. Kobayashi, and B. Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424–435, 2012.
- 33 Y. Kobayashi and K. Kawarabayashi. Algorithms for finding an induced cycle in planar graphs and bounded genus graphs. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1146–1155. SIAM, 2009.

- 34 S. G. Kolliopoulos and C. Stein. Approximating disjoint-path problems using packing integer programs. *Mathematical Programming*, 99(1):63–87, 2004.
- 35 M. Kramer and J. V. Leeuwen. The complexity of wirerouting and finding minimum area layouts for arbitrary vlsi circuits. *Advances in Computing Research*, 2:129–146, 1984.
- 36 D. Lokshtanov, P. Misra, M. Pilipczuk, S. Saurabh, and M. Zehavi. An exponential time parameterized algorithm for planar disjoint paths. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 1307–1316, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384250.
- 37 D. Lokshtanov, S. Saurabh, and M. Zehavi. Efficient graph minors theory and parameterized algorithms for (planar) disjoint paths. In *Treewidth, Kernels, and Algorithms*, pages 112–128. Springer, 2020.
- 38 S. Natarajan and A. P. Sprague. Disjoint paths in circular arc graphs. *Nordic Journal of Computing*, 3:256–270, 1996.
- 39 T. Nishizeki, J. Vygen, and X. Zhou. The edge-disjoint paths problem is NP-complete for series-parallel graphs. *Discrete Applied Mathematics*, 115(1–3):177–186, 2001.
- 40 B. Reed. Rooted routing in the plane. *Discrete Applied Mathematics*, 57(2-3):213–227, 1995.
- 41 Bruce A Reed. Tree width and tangles: A new connectivity measure and some applications. *Surveys in combinatorics*, pages 87–162, 1997.
- 42 N. Robertson and P. D. Seymour. Graph minors XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.
- 43 N. Robertson and P. D. Seymour. Graph minors XXII. Irrelevant vertices in linkage problems. *Journal of Combinatorial Theory, Series B*, 102(2):530–563, 2012.
- 44 A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.
- 45 A. Srinivas and E. Modiano. Finding minimum energy disjoint paths in wireless ad-hoc networks. *Wireless Networks*, 11:401–417, 2005.
- 46 M. Włodarczyk and M. Zehavi. Planar disjoint paths, treewidth, and kernels. In *Proceedings of the 64th IEEE Symposium on Foundations of Computer Science (FOCS) 2023*, 2023.
- 47 Y. Yang, Y. R. Shrestha, W. Li, and J. Guo. On the kernelization of split graph problems. *Theoretical Computer Science*, 734:72–82, 2018.
- 48 X. Zhou, S. Tamura, and T. Nishizeki. Finding edge-disjoint paths in partial k-trees. *Algorithmica*, 26(1):3–30, 2000.

A Brief Survey of Related Works

Both VDP and EDP are known to be NP-complete for planar graphs [35], line graphs [27], and split graphs [27]. Moreover, VDP is known to be NP-complete on interval graphs [38] and grids [35] as well. Both VDP and EDP were studied from the viewpoint of structural parameterization. While VDP is FPT parameterized by treewidth [41], EDP remains NP-complete even on graphs with treewidth at most 2 [39]. Gurski and Wange [25] showed that VDP is solvable in linear time on co-graphs but becomes NP-complete for graphs with clique-width at most 6. As noted by Heggernes et al. [27], there is a reduction from VDP on general graphs to EDP on line graphs, and from EDP on general graphs to VDP on line graphs. Since a graph with treewidth ℓ has clique-width at most $2\ell + 2$ [25], EDP can also be shown to be NP-complete on graphs with clique-width at most 6 [27].

The Graph Minors theory of Robertson and Seymour provides some of the most important algorithmic results of the modern graph theory. Unfortunately, these algorithms, along with the $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$ algorithms for VDP and EDP (when k is fixed), respectively [42, 32], hide such big constants that they have earned a name for themselves: “galactic algorithms”. Since then, finding efficient FPT algorithms for VDP and EDP has been a tantalizing question for researchers. Several improvements have been made for the class of planar graphs [1, 36, 40, 43], chordal graphs [29], and bounded-genus graphs [40, 14, 33].

Concerning kernelization (in addition to the works surveyed in the introduction), we note that Ganian and Ordyniak [21] proved that EDP admits a linear kernel parameterized by the feedback edge set number. Recently, Golovach et al. [23] proved that SET-RESTRICTED DISJOINT PATHS, a variant of VDP where each terminal pair has to find its path from a predefined set of vertices, does not admit a polynomial compression on interval graphs unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

The optimization variants of VDP and EDP – MAXVDP and MAXEDP – are well-studied in the realm of approximation algorithms [6, 15, 18, 22, 34]. Chekuri et al. [6] gave an $\mathcal{O}(\sqrt{n})$ -approximation algorithm for MAXEDP on general graphs, matching the $\Omega(\sqrt{n})$ lower bound provided by Garg et al. [22]. Recently, MAXVDP has gained much attention on planar graphs [7, 8, 10, 11, 9]. Highlights of this line of works (MAXVDP on planar graphs) include an approximation algorithm with approximation ratio $\mathcal{O}(n^{\frac{9}{19}} \log^{\mathcal{O}(1)} n)$ [8], and hardness of approximating within a factor of $n^{\Omega(1/(\log \log n)^2)}$ [10].

B Preliminaries

B.1 Parameterized Complexity

Standard notions in Parameterized Complexity not explicitly defined here can be found in [12]. Let Π be an NP-hard problem. In the framework of Parameterized Complexity, each instance of Π is associated with a *parameter* k . We say that Π is *fixed-parameter tractable* (FPT) if any instance (I, k) of Π is solvable in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where f is some computable function of k . Two instances (I, k) and (I', k') (possibly of different problems) are *equivalent* if: (I, k) is a Yes-instance if and only if (I', k') is a Yes-instance.

A parameterized (decision) problem Π admits a *kernel* of size $f(k)$ for some computable function f that depends only on k if the following is true: There exists an algorithm (called a *kernelization algorithm*) that runs in $(|I| + k)^{\mathcal{O}(1)}$ time and translates any input instance (I, k) of Π into an equivalent instance (I', k') of Π such that the size of (I', k') is bounded by $f(k)$. If the function f is polynomial (resp., linear) in k , then the problem is said to admit a *polynomial kernel* (resp., *linear kernel*). It is well known that a decidable parameterized problem is FPT if and only if it admits a kernel [12].

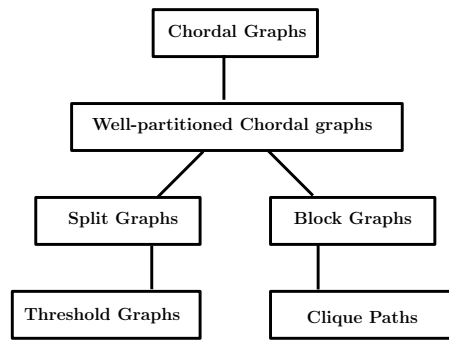
To design kernelization algorithms, we rely on the notion of *reduction rule*, defined below.

► **Definition 63** (Reduction Rule). *A reduction rule is a polynomial-time procedure that consists of a condition and an operation, and its input is an instance (I, k) of a parameterized problem Π . If the condition is true, then the rule outputs a new instance (I', k') of Π such that $k' \leq k$.*

A reduction rule is *safe*, when the condition is true, (I, k) and (I', k') are equivalent. Throughout this paper, the reduction rules will be numbered, and the reduction rules will be applied exhaustively in the increasing order of their indices. So, if reduction rules i and j , where $i < j$, are defined for a problem, then i will be applied exhaustively before j . Notice that after the application of rule j , the condition of rule i might become true. In this situation, we will apply rule i again (exhaustively). In other words, when we apply rule j , we always assume that the condition of rule i is false.

B.2 Graph Classes

A graph G is a *chordal graph* if every cycle in G of length at least four has a *chord*, that is, an edge joining two non-consecutive vertices of the cycle. In what follows, we define several subclasses of the class of chordal graphs, namely, *complete graphs*, *block graphs*, *split*



■ **Figure 2** The inclusive relationship among the subclasses of chordal graphs studied in this paper.

graphs, threshold graphs, and well-partitioned chordal graphs. A graph whose vertex set is a clique is a *complete graph*. A vertex is a *cut vertex* in a graph G if removing it increases the total number of connected components in G . A *block* of a graph G is a maximal connected subgraph of G that does not contain any cut vertex. A graph G is a *block graph* if the vertex set of every block in G is a clique. A block of a block graph G is an *end block* if it contains exactly one cut vertex. Note that a block graph that is not a complete graph has at least two end blocks. A graph G is a *split graph* if there is a partition (C, I) of $V(G)$ such that C is a clique and I is an independent set. A split graph G is a *threshold graph* if there exists a linear ordering of the vertices in I , say, $(v_1, v_2, \dots, v_{|I|})$, such that $N(v_1) \subseteq N(v_2) \subseteq \dots \subseteq N(v_{|I|})$.

An undirected graph in which any two vertices are connected by exactly one path is a *tree*. A tree with at most two vertices or exactly one non-pendant vertex is a *star*. The vertices of degree one in a tree are called *leaves*. Note that a split graph admits a partition of its vertex set into cliques that can be arranged in a star structure, where the leaves are cliques of size one. Motivated by this definition of split graphs, Ahn et al. [2] introduced *well-partitioned chordal graphs*, which are defined by relaxing the definition of split graphs in the following two ways: (i) by allowing the parts of the partition to be arranged in a tree structure instead of a star structure, and (ii) by allowing the cliques in each part to have arbitrary size instead of one. A more formal definition of a well-partitioned chordal graph is given below.

► **Definition 64** (Well-Partitioned Chordal Graph). *A connected graph G is a well-partitioned chordal graph if there exists a partition \mathcal{B} of $V(G)$ and a tree \mathcal{T} having \mathcal{B} as its vertex set such that the following hold.*

- (i) *Each part $X \in \mathcal{B}$ is a clique in G .*
- (ii) *For each edge $XY \in E(\mathcal{T})$, there are subsets $X' \subseteq X$ and $Y' \subseteq Y$ such that $E(G[X, Y]) = X' \times Y'$.*
- (iii) *For each pair of distinct $X, Y \in \mathcal{B}$ with $XY \notin E(\mathcal{T})$, $E(G[X, Y]) = \emptyset$.*

The tree \mathcal{T} in Definition 64 is called a *partition tree* of G , and the elements of \mathcal{B} are called its *bags*. Notice that a well-partitioned chordal graph can have multiple partition trees.

► **Proposition 65** ([2]). *Given a well-partitioned chordal graph G , a partition tree of G can be found in polynomial time.*

► **Definition 66** (Boundary of a Well-Partitioned Chordal Graph). *Let \mathcal{T} be a partition tree of a well-partitioned chordal graph G and let $XY \in E(\mathcal{T})$. The boundary of X with respect to Y , denoted by $\text{bd}(X, Y)$, is the set of vertices of X that have a neighbor in Y , i.e., $\text{bd}(X, Y) = \{x \in X : N_G(x) \cap Y \neq \emptyset\}$.*

► **Remark 67.** Note that all the graph classes considered in this paper are closed under vertex deletion. Therefore, throughout this paper, if G belongs to class \mathcal{Z} and G' is obtained from G after deleting some vertices, then we assume that G' also belongs to \mathcal{Z} without mentioning it explicitly.

The inclusion relationship among various subclasses of chordal graphs discussed in this paper is shown in Figure 2.

C NP-hardness for Complete Graphs

In this section, we prove that EDP is NP-hard on complete graphs by giving a polynomial-time reduction from EDP on general graphs, which is known to be NP-hard [35]. Our reduction is based on the standard technique of adding missing edges and placing a terminal pair on the endpoints of the added edge. This technique was also used to prove the NP-hardness of EDP for split graphs [27].

► **Theorem 1.** *EDP is NP-hard on complete graphs.*

Proof. Let (G, \mathcal{X}, k) be an instance of EDP, where $\mathcal{X} = \{(s_1, t_1), \dots, (s_k, t_k)\}$. Define the graph G' as follows: Let $V(G') = V(G)$ and $E(G') = E(G) \cup \{uv : uv \notin E(G)\}$. Furthermore, let $\mathcal{T} = \{(s_{uv}, t_{uv}) : uv \in E(G') \setminus E(G), s_{uv} = u, t_{uv} = v\}$. Note that for ease of notation, we denote the terminal pairs in \mathcal{T} by (s_{uv}, t_{uv}) rather than (u, v) . Also, for an edge $uv \in E(G') \setminus E(G)$, we introduce either (s_{uv}, t_{uv}) or (s_{vu}, t_{vu}) in \mathcal{T} , not both. Let $\mathcal{X}' = \mathcal{X} \cup \mathcal{T}$. Now, we claim that (G, \mathcal{X}, k) and (G', \mathcal{X}', k') are equivalent instances of EDP, where $k' = k + |\mathcal{T}|$. Let $\mathcal{P}_{\mathcal{T}} = \{P_{uv} : (s_{uv}, t_{uv}) \in \mathcal{T}\}$.

In one direction, let $\mathcal{P} = \{P_1, \dots, P_k\}$ be a solution of (G, \mathcal{X}, k) . Since for every $(s_{uv}, t_{uv}) \in \mathcal{T}$, the edge uv does not belong to any path in \mathcal{P} , $\mathcal{P} \cup \mathcal{P}_{\mathcal{T}}$ is a set of edge-disjoint paths in G' . As $\mathcal{X}' = \mathcal{X} \cup \mathcal{T}$, $\mathcal{P} \cup \mathcal{P}_{\mathcal{T}}$ is a solution of (G', \mathcal{X}', k') .

In the other direction, let \mathcal{P}' be a solution of (G', \mathcal{X}', k') that contains as many paths from $\mathcal{P}_{\mathcal{T}}$ as possible. Next, we claim that $\mathcal{P}_{\mathcal{T}} \subseteq \mathcal{P}'$. Targeting a contradiction, suppose that there exists a terminal pair, say, $(s_{uv}, t_{uv}) \in \mathcal{T}$, such that $P_{uv} \notin \mathcal{P}'$. Let Q denote the path in \mathcal{P}' connecting s_{uv} and t_{uv} . If none of the paths in \mathcal{P}' uses the edge uv , then the set $(\mathcal{P}' \setminus Q) \cup \{P_{uv}\}$ is a solution of (G', \mathcal{X}', k') containing more paths from $\mathcal{P}_{\mathcal{T}}$ than \mathcal{P}' , contradicting the choice of \mathcal{P}' . Hence, there must be a unique path $P^* \in \mathcal{P}'$ that uses the edge uv . Let s^* and t^* be the two terminals that are connected by the path P^* . Let W denote the walk between s^* and t^* obtained from P^* by replacing the edge uv with the path Q (there may be some vertices that are repeated in W). Since $E(W) = (E(P^*) \cup E(Q)) \setminus \{uv\}$, W is edge-disjoint from every path in $\mathcal{P}' \setminus \{P^*, Q\}$ (as \mathcal{P}' is a set of edge-disjoint paths). Let Q^* be a path between s^* and t^* that uses a subset of edges of W , which again is edge-disjoint from every path in $\mathcal{P}' \setminus \{P^*, Q\}$. Hence, $\mathcal{P} = (\mathcal{P}' \setminus \{P^*, Q\}) \cup \{P_{uv}, Q^*\}$ is a solution of (G', \mathcal{X}', k') that contains one more path from $\mathcal{P}_{\mathcal{T}}$ than \mathcal{P}' . This contradicts the choice of \mathcal{P}' , and thus implies that $\mathcal{P}_{\mathcal{T}} \subseteq \mathcal{P}'$. Since $\mathcal{X}' = \mathcal{X} \cup \mathcal{T}$ and $\mathcal{P}_{\mathcal{T}}$ contains the edge-disjoint paths between the terminal pairs present in \mathcal{T} , $\mathcal{P}' \setminus \mathcal{P}_{\mathcal{T}}$ must contain the edge-disjoint paths between the terminal pairs present in \mathcal{X} . Thus, $\mathcal{P}' \setminus \mathcal{P}_{\mathcal{T}}$ is a solution of (G, \mathcal{X}, k) . ◀

Parameterized Complexity Classification for Interval Constraints

Konrad K. Dabrowski   

School of Computing, Newcastle University, UK

Peter Jonsson   

Department of Computer and Information Science, Linköping University, Sweden

Sebastian Ordyniak  

School of Computing, University of Leeds, UK

George Osipov   

Department of Computer and Information Science, Linköping University, Sweden

Marcin Pilipczuk   

Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland
IT University Copenhagen, Denmark

Roohani Sharma   

Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

Constraint satisfaction problems form a nicely behaved class of problems that lends itself to complexity classification results. From the point of view of parameterized complexity, a natural task is to classify the parameterized complexity of MINCSP problems parameterized by the number of unsatisfied constraints. In other words, we ask whether we can delete at most k constraints, where k is the parameter, to get a satisfiable instance. In this work, we take a step towards classifying the parameterized complexity for an important infinite-domain CSP: *Allen's interval algebra* (IA). This CSP has closed intervals with rational endpoints as domain values and employs a set \mathcal{A} of 13 basic comparison relations such as “precedes” or “during” for relating intervals. IA is a highly influential and well-studied formalism within AI and qualitative reasoning that has numerous applications in, for instance, planning, natural language processing and molecular biology. We provide an FPT vs. W[1]-hard dichotomy for MINCSP(Γ) for all $\Gamma \subseteq \mathcal{A}$. IA is sometimes extended with unions of the relations in \mathcal{A} or first-order definable relations over \mathcal{A} , but extending our results to these cases would require first solving the parameterized complexity of DIRECTED SYMMETRIC MULTICUT, which is a notorious open problem. Already in this limited setting, we uncover connections to new variants of graph cut and separation problems. This includes hardness proofs for simultaneous cuts or feedback arc set problems in directed graphs, as well as new tractable cases with algorithms based on the recently introduced *flow augmentation* technique. Given the intractability of MINCSP(\mathcal{A}) in general, we then consider (parameterized) approximation algorithms. We first show that MINCSP(\mathcal{A}) cannot be polynomial-time approximated within any constant factor and continue by presenting a factor-2 fpt-approximation algorithm. Once again, this algorithm has its roots in flow augmentation.

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

Keywords and phrases (minimum) constraint satisfaction problem, Allen's interval algebra, parameterized complexity, cut problems

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.11

Related Version *Full Version:* arXiv:2305.13889

Funding *Peter Jonsson:* Partially supported by the Swedish Research Council (VR) under grant 2021-0437 and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.



© Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, Marcin Pilipczuk, and Roohani Sharma;

licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 11; pp. 11:1–11:19

Leibniz International Proceedings in Informatics



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

Sebastian Ordyniak: Supported by the Engineering and Physical Sciences Research Council (EPSRC), project EP/V00252X/1).

George Osipov: Supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

Marcin Pilipczuk: During this research Marcin was part of BARC, supported by the VILLUM Foundation grant 16582.

1 Introduction

Background. The *constraint satisfaction problem* over a constraint language Γ ($\text{CSP}(\Gamma)$) is the problem of deciding whether there is a variable assignment which satisfies a set of constraints, where each constraint is constructed from a relation in Γ . CSPs over different constraint languages form a nicely behaved class of problems that lends itself to complexity classification results. Such results are an important testbed for studying the power of algorithmic techniques and proving their limitations – A prime example is the dichotomy theorem for finite-domain CSPs that was conjectured by Feder and Vardi [28] and independently proved by Bulatov [14] and Zhuk [54]. Here, all hardness results were known since the work of Bulatov, Jeavons and Krokhin [15] and the algorithms of [14] and [54] completed the proof of the conjecture. In between, *lots* of work went into studying the problem from various algorithmic and algebraic angles, and many ideas emerging from this project have been re-used in different contexts (such as infinite-domain CSPs [8] or *promise CSPs* [42]). Optimization versions of the CSP such as MAXCSP and MINCSP (where the goal is to find an assignment that maximises the number of satisfied constraints (MAXCSP) or minimises the number of unsatisfied constraints (MINCSP)) and the generalization Valued CSP (VCSP) have also been intensively studied. Some notable results include the proof of that every finite-domain VCSP is either polynomial-time solvable or NP-complete [40], and the optimal approximability result for finite-domain MAXCSP under the Unique Games Conjecture [50]. One should note that even if the P/NP borderline for finite-domain VCSPs is fully known, there are big gaps in our understanding of the corresponding FPT/W[1] borderline (with parameter solution weight). The situation is even worse if we consider infinite-domain optimization versions of the CSPs, since we cannot expect to get a full picture of the P/NP borderline even for the basic CSP problem [9].

In the parameterized complexity world, MINCSP is a natural problem to study. Subproblems that have gained attraction include Boolean constraint languages [13, 37, 51], Dechter et al.’s [25] simple temporal problem (STP) [22], linear inequalities [5] and linear equations [20, 24], to list a few. Highly interesting results have emerged from studying the parameterized complexity of problems like these. For instance, the recent dichotomy for MINCSPs over the Boolean domain by Kim et al. [37] was obtained using a novel technique called *directed flow augmentation*. Recent work has indicated *temporal* CSPs as a possible next step [27, 38]. Temporal CSPs are CSPs where the relations underlying the constraints are first-order definable in $(\mathbb{Q}; <)$. The computational complexity of temporal CSPs where we fix the set of allowed constraints exhibits a dichotomy: every such problem is either polynomial-time solvable or NP-complete [11]. The MINCSP problem for temporal CSPs is closely related to a number of graph separation problems. For example, if we take the rationals as the domain and allow constraints \leq and $<$, the MINCSP problem is equivalent to DIRECTED SUBSET FEEDBACK ARC SET [39], a problem known to be fixed-parameter tractable for two different, but both quite involved reasons [17, 39]. If we allow the relations \leq and \neq , we obtain a problem equivalent to DIRECTED SYMMETRIC MULTICUT, whose

parameterized complexity is identified as the main open problem in the area of directed graph separation problems [27, 39]. Another related way forward is to analyse the MINCSP for *Allen’s interval algebra*. Allen’s interval algebra is a highly influential formalism within AI and qualitative reasoning that has numerous applications, e.g. in planning [4, 48, 49], natural language processing [26, 52] and molecular biology [31]. This CSP uses closed intervals with rational endpoints as domain values and employs a set \mathcal{A} of 13 basic comparison relations such as “precedes” (one interval finishes before the other starts) or “during” (one interval is a strict subset of the other); see Table 1. Formally speaking, the CSP for the interval algebra is not a temporal CSP, since the underlying domain is based on intervals instead of points. This difference is important: complexity classifications for the interval algebra have been harder to obtain than for temporal constraints. There are full classifications for binary relations [41] and for first-order definable constraint languages containing all basic relations [10]; a classification for all first-order definable constraint languages appears remote.

Our contributions. The aim in this paper is to initiate a study of MINCSP in the context of Allen’s interval algebra. Obtaining a full parameterized complexity classification for Allen’s interval algebra would entail resolving the status of DIRECTED SYMMETRIC MULTICUT and we do not aim at this very ambitious task. Instead, we restrict ourselves to languages that are subsets of \mathcal{A} and do not consider more involved expressions (say, first-order logic) built on top of \mathcal{A} . Even in this limited quest, we are able to uncover new relations to graph separation problems, and new areas of both tractability and intractability. One of the main ingredients for both our tractability and intractability results is a particular characterization of unsatisfiable instances of $\text{CSP}(\mathcal{A})$. A combinatorial analysis of the relations in \mathcal{A} allows us to identify the minimal obstructions given by certain arc-labelled mixed cycles of the instance. That is, for certain key subsets $\Gamma \subseteq \mathcal{A}$, we provide a complete description of *bad* cycles such that an instance of $\text{CSP}(\Gamma)$ is satisfiable if and only if it does not contain a bad cycle. This allows us to show that $\text{MINCSP}(\Gamma)$ is equivalent to the problem of finding a minimum set of arcs that hit every bad cycle in an arc-labelled mixed graph. We prove that there are seven inclusion-wise maximal subsets Γ of \mathcal{A} such that $\text{MINCSP}(\Gamma)$ is in FPT, and that $\text{MINCSP}(\Gamma)$ is W[1]-hard in all other cases. We show that $\text{MINCSP}(\mathcal{A})$ is not approximable in polynomial time within *any* constant under the UGC. In fact, we prove this to hold for $\text{MINCSP}(r)$ whenever $r \in \mathcal{A} \setminus \{\equiv\}$. As a response to this, we suggest the use of *fixed-parameter approximation algorithms*. We show that $\text{MINCSP}(\mathcal{A})$ admits such an algorithm with approximation ratio 2 and a substantially faster algorithm with approximation ratio 4. We describe the results in greater detail below.

Intractability results. Our intractability results are based on novel W[1]-hardness results for a variety of natural *paired* and *simultaneous* cut and separation problems, which we believe to be of independent interest. Here, the input consists of two (directed or undirected) graphs and the task is to find a “generalized” cut that extends to both graphs. The two input graphs share some arcs/edges that can be deleted simultaneously at unit cost, and the goal is to compute a set of k arcs/edges that is a cut in both graphs. Both paired and simultaneous problems have recently received attention from the parameterized complexity community [1, 2, 37]. In the FPT/W[1]-hardness dichotomy for the Boolean domain [37], the fundamental difficult problem is PAIRED CUT (proven to be W[1]-hard by Marx and Razgon [47]): given an integer k and a directed graph G with two terminals $s, t \in V(G)$ and some arcs grouped into *pairs*, delete at most k pairs to cut all paths from s to t . An intuitive reason why PAIRED CUT is difficult can be seen as follows. Assume G contains two long

st -paths P and Q and the arcs of P are arbitrarily paired with the arcs of Q . Then, one cuts both paths with a cost of only one pair, but the arbitrary pairing of the arcs allow us to encode an arbitrary permutation – which is very powerful for encoding edge-choice gadgets when reducing from MULTICOLOURED CLIQUE. Our strategy for proving $W[1]$ -hardness of paired problems elaborates upon this idea. In this case, the needed gadgets are quite succinct and their construction is simplified by the fact that we can recycle ideas from [24]. Our hardness proofs for simultaneous problems also use the same underlying idea, but they are much more complicated. The simultaneous setting is obviously not as versatile as the arbitrary pairing of PAIRED CUT. However, the possibility of choosing common arcs/edges while deleting them at unit costs still leaves enough freedom to encode arbitrary permutations. Such permutations enable us to construct edge-choice gadgets by “simulating” the low-level features that are available for free in paired problems. The resulting construction is complex and appears to contain ideas that may be useful for proving hardness of other kinds of simultaneous problems.

Altogether, we obtain novel $W[1]$ -hardness results for PAIRED CUT FEEDBACK ARC SET, SIMULTANEOUS ST-SEPARATOR, SIMULTANEOUS DIRECTED ST-CUT, and SIMULTANEOUS DIRECTED FEEDBACK ARC SET. This allows us to identify six intractable fragments that are subsets of basic interval relations: $\{m, r_1\}$ for $r_1 \in \{\equiv, s, f\}$, $\{d, r_2\}$ for $r_2 \in \{o, p\}$ and $\{p, o\}$. The hardness reduction for $\{m, r_1\}$, is based on the hardness of PAIRED CUT and PAIRED CUT FEEDBACK ARC SET. The other hardness results are based on reductions from SIMULTANEOUS DIRECTED FEEDBACK ARC SET, whose $W[1]$ -hardness is shown by a reduction from SIMULTANEOUS ST-SEPARATOR. The reductions from SIMULTANEOUS DIRECTED FEEDBACK ARC SET are non-trivial but both their presentation and their correctness proofs are highly simplified by our concrete descriptions of bad cycles.

Tractability results. We identify seven maximal tractable sets of basic interval relations: $\{m, p\}$ and $\{r_1, r_2, \equiv\}$ for $r_1 \in \{s, f\}$ and $r_2 \in \{p, d, o\}$. All problems are handled by reductions to variants of DIRECTED FEEDBACK ARC SET (DFAS). DFAS and variations are extensively studied in parameterized complexity [6, 12, 16, 17, 29, 30, 45, 46]. DFAS is equivalent to MINCSP($<$), and a variant particularly important in our work is SUB-DFAS equivalent to MINCSP($<, \leq$), where the goal is to destroy only directed cycles that have a $<$ -arc. To show that MINCSP(m, p) is in FPT, we use a straightforward reduction to MINCSP($<, =$) which, in turn, reduces to MINCSP($<, \leq$).

For the remaining six tractable cases, we reduce them all to a new variant of DFAS called MIXED FEEDBACK ARC SET WITH SHORT AND LONG ARCS (LS-MFAS). The input is a mixed graph with edges and long and short arcs. Forbidden cycles in this graph are of two types: (1) cycles with at least one short arc, no long arcs, and all short arcs in the same direction, and (2) cycles with at least one long arc, all long arcs in the same direction, but the short arcs can be traversed in arbitrary direction. One intuitive way to think about the problem is to observe that a graph G has no forbidden cycle if there exists a placement of the vertices on the number line with certain distance constraints represented by the edges and arcs. Vertices connected by edges (which correspond to \equiv -constraints) should be placed at the same point. If there is an arc (u, v) , we need to place u before v . Moreover, if the arc is long, then the distance from u to v should be big (say, greater than twice the number of vertices), while if the arc is short, then the distance from u to v should be small (say, at most 1). The reduction from MINCSP(r_1, r_2, \equiv) to LS-MFAS creates a mixed graph with edges for \equiv -constraints, short arcs for r_1 -constraints and long arcs for r_2 -constraint. For correctness, consider for example the case with $r_1 = s$ and $r_2 = p$. Forbidden cycles of the first

and the second kind imply an unsatisfiable order on the right and left endpoints, respectively. On the other hand, if bad cycles are absent, we can assign intervals as follows: if two variables are \equiv -connected, they are assigned the same interval, if they are s -connected, their intervals have the same left endpoints, left endpoints are ordered according to p -constraints, and the right endpoints of s -connected intervals are ordered according to the s -constraints.

Our algorithm for LS-MFAS builds upon the algorithm of [39] for SUB-DFAS. By iterative compression (see e.g. Chapter 4 in [21]) and branching, we may assume access to $k + 1$ vertices that intersect all forbidden cycles, and we know their relative positions in the graph obtained after deleting a hypothetical optimal solution. The aforementioned “placing on a line” way of phrasing the lack of forbidden cycles is the main reason why this leads to a complete algorithm. In the next step, we try to place all remaining vertices relative to the terminals while breaking at most k distance constraints. Note that the number of ways in which a vertex can relate to a terminal is constant: it may be placed at a short/long distance before/after the terminal, or in the same position. Thus, we can define $O(k)$ types for each vertex, and the types determine whether distance constraints are satisfied or not. The optimal type assignment is then obtained by a reduction to BUNDLED CUT with pairwise-linked deletable arcs, a workhorse problem shown to be fixed-parameter tractable in [37]. We remark that our algorithms also handle the weighted versions of the problems.

Approximation results. In response to the negative complexity results for $\text{MINCSP}(\mathcal{A})$, we consider approximation algorithms. We show that $\text{MINCSP}(\mathcal{A})$ is not approximable in polynomial time within any constant under the UGC. We relax the restrictions even more by allowing our approximation algorithms to run in fixed-parameter tractable time. We show that $\text{MINCSP}(\mathcal{A})$ admits such an algorithm with $c = 2$ and a substantially faster algorithm with $c = 4$. Hence, fpt-approximation is *much* more powerful than ordinary polynomial-time approximation in this case. These results are based on the observation that every relation in \mathcal{A} can be defined as a conjunction of $\{<, =\}$ -constraints on the endpoints. In the relaxation, we disregard conjunctions and view all $\{<, =\}$ -constraints as an instance of $\text{MINCSP}(<, =)$, which is then reduced to SUB-DFAS. By invoking the SUB-DFAS algorithm of [39], one obtains a 2-approximation algorithm for the weighted variant of the problem.

Roadmap. We present the necessary preliminaries in Section 2. Section 3 is a bird’s eye view of our results for the parameterized complexity and approximability of $\text{MINCSP}(\Gamma)$ and the technical details are collected in the following sections. We describe the minimal obstructions to satisfiability for certain subsets of \mathcal{A} in Section 4. These results are essential for connecting the $\text{MINCSP}(\mathcal{A})$ problem with the graph-oriented view that we use. We complete our dichotomy result by a number of fixed-parameter algorithms in Section 5 and a collection of $W[1]$ -hardness results in Section 6. We conclude the paper with a discussion of our results and future research directions in Section 7. This is a shortened version of the full paper, which can be found on arXiv [23].

2 Preliminaries

In this section, we briefly present the rudiments of parameterized complexity, define the CSP and MINCSP problems, and provide some basics concerning interval relations. Before we begin, we need some terminology and notation for graphs. Let G be a (directed or undirected) graph; we allow graphs to contain loops. We denote the set of vertices in G by $V(G)$. If G is undirected, then $E(G)$ denotes the set of edges in G . If G is directed, then $A(G)$ denotes the

11:6 Parameterized Complexity Classification for Interval Constraints

■ **Table 1** The thirteen basic relations in Allen’s Interval Algebra. The endpoint relations $I^- < I^+$ and $J^- < J^+$ that are valid for all relations have been omitted.

Basic relation	Example	Endpoint Relations
I precedes J J preceded by I	p pi	$I^+ < J^-$
I meets J J met-by I	m mi	$I^+ = J^-$
I overlaps J J overlapped-by I	o oi	$I^- < J^- < I^+ < J^+$
I during J J includes I	d di	$I^- > J^-, I^+ < J^+$
I starts J J started by I	s si	$I^- = J^-, I^+ < J^+$
I finishes J J finished by I	f fi	$I^+ = J^+, I^- > J^-$
I equals J	\equiv	$I^- = J^-, I^+ = J^+$

set of arcs in G , and $E(G)$ denotes the set of edges in the underlying undirected graph of G . We use uv to denote an undirected edge with end-vertices u and v . We use (u, v) to denote a directed arc from u to v ; u is the *tail* and v is the *head*. For $X \subseteq E(G)$, we write $G - X$ to denote the directed graph obtained by removing all edges/arcs corresponding to X from G if G is undirected and $A(G - X) = A(G) \setminus \{(u, v), (v, u) \mid \{u, v\} \in X\}$ if G is directed. If $X \subseteq V(G)$, then we let $G - X = G[V(G) \setminus X]$ be the subgraph induced in G by $V(G) \setminus X$. An *st*-cut in G is a set of edges/arcs X such that the vertices s and t are separated in $G - X$.

A *parameterized* problem is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is the input alphabet. The parameterized complexity class FPT contains the problems decidable in $f(k) \cdot n^{O(1)}$ time, where f is a computable function and n is the instance size. Reductions between parameterized problems need to take the parameter into account. To this end, we use *parameterized reductions* (or fpt-reductions). Consider two parameterized problems $L_1, L_2 \subseteq \Sigma^* \times \mathbb{N}$. A mapping $P : \Sigma^* \times \mathbb{N} \rightarrow \Sigma^* \times \mathbb{N}$ is a parameterized reduction from L_1 to L_2 if (1) $(x, k) \in L_1$ if and only if $P((x, k)) \in L_2$, (2) the mapping can be computed in $f(k) \cdot n^{O(1)}$ time for some computable function f , and (3) there is a computable function $g : \mathbb{N} \rightarrow \mathbb{N}$ such that for all $(x, k) \in \Sigma^* \times \mathbb{N}$, if $(x', k') = P((x, k))$, then $k' \leq g(k)$. We will sometimes prove that certain problems are not in FPT. The class W[1] contains all problems that are fpt-reducible to INDEPENDENT SET parameterized by the number of vertices in the independent set. Showing W[1]-hardness (by an fpt-reduction) for a problem rules out the existence of an fpt algorithm under the standard assumption that $\text{FPT} \neq \text{W}[1]$.

We continue by defining CSPs. A *constraint language* Γ is a set of relations over a domain D . Each relation $R \in \Gamma$ has an associated *arity* $r \in \mathbb{N}$ and $R \subseteq D^r$. All relations considered in this paper are binary and all constraint languages are finite. An instance \mathcal{I} of $\text{CSP}(\Gamma)$ consists of a set of variables $V(\mathcal{I})$ and a set of constraints $C(\mathcal{I})$ of the form $R(x, y)$, where $R \in \Gamma$ and $x, y \in V(\mathcal{I})$. To simplify notation, we may write $R(x, y)$ as xRy . An assignment $\varphi : V(\mathcal{I}) \rightarrow D$ *satisfies* a constraint $R(x, y)$ if $(\varphi(x), \varphi(y)) \in R$ and *violates* $R(x, y)$ if $(\varphi(x), \varphi(y)) \notin R$. The assignment φ is a *satisfying assignment* (or a *solution*) if it satisfies every constraint in $C(\mathcal{I})$.

CSP(Γ)	
INSTANCE:	An instance \mathcal{I} of CSP(Γ).
QUESTION:	Does \mathcal{I} admit a satisfying assignment?

The *value* of an assignment φ for \mathcal{I} is the number of constraints in $C(\mathcal{I})$ satisfied by φ . For any subset of constraints $X \subseteq C(\mathcal{I})$, let $\mathcal{I} - X$ denote the instance with $V(\mathcal{I} - X) = V(\mathcal{I})$ and $C(\mathcal{I} - X) = C(\mathcal{I}) \setminus X$. The (parameterized) *almost constraint satisfaction problem* (MINCSP(Γ)) is defined as follows:

MINCSP(Γ)	
INSTANCE:	An instance \mathcal{I} of CSP(Γ) and an integer k .
PARAMETER:	k .
QUESTION:	Is there a set $X \subseteq C(\mathcal{I})$ such that $ X \leq k$ and $\mathcal{I} - X$ is satisfiable?

Next, we review the basics of *Allen's interval algebra* [3] (IA). Its domain is the set \mathbb{I} of all pairs $(x, y) \in \mathbb{Q}^2$ such that $x < y$, i.e. \mathbb{I} can be viewed as the set of all closed intervals $[a, b]$ of rational numbers. If $I = [a, b] \in \mathbb{I}$, then we write I^- for a and I^+ for b . Let \mathcal{A} denote the set of 13 *basic* relations that are presented in Table 1, and let $2^{\mathcal{A}}$ denote the 8192 binary relations that can be formed by taking unions of relations in \mathcal{A} . The complexity of CSP(Γ) is known for every $\Gamma \subseteq 2^{\mathcal{A}}$ [41] and in each case CSP(X) is either polynomial-time solvable or NP-complete. In particular, CSP(\mathcal{A}) is in P. When considering subsets $\Gamma \subseteq \mathcal{A}$, note that any constraint $xriy$ is equivalent to yrx for $r \in \{\text{p, m, o, d, s, f}\}$, so we may assume that $r \in \Gamma$ if and only if $ri \in \Gamma$. Furthermore, for the remainder of the paper, we may assume $\mathcal{A} = \{\text{p, m, o, d, s, f, } \equiv\}$.

When studying MINCSP and its parameterized complexity, it is convenient to allow *crisp* constraints, i.e. constraints that cannot be deleted. Formally, for a language $\Gamma \subseteq \mathcal{A}$ and a relation $r \in \Gamma$, we say that Γ *supports crisp r -constraints* if, for every value of the parameter $k \in \mathbb{N}$, we can construct an instance \mathcal{I}_r of MINCSP(Γ) with variables $x, y \in V(\mathcal{I}_r)$ (and possibly some auxiliary variables) such that the constraint xry is equivalent to $\mathcal{I}_r - X$ for all $X \subseteq C(\mathcal{I}_r)$ such that $|X| \leq k$. Then, if we want to enforce a constraint xry in an instance of MINCSP(Γ), we can use \mathcal{I}_r with fresh variables $V(\mathcal{I}_r) \setminus \{x, y\}$ in its place. Straightforward reasoning about interval constraints readily shows that every $r \in \mathcal{A}$ supports crisp constraints, and this also holds for the constraint language $\{<, =\}$.

3 Overview

In this section we prove the dichotomy theorem for the parameterized complexity of MINCSP(Γ) for every subset $\Gamma \subseteq \mathcal{A}$ of interval relations. We also discuss constant-factor approximation algorithms for MINCSP(\mathcal{A}). Some observations reduce the number of subsets of relations that we need to consider in the classification. For the first one, we need a simplified definition of *implementations*. More general definitions are used in e.g. [33] and [35].

► **Definition 1.** *Let Γ be a constraint language and r be a binary relation over the same domain. A (simple) implementation of a relation r in Γ is an instance C_r of CSP(Γ) with primary variables x_1, x_2 and, possibly, auxiliary variables y_1, \dots, y_ℓ such that:*

- *if an assignment φ satisfies C_r , then it satisfies the constraint x_1rx_2 ;*
- *if an assignment φ' does not satisfy x_1rx_2 , then it cannot be extended to the auxiliary variables y_1, \dots, y_ℓ so that it satisfies C_r .*
- *if an assignment φ' does not satisfy x_1rx_2 , then it can be extended to the auxiliary variables y_1, \dots, y_ℓ so that all but one constraint in C_r are satisfied.*

In this case we say that Γ implements r .

Intuitively, we can replace every occurrence of a constraint xy with its implementation in Γ while preserving the cost of making the instance satisfiable. This intuition is made precise in the following lemma, and identifying the two implementations in Lemma 3 is left to the reader.

► **Lemma 2** (Proposition 5.2 in [35]). *Let Γ be a constraint language that implements a relation r . If $\text{MINCSP}(\Gamma)$ is in FPT, then so is $\text{MINCSP}(\Gamma \cup \{r\})$. If $\text{MINCSP}(\Gamma \cup \{r\})$ is $W[1]$ -hard, then so is $\text{MINCSP}(\Gamma)$.*

► **Lemma 3** (Implementations). *Let $\Gamma \subseteq \mathcal{A}$ be a subset of interval relations. If Γ contains m , then Γ implements p , and if Γ contains f and s , then Γ implements d and o .*

Another observation utilizes the symmetry of interval relations. By switching the left and the right endpoints of all intervals in an instance \mathcal{I} of $\text{MINCSP}(\mathcal{A})$ and then negating their values, we obtain a *reversed instance* \mathcal{I}^R . Formally, instance \mathcal{I}^R of $\text{CSP}(\mathcal{A})$ has the same set of variables as \mathcal{I} , and contains a constraint $uf(r)v$ for every urv in $C(\mathcal{I})$, where $f : \mathcal{A} \rightarrow \mathcal{A}$ is defined as $f(r) = ri$ for $r \in \{m, p, o\}$, $f(\equiv) = \equiv$, $f(d) = d$, $f(s) = f$ and $f(f) = s$.

► **Lemma 4** (Lemma 4.2 of [41]). *An instance \mathcal{I} of $\text{CSP}(\mathcal{A})$ is satisfiable if and only if the reversed instance \mathcal{I}^R is satisfiable.*

To obtain our results, we use combinatorial tools and represent an instance \mathcal{I} of $\text{CSP}(\mathcal{A})$ as an *arc-labelled mixed graph* $G_{\mathcal{I}}$, i.e. a graph that contains edges for symmetric constraints and labelled arcs for asymmetric ones. More precisely, the graph $G_{\mathcal{I}}$ is obtained by introducing all variables of \mathcal{I} as vertices, adding directed arcs (u, v) labelled with $r \in \mathcal{A} \setminus \{\equiv\}$ for every constraint urv in $C(\mathcal{I})$, and undirected edges uv for every constraint $u \equiv v$ in \mathcal{I} . Note that $G_{\mathcal{I}}$ may have parallel arcs with different labels and may contain loops. The undirected graph underlying $G_{\mathcal{I}}$ is called the *primal graph* of \mathcal{I} ; we allow the primal graph to contain loops and parallel edges (in both cases, this will mean the primal graph contains a cycle). The advantage of the graph representation is supported by the following lemma:

► **Lemma 5** (Cycles). *Let \mathcal{I} be an inclusion-wise minimal unsatisfiable instance of $\text{CSP}(\mathcal{A})$ (i.e. removing any constraint of \mathcal{I} results in a satisfiable instance). Then the primal graph of \mathcal{I} is a cycle.*

The proof of the lemma is deferred to Section 4. All cycles discussed in the rest of the section are cycles of the primal graph. From the combinatorial point of view, minimal unsatisfiable instances are *bad cycles* in the labelled graph. For example, in $\text{MINCSP}(p)$, the bad cycles correspond to the directed cycles. For $\text{MINCSP}(p, \equiv)$, the bad cycles contain at least one p -arc and all p -arcs in the same direction. Thus, $\text{MINCSP}(\Gamma)$ can now be cast as a certain feedback edge set problem – our goal is to find a set of k edges in the primal graph that intersects all bad cycles. We present such a characterization for several cases in Section 4.

Our algorithmic results can be summarized as follows.

► **Lemma 6.** *$\text{MINCSP}(m, p)$ and $\text{MINCSP}(r_1, r_2, \equiv)$ are in FPT for $r_1 \in \{s, f\}$ and $r_2 \in \{p, o, d\}$.*

The algorithm for $\text{MINCSP}(m, p)$ is obtained using a simple reduction to **SUBSET DIRECTED FEEDBACK ARC SET**.

SUBSET DIRECTED FEEDBACK ARC SET (SUB-DFAS)

INSTANCE: A directed graph G , a subset of red arcs $R \subseteq A(G)$, and an integer k .
 PARAMETER: k .
 QUESTION: Is there a subset $Z \subseteq A(G)$ of size at most k such that $G - Z$ contains no directed cycles with at least one red arc?

Chitnis et al. [17] have proved that SUB-DFAS is solvable in $O^*(2^{O(k^3)})$ time. The algorithm for the remaining cases is more complicated and relies on the bad cycle characterization in Section 4 and a sophisticated modification of the algorithm for SUB-DFAS from [39] in Section 5.

For the negative results, we start by proving $W[1]$ -hardness for certain paired and simultaneous graph cut problems, and we identify $\Gamma \subseteq \mathcal{A}$ such that paired or simultaneous problems reduce to $\text{MINCSP}(\Gamma)$. For intuition, consider a constraint $x \equiv y$. If we consider the left and the right endpoints separately, then \equiv implies two equalities: $x^- = y^-$ and $x^+ = y^+$. Together with another relation (e.g. m), this double-equality relation can be used to encode the pairing of the edges of two graphs (namely, the left-endpoint graph and the right-endpoint graph). We note that the double-equality relation is also the cornerstone of all hardness results in the parameterized complexity classification of BOOLEAN MINCSP [37]. Lemma 7 is based on paired problems and Lemma 8 is based on simultaneous problems.

► **Lemma 7.** $\text{MINCSP}(m, \equiv)$, $\text{MINCSP}(m, s)$ and $\text{MINCSP}(m, f)$ are $W[1]$ -hard.

► **Lemma 8.** $\text{MINCSP}(d, o)$, $\text{MINCSP}(p, o)$ and $\text{MINCSP}(d, p)$ are $W[1]$ -hard.

Combining all results above, we are ready to present the full classification.

► **Theorem 9 (Full classification).** Let $\Gamma \subseteq \mathcal{A}$ be a subset of interval relations. Then $\text{MINCSP}(\Gamma)$ is in FPT if $\Gamma \subseteq \{m, p\}$ or $\Gamma \subseteq \{r_1, r_2, \equiv\}$ for any $r_1 \in \{s, f\}$ and $r_2 \in \{p, o, d\}$, and $W[1]$ -hard otherwise.

$W[1]$ -hardness of $\text{MINCSP}(\mathcal{A})$ motivates us to look at approximation algorithms for this problem. Our first observation is that $\text{MINCSP}(r)$ for any $r \in \mathcal{A} \setminus \{\equiv\}$ is NP-hard to approximate within any constant under the Unique Games Conjecture (UGC) of Khot [34]. This follows by combining two facts: Lemma 11, which implies that an instance \mathcal{I} of $\text{CSP}(r)$ is satisfiable if and only if the arc-labelled graph $G_{\mathcal{I}}$ is acyclic, and Corollary 1.2 in [32], which states that under the UGC, $\text{DIRECTED FEEDBACK ARC SET (DFAS)}$ is NP-hard to approximate within any constant [32]. If we allow the approximation algorithm to run in fpt time, then we obtain the following result.

► **Theorem 10.** $\text{MINCSP}(\mathcal{A})$ is 2-approximable in $O^*(2^{O(k^3)})$ time and 4-approximable in $O^*(2^{O(k)})$ time.

Proof sketch. We obtain the algorithms by reducing the problem to SUB-DFAS and invoking the exact algorithm of [21] and the faster $O^*(2^{O(k)})$ time 2-approximation algorithm of [44], respectively. There are straightforward reductions from $\text{MINCSP}(<, =)$ to $\text{MINCSP}(\leq, =)$ to SUB-DFAS, so we focus on the reduction from $\text{MINCSP}(\mathcal{A})$ to $\text{MINCSP}(<, =)$. Let (\mathcal{I}, k) be an instance of $\text{MINCSP}(\mathcal{A})$. Replace every constraint $x\{o\}y$ by its implementation in $\{s, f\}$ according to Lemma 3. By Lemma 2, this does not change the cost of the instance. Using Table 1, we can rewrite all constraints of \mathcal{I}' as conjunctions of two atomic constraints of the form $x < y$ or $x = y$. Disregarding the pairing, let S be the set of all atomic constraints. Apply one of the $\text{MINCSP}(<, =)$ algorithms to $(S, 2k)$. On the one hand,

deleting k constraints from \mathcal{I}' corresponds to deleting at most $2k$ constraints in S . On the other hand, if there is $X \subseteq S$, $|X| \leq 2k$, such that $S - X$ is satisfiable, define the set of interval constraints X' such that at least one of the defining $\{<, =\}$ -constraints is in X . Noting that $\mathcal{I} - X'$ is satisfiable and $|X'| \leq |X| \leq 2k$ completes the proof. ◀

4 Bad Cycles

In this section, we sketch the proof of Lemma 5 and describe the minimal obstructions to satisfiability for certain subsets of \mathcal{A} , along with a brief sketch of why these are the minimal obstructions.

► **Lemma 5** (Cycles). *Let \mathcal{I} be an inclusion-wise minimal unsatisfiable instance of $\text{CSP}(\mathcal{A})$ (i.e. removing any constraint of \mathcal{I} results in a satisfiable instance). Then the primal graph of \mathcal{I} is a cycle.*

The proof of Lemma 5 starts by taking a minimal unsatisfiable instance \mathcal{I} . Using Table 1, we write \mathcal{I} as an instance \mathcal{I}' of the *point algebra* (PA) [53] CSP, which takes rationals \mathbb{Q} as the variable domain and we use only the basic constraint language $\{<, =\}$, where the relations are interpreted in the obvious way. This instance \mathcal{I}' must contain a minimal unsatisfiable sub-instance \mathcal{I}'' of the point algebra, which has a cycle as its primal graph. We then map the constraints in \mathcal{I}'' back to the constraints in \mathcal{I} that implied them, and find that \mathcal{I} must also have a cycle as its primal graph.

► **Lemma 11** (Bad Cycles). *Let \mathcal{I} be an instance of $\text{CSP}(r_1, r_2)$ for some $r_1, r_2 \in \mathcal{A}$, and consider the arc-labelled mixed graph $G_{\mathcal{I}}$. Then \mathcal{I} is satisfiable if and only if $G_{\mathcal{I}}$ does not contain any bad cycles described below.*

1. If $r_1 = \mathbf{d}$ and $r_2 = \mathbf{p}$, then the bad cycles are cycles with \mathbf{p} -arcs in the same direction and no \mathbf{d} -arcs meeting head-to-head.
2. If $r_1 = \mathbf{d}$ and $r_2 = \mathbf{o}$, then the bad cycles are cycles with all \mathbf{d} -arcs in the same direction and all \mathbf{o} -arcs in the same direction (the direction of the \mathbf{d} -arcs may differ from that of the \mathbf{o} -arcs).
3. If $r_1 = \mathbf{o}$ and $r_2 = \mathbf{p}$, then the bad cycles are (a) directed cycles of \mathbf{o} -arcs and (b) cycles with all \mathbf{p} -arcs in the forward direction, with every consecutive pair of \mathbf{o} -arcs in the reverse direction separated by a \mathbf{p} -arc (this case includes directed cycles of \mathbf{p} -arcs).
4. If $r_1 \in \{\mathbf{f}, \mathbf{s}\}$ and $r_2 \in \{\mathbf{d}, \mathbf{o}, \mathbf{p}\}$, then the bad cycles are (a) directed cycles of r_1 -arcs and (b) cycles with at least one r_2 -arc and all r_2 -arcs in the same direction (and r_1 -arcs directed arbitrarily).

5 FPT Algorithms

We prove Lemma 6 in this section. The fpt algorithm for $\{\mathbf{m}, \mathbf{p}\}$ is simple and we omit the details: it works by first reducing the problem to $\text{MINCSP}(<, =)$ and then to SUB-DFAS . The remaining six cases are handled by reducing them to a fairly natural generalization of $\text{DIRECTED FEEDBACK ARC SET}$ problem, and showing that this problem is in FPT.

Lemma 11.4 suggests that all six remaining fragments allow uniform treatment. Indeed, to check whether an instance of $\text{CSP}(r_1, r_2, \equiv)$ is satisfiable, one can identify all variables constrained to be equal. This corresponds exactly to contracting all edges in the graph $G_{\mathcal{I}}$. Then \mathcal{I} becomes an instance of $\text{CSP}(r_1, r_2)$, and the criterion of Lemma 11.4 applies. This observation allows us to formulate $\text{MINCSP}(r_1, r_2, \equiv)$ as a variant of feedback arc set on mixed graphs.

► **Definition 12.** Consider a mixed graph G with arcs of two types – short and long – and a walk W in G from u to v that may ignore direction of the arcs. The walk W is undirected if it only contains edges, it is short if it contains a short arc but no long arcs, and it is long if it contains a long arc. The walk W is directed if it is either short and all short arcs are directed from u to v or if it is long and all long arcs are directed from u to v . If W is short or long, but not directed, it is mixed.

Note that short arcs on a long-directed walk may be directed arbitrarily.

MIXED FEEDBACK ARC SET WITH SHORT AND LONG ARCS (LS-MFAS)

INSTANCE: A mixed graph G with the arc set $A(G)$ partitioned into *short* A_s and *long* A_ℓ , and an integer k .

PARAMETER: k .

QUESTION: Is there a set $Z \subseteq E(G) \cup A(G)$ with $|Z| \leq k$ such that $G - Z$ contains neither short-directed cycles nor long-directed cycles?

The main result of this section is the following theorem.

► **Theorem 13.** LS-MFAS can be solved in $O^*(2^{O(k^8 \log k)})$ time.

We see that Lemma 11.4 and Theorem 13 imply $\text{Mincsp}(r_1, r_2, \equiv)$ being in FPT whenever $r_1 \in \{\text{s}, \text{f}\}$ and $r_2 \in \{\text{p}, \text{o}, \text{d}\}$. It is informative to understand the structure of mixed graphs without bad cycles in the sense of LS-MFAS. The proof of the following lemma is fairly easy with the placing-vertices-on-the-number-line intuition from the introduction.

► **Lemma 14.** Let G be a mixed graph with long and short arcs. Then G contains no long-directed cycles nor short-directed cycles if and only if there exists a pair of mappings $\sigma_1, \sigma_2 : V(G) \rightarrow \mathbb{N}$ such that

1. for every $u, v \in V(G)$, u and v are connected by an undirected walk if and only if $(\sigma_1, \sigma_2)(u) = (\sigma_1, \sigma_2)(v)$;
2. for every $u, v \in V(G)$, there exists a short (u, v) -walk in G if and only if $\sigma_1(u) = \sigma_1(v)$;
3. for every $u, v \in V(G)$, if there exists a short-directed (u, v) -walk in G , then $\sigma_2(u) < \sigma_2(v)$;
4. for every $u, v \in V(G)$, if there exists a long-directed (u, v) -walk in G , then $\sigma_1(u) < \sigma_1(v)$.

We now introduce the technical machinery used in our algorithm for LS-MFAS. We start by using iterative compression, a standard method in parameterized algorithms (see e.g. Chapter 4 in [21]). This allows us to assume access to a set of $k + 1$ edges and arcs intersecting every bad cycle. The problem resulting from iterative compression reduces to BUNDLED CUT with pairwise-linked deletable edges, defined in [37] and solved using the flow-augmentation technique of [36]. To describe BUNDLED CUT, let G be a directed graph with two distinguished vertices $s, t \in V(G)$. Let \mathcal{B} be a family of pairwise disjoint subsets of $E(G)$, which we call *bundles*. The edges of $\bigcup \mathcal{B}$ are *soft* and the edges of $E(G) \setminus \bigcup \mathcal{B}$ are *crisp*. A set $Z \subseteq \bigcup \mathcal{B}$ *violates* a bundle $B \in \mathcal{B}$ if $Z \cap B \neq \emptyset$ and *satisfies* B otherwise.

BUNDLED CUT

INSTANCE: A directed graph G , two distinguished vertices $s, t \in V(G)$, a family \mathcal{B} of pairwise disjoint subsets of $E(G)$, and an integer k .

PARAMETER: k .

QUESTION: Is there an st -cut $Z \subseteq \bigcup \mathcal{B}$ that violates at most k bundles?

11:12 Parameterized Complexity Classification for Interval Constraints

■ **Table 2** Correspondence between edges, short arcs and long arcs of the LS-MFAS instance and the arcs introduced in the reduction to BUNDLED CUT in Theorem 13.

	i odd	i even, j odd	i even, j even
Edge uv	$(i, j) \rightarrow (i, j)$ $(i, j) \leftarrow (i, j)$	$(i, j) \rightarrow (i, j)$ $(i, j) \leftarrow (i, j)$	$(i, j) \rightarrow (i, j)$ $(i, j) \leftarrow (i, j)$
Short (u, v)	$(i, j) \rightarrow (i, j)$ $(i, 1) \leftarrow (i, j)$	$(i, j) \rightarrow (i, j)$ $(i, 1) \leftarrow (i, j)$	$(i, j) \rightarrow (i, j + 1)$ $(i, 1) \leftarrow (i, j)$
Long (u, v)	$(i, 1) \rightarrow (i, 1)$	$(i, j) \rightarrow (i + 1, 1)$	$(i, j) \rightarrow (i + 1, 1)$

In general, BUNDLED CUT is W[1]-hard even if all bundles are of size 2. However, there is a special case of BUNDLED CUT that is tractable. Let $(G, s, t, \mathcal{B}, k)$ be a BUNDLED CUT instance. A soft arc e is *deletable* if there is no crisp copy of e in G . An instance $(G, s, t, \mathcal{B}, k)$ has *pairwise-linked deletable arcs* if for every $B \in \mathcal{B}$ and every two deletable arcs $e_1, e_2 \in B$, there exists in G a path from an endpoint of one of the arcs e_1, e_2 to an endpoint of the second of those arcs that does not use any arcs of $\mathcal{B} \setminus \{B\}$. The assumption of pairwise-linked deletable arcs makes BUNDLED CUT tractable.

► **Theorem 15** (Theorem 4.1 of [38]). *BUNDLED CUT instances with pairwise-linked deletable arcs can be solved in $O^*(2^{O(k^4 d^4 \log(kd))})$ time, where d is the maximum number of deletable arcs in a single bundle.*

Armed with Lemma 14 and Theorem 15, we are ready to prove the main result.

Proof of Theorem 13. Let (G, A_s, A_t, k) be an instance of LS-MFAS. By iterative compression, we may assume that we have access to a set $Y \subseteq V(G)$ of size at most $k + 1$ that intersects all bad cycles. We refer to the vertices of Y as *terminals*.

Fix a hypothetical solution $Z \subseteq A(G) \cup E(G)$. Guess which pairs of terminals are connected by undirected paths in $G - Z$ and identify them. Define an *ordering* $\sigma : Y \rightarrow \mathbb{N} \times \mathbb{N}$ that maps terminals to

$$\{(1, 1), \dots, (1, q_1), \dots, (i, 1), \dots, (i, q_i), \dots, (p, 1), \dots, (p, q_p)\}$$

such that the following hold. For every pair of terminals $y, y' \in Y$, let $\sigma(y) = (i, j)$ and $\sigma(y') = (i', j')$ where (1) $i = i'$ if y and y' are connected by a short path in $G - Z$, (2) $j < j'$ if y reaches y' by a short-directed path in $G - Z$, and (3) $i < i'$ if y reaches y' by a long-directed path in $G - Z$. Note that σ exists by Lemma 14. If an ordering satisfies the conditions above, we say that it is *compatible with $G - Z$* . In what follows, we write $(i, j) < (i', j')$ to denote that (i, j) lexicographically precedes (i', j') , i.e. either $i = i'$ and $j < j'$ or $i < i'$.

For the algorithm, proceed by guessing an ordering σ , creating $2^{O(k \log k)}$ branches in total. For each σ , create an instance $(H := H(G, \sigma), \mathcal{B} := \mathcal{B}(G, \sigma), k)$ of BUNDLED CUT as follows. Introduce two distinguished vertices s and t in H . For every vertex $v \in V(G)$, create vertices v_1^i in H for all odd $i \in [2p + 1]$ and vertices v_j^i in H for all even $i \in [2p + 1]$ and all $j \in [2q_i + 1]$. Connect the vertices created above by *downward arcs* $(v_j^i, v_{j'}^{i'})$ for all $(i, j) > (i', j')$. For every terminal y , let $\sigma(y) = (i, j)$, and add arcs (s, y_{2j}^{2i}) and (y_{2j+1}^{2i}, t) in H . Using the rules below, create a bundle B_e in \mathcal{B} for every $e \in E(G) \cup A(G)$, add the newly created arcs to H .

- For an edge $e = uv$, let B_e consist of the arcs (u_j^i, v_j^i) and (v_j^i, u_j^i) for all (i, j) .
- For short arcs $e = (u, v)$, let B_e consist of the arcs (u_j^i, v_j^i) for all i, j such that i or j is odd, the arcs (u_j^i, v_{j+1}^i) for all even i, j , and the arcs (v_j^i, u_1^i) for all i, j .
- For long arcs $e = (u, v)$, let B_e consist of the arcs (u_1^i, v_1^i) for all odd i , and arcs the arcs (u_j^i, v_1^{i+1}) for all even i and all j .

This completes the construction. Bundle construction rules are summarized in Table 2. Observe that the downward arcs ensure that (H, \mathcal{B}, k) has the pairwise-linked deletable arc property. Moreover, the bundle size is $O(k)$, so we can solve (H, \mathcal{B}, k) in $O^*(2^{O(k^8 \log k)})$ time.

We now sketch the correctness argument. Fix a guessed ordering σ . For any candidate solution W in $(H := H(G, \sigma), \mathcal{B} = \mathcal{B}(G, \sigma), k)$, the existence of downward arcs imply that for every $v \in V(G)$ there is a threshold (i_v, j_v) such that v_j^j is reachable from s in $H - W$ if and only if $(i, j) \leq (i_v, j_v)$. This threshold is meant to indicate that v should be placed on the line somewhere around the terminal x for which $\sigma(x) = (\lfloor i_v/2 \rfloor, \lfloor j_v/2 \rfloor)$. A short walk from u to v in G projects, for every even i and even j , to a walk from u_i^j to v_i^{j+1} . A long walk from u to v in G projects, for every odd i , to a walk from u_i^1 to v_i^1 . Together with the fact that terminals intersect all forbidden cycles in G , this gives a correspondence between forbidden cycles in G and st -paths in H . ◀

6 W[1]-hard Problems

Here, we show Lemmas 7 and 8. As the first and most challenging step, we show $W[1]$ -hardness for variants of paired and simultaneous graph cut problems from which we then reduce to the hard variants of $\text{MINCSP}(\Gamma)$. Our reductions will make use of the following well-known problem, whose $W[1]$ -hardness follows by a simple reduction from $\text{MULTICOLOURED CLIQUE}$ (see e.g. Exercise 13.3 in [21]).

MULTICOLOURED BICLIQUE (MC-BICLIQUE)	
INSTANCE:	An undirected graph G with a partition $V(G) = A_1 \uplus \dots \uplus A_k \uplus B_1 \uplus \dots \uplus B_k$, where $ A_i = B_i = n$ for each $i \in [k]$ and both $\uplus_{i \in [k]} A_i$ and $\uplus_{i \in [k]} B_i$ form independent sets in G .
PARAMETER:	k .
QUESTION:	Does G contain $K_{k,k}$ as a subgraph, a.k.a. a <i>multicoloured biclique</i> ?

6.1 Paired Problems

We consider the problems PAIRED CUT and $\text{PAIRED CUT FEEDBACK ARC SET (PCFAS)}$ in what follows.

PAIRED CUT	
INSTANCE:	Undirected graphs G_1 and G_2 , vertices $s_i, t_i \in V(G_i)$, a set of disjoint edge pairs $\mathcal{B} \subseteq E(G_1) \times E(G_2)$, and an integer k .
PARAMETER:	k .
QUESTION:	Is there a subset $X \subseteq \mathcal{B}$ such that $ X \leq k$ and $X_i = \{e_i \mid \{e_1, e_2\} \in X\}$ is an $\{s_i, t_i\}$ -cut in G_i for both $i \in \{1, 2\}$?

The PCFAS problem is similar, but G_2 is directed and X_2 is required to be such that $G_2 - X_2$ is acyclic (instead of being an $\{s_2, t_2\}$ -cut). We show $W[1]$ -hardness of both problems. Since both reductions are from MC-BICLIQUE and quite similar, we start to describe the common part of both reductions. Let $I = (G, A_1, \dots, A_k, B_1, \dots, B_k, k)$ be an instance of MC-BICLIQUE. We define two directed graphs G_A and G_B as follows. G_A

11:14 Parameterized Complexity Classification for Interval Constraints

contains the vertices s_A and t_A . Moreover, for every $i \in [k]$, G_A contains the vertices in $P_i^A = \{v_{i,1}, \dots, v_{i,n-1}\}$. For convenience, we let $v_{i,0} = s_A$ and $v_{i,n} = t_A$ for every $i \in [k]$. Moreover, for every vertex $a_{i,j}$ and every $i' \in [k]$, G_A contains the directed path $P_{i,j,i'}^A$ from $v_{i,j-1}$ to $v_{i,j}$ that has one edge (using fresh auxiliary vertices) for every edge between $a_{i,j}$ and a vertex in $B_{i'}$. Therefore, we may assume in what follows that there is a bijection between the edges of $P_{i,j,i'}^A$ and the edges between $a_{i,j}$ and a vertex in $B_{i'}$. This concludes the description of G_A . G_B is defined very similarly to G_A with the roles of the sets A_1, \dots, A_k and B_1, \dots, B_k being reversed.

Finally, define a set $\mathcal{B} \subseteq E(G_A) \times E(G_B)$ of bundles as follows. For every edge $e = \{a_{i,j}, b_{i',j'}\} \in E(G)$, \mathcal{B} contains the pair (e^A, e^B) , where e^A is the edge corresponding to e on the path $P_{i,j,i'}^A$ and e^B is the edge corresponding to e on the path $P_{i',j',i}^B$. This concludes the construction and the following lemma shows its main property.

► **Lemma 16.** *$I = (G, A_1, \dots, A_k, B_1, \dots, B_k, k)$ is a yes-instance of MC-BICLIQUE if and only if there is a set $X \subseteq \mathcal{B}$ with $|X| = k^2$ and $X_A = \{e \mid (e, e') \in X\}$ is an (s_A, t_A) -cut in G_A and $X_B = \{e' \mid (e, e') \in X\}$ is an (s_B, t_B) -cut in G_B .*

The lemma above makes it relatively straightforward to show W[1]-hardness for PAIRED CUT and PCFAS.

► **Lemma 17.** *PAIRED CUT and PCFAS are W[1]-hard.*

6.2 Simultaneous Problems

In this section we prove W[1]-hardness of several simultaneous cut problems. Our basis is the following problem.

SIMULTANEOUS SEPARATOR (SIM-SEPARATOR)	
INSTANCE:	Two directed graphs D_1 and D_2 with $V = V(D_1) = V(D_2)$, vertices $s, t \in V$, and an integer k .
PARAMETER:	k .
QUESTION:	Is there a subset $X \subseteq V \setminus \{s, t\}$ of size at most k such that neither $D_1 - X$ nor $D_2 - X$ contains a path from s to t ?

We begin by proving that this problem is W[1]-hard in Theorem 18. We will then prove that simultaneous variants of DIRECTED ST-CUT and DIRECTED FEEDBACK ARC SET are W[1]-hard via reductions from SIM-SEPARATOR; these results can be found in Theorems 19 and 20, respectively. It will be convenient to use the term *st-separator* when working with directed graphs: given a directed graph $G = (V, E)$ and two vertices $s, t \in V$, we say that $X \subseteq V \setminus \{s, t\}$ is an *st-separator* if the graph $G - X$ contains no directed path from s to t and no directed path from t to s .

► **Theorem 18.** *SIM-SEPARATOR is W[1]-hard even if both input digraphs are acyclic.*

We continue by using the W[1]-hardness of SIM-SEPARATOR to prove W[1]-hardness of the following two problems.

SIMULTANEOUS DIRECTED ST-CUT (SIM-CUT)	
INSTANCE:	Two directed graphs D_1 and D_2 with $V = V(D_1) = V(D_2)$, vertices $s, t \in V$, and an integer k .
PARAMETER:	k .
QUESTION:	Is there a subset $X \subseteq E(D_1) \cup E(D_2)$ of size at most k such that neither $D_1 - X$ nor $D_2 - X$ contains a path from s to t ?

SIMULTANEOUS DIRECTED FEEDBACK ARC SET (SIM-DFAS)

INSTANCE: Directed graphs D_1, D_2 with $V = V(D_1) = V(D_2)$, and an integer k .
 PARAMETER: k .
 QUESTION: Is there a subset $X \subseteq E(D_1) \cup E(D_2)$ of size at most k such that both $D_1 - X$ and $D_2 - X$ are acyclic?

► **Theorem 19.** *SIM-CUT is $W[1]$ -hard even if both input digraphs are acyclic.*

► **Theorem 20.** *SIM-DFAS is $W[1]$ -hard.*

6.3 Intractable Fragments

We begin by proving that $\text{MINCSP}(m, \equiv)$ and $\text{MINCSP}(m, s)$ are $W[1]$ -hard. We introduce two binary relations: let \equiv^- denote the *left-equals* relation and \equiv^+ denote the *right-equals* relation, which hold for any pair of intervals with matching left endpoints and right endpoints, respectively. Both relations can be implemented using only m as follows: $\{zmx, zmy\}$ implements $x \equiv^- y$ where z is a fresh variable; similarly, $\{xmz, ymz\}$ implements $x \equiv^+ y$. Thus, we may assume that the relations \equiv^- and \equiv^+ are available whenever we have access to the m relation. We are now ready to present the reduction for $\text{MINCSP}(m, \equiv)$, which will be from the PAIRED CUT problem that was shown to be $W[1]$ -hard in Lemma 17.

► **Theorem 21.** *$\text{MINCSP}(m, \equiv)$ is $W[1]$ -hard.*

We continue by showing that $\text{MINCSP}(m, s)$ is $W[1]$ -hard. First note even though we no longer have access to \equiv , we can add the constraints $x \equiv^- y$ and $x \equiv^+ y$ which imply $x \equiv y$. As previously, the relations \equiv^- and \equiv^+ can be implemented using only m . We remark that $\{x \equiv^- y, x \equiv^+ y\}$ is *not* an implementation of \equiv , so we can only use \equiv in crisp constraints. Our reduction is based on the PCFAS problem, which was shown to be $W[1]$ -hard in Lemma 17. While the reduction is quite similar to the reduction for $\text{MINCSP}(m, \equiv)$, it is non-trivial to replace the role of \equiv with s .

► **Theorem 22.** *$\text{MINCSP}(m, s)$ is $W[1]$ -hard.*

We finally show the $W[1]$ -hardness of $\text{MINCSP}(d, p)$, $\text{MINCSP}(d, o)$, and $\text{MINCSP}(p, o)$ via parameterized reductions from SIM-DFAS (which is a $W[1]$ -hard problem by Theorem 20).

► **Theorem 23.** *$\text{MINCSP}(d, p)$, $\text{MINCSP}(d, o)$, and $\text{MINCSP}(p, o)$ are $W[1]$ -hard.*

7 Discussion

We have initiated a study of the parameterized complexity of MINCSP for Allen’s interval algebra. We prove that MINCSP restricted to the relations in \mathcal{A} exhibits a dichotomy: $\text{MINCSP}(\Gamma)$ is either fixed-parameter tractable or $W[1]$ -hard when $\Gamma \subseteq \mathcal{A}$. Even though the restriction to the relations in \mathcal{A} may seem severe, one should keep in mind that a CSP instance over \mathcal{A} is sufficient for representing *definite* information about the relative positions of intervals. In other words, such an instance can be viewed as a data set of interval information and the MINCSP problem can be viewed as a way of filtering out erroneous information (that may be the result of contradictory sources of information, noise in the measurements, human mistakes etc.) Various ways of “repairing” unsatisfiable data sets of qualitative information have been thoroughly discussed by many authors; see, for instance, [7, 18, 19] and the references therein.

Proving a full parameterized complexity classification for Allen’s interval algebra is hindered by a barrier: such a classification would settle the parameterized complexity of DIRECTED SYMMETRIC MULTICUT, and this problem is considered to be one of the main open problems in the area of directed graph separation problems [27, 39]. This barrier comes into play even in very restricted cases: as an example, it is not difficult to see that MINCSP for the two Allen relations $(f \cup fi)$ and $(f \cup \equiv)$ is equivalent to the MINCSP problem for the two PA relations \neq and \leq and thus equivalent to DIRECTED SYMMETRIC MULTICUT.

One way of continuing this work without necessarily settling the parameterized complexity of DIRECTED SYMMETRIC MULTICUT is to consider fpt approximability: it is known that DIRECTED SYMMETRIC MULTICUT is 2-approximable in fpt time [27]. Thus, a possible research direction is to analyse the fpt approximability for MINCSP(Γ) when Γ is a subset of $2^{\mathcal{A}}$ or, more ambitiously, when Γ is first-order definable in \mathcal{A} . A classification that separates the cases that are constant-factor fpt approximable from those that are not may very well be easier to obtain than mapping the FPT/W[1] borderline. There is at least one technical reason for optimism here, and we introduce some definitions to outline this idea. An n -ary relation R is said to have a *primitive positive definition* (pp-definition) in a structure Γ if it can be first-order defined by only using the relations in Γ together with the equality relation and the operators existential quantification and conjunction. If the equality relation is not needed, then we say that R has an *equality-free primitive positive definition* (efpp-definition) in Γ . Bonnet et al. [13, Lemma 10] have shown that constant-factor fpt approximability is preserved by efpp-definitions [13], i.e. if R is efpp-definable in Γ and MINCSP(Γ) is constant-factor fpt approximable, then MINCSP($\Gamma \cup \{R\}$) is also constant-factor fpt approximable. Bonnet et al. focus on Boolean domains, but it is clear that their Lemma 10 works for problems with arbitrarily large domains. Lagerkvist [43, Lemma 5] has shown that in most cases one can use pp-definitions instead of efpp-definitions. This implies that the standard algebraic approach via polymorphisms (that, for instance, underlies the full complexity classification of finite-domain CSPs [14, 54]) often becomes applicable when analysing constant-factor fpt approximability. One should note that, on the other hand, the exact complexity of MINCSP is only preserved by much more limited constructions such as *proportional implementations* (see Section 5.2. in [35]). We know from the literature that this may be an important difference: it took several years after Bonnet et al.’s classification of approximability before the full classification of exact parameterized complexity was obtained using a much more complex framework [37]. It is also worth noting that parameterized approximation results for MINCSP may have very interesting consequences, e.g. [13] resolved the parameterized complexity of EVEN SET, which was a long-standing open problem.

References

- 1 Akanksha Agrawal, Daniel Lokshtanov, Amer E. Mouawad, and Saket Saurabh. Simultaneous feedback vertex set: A parameterized perspective. *ACM Transactions on Computation Theory*, 10(4):1–25, 2018.
- 2 Akanksha Agrawal, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Simultaneous feedback edge set: a parameterized perspective. *Algorithmica*, 83(2):753–774, 2021.
- 3 James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
- 4 James F. Allen and Johannes A. G. M. Koomen. Planning using a temporal world model. In *Proc. 8th International Joint Conference on Artificial Intelligence (IJCAI-1983)*, pages 741–747, 1983.

- 5 Kristóf Bérczi, Alexander Göke, Lydia Mirabel Mendoza Cadena, and Matthias Mnich. Resolving infeasibility of linear systems: A parameterized approach. *CoRR*, abs/2209.02017, 2022.
- 6 Benjamin Bergougnoux, Eduard Eiben, Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Towards a polynomial kernel for directed feedback vertex set. *Algorithmica*, 83(5):1201–1221, 2021.
- 7 Leopoldo Bertossi and Jan Chomicki. Query answering in inconsistent databases. In *Logics for Emerging Applications of Databases*, pages 43–83. Springer, 2004.
- 8 Manuel Bodirsky. *Complexity of Infinite-Domain Constraint Satisfaction*. Cambridge University Press, 2021.
- 9 Manuel Bodirsky and Martin Grohe. Non-dichotomies in constraint satisfaction complexity. In *Proc. 35th International Colloquium on Automata, Languages and Programming (ICALP-2008)*, pages 184–196, 2008.
- 10 Manuel Bodirsky, Peter Jonsson, Barnaby Martin, Antoine Mottet, and Zaneta Semanisinová. Complexity classification transfer for CSPs via algebraic products. *CoRR*, abs/2211.03340, 2022.
- 11 Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *Journal of the ACM*, 57(2):9:1–9:41, 2010.
- 12 Marthe Bonamy, Łukasz Kowalik, Jesper Nederlof, Michał Pilipczuk, Arkadiusz Socała, and Marcin Wrochna. On directed feedback vertex set parameterized by treewidth. In *Proc. 44th International Workshop on Graph-Theoretic Concepts in Computer Science (WG-2018)*, volume 11159, pages 65–78, 2018.
- 13 Édouard Bonnet, László Egri, and Dániel Marx. Fixed-parameter approximability of Boolean MinCSPs. In *Proc. 24th Annual European Symposium on Algorithms (ESA-2016)*, pages 18:1–18:18, 2016.
- 14 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *Proc. 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS-2017)*, pages 319–330, 2017.
- 15 Andrei A. Bulatov, Peter Jeavons, and Andrei A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005.
- 16 Jianer Chen, Yang Liu, Songjian Lu, Barry O’Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. In *Proc. 40th Annual ACM Symposium on Theory of Computing (STOC-2008)*, pages 177–186, 2008.
- 17 Rajesh Chitnis, Marek Cygan, Mohammataghi Hajiaghayi, and Dániel Marx. Directed subset feedback vertex set is fixed-parameter tractable. *ACM Transactions on Algorithms*, 11(4):1–28, 2015.
- 18 Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1-2):90–121, 2005.
- 19 Jean-François Condotta, Issam Nouaouri, and Michael Sioutis. A SAT approach for maximizing satisfiability in qualitative spatial and temporal constraint networks. In *Proc. 15th International Conference on the Principles of Knowledge Representation and Reasoning (KR-2016)*, 2016.
- 20 Robert Crowston, Gregory Gutin, Mark Jones, and Anders Yeo. Parameterized complexity of satisfying almost all linear equations over \mathbb{F}_2 . *Theory of Computing Systems*, 52(4):719–728, 2013.
- 21 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.
- 22 Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, and George Osipov. Resolving inconsistencies in simple temporal problems: A parameterized approach. In *Proc. 36th AAAI Conference on Artificial Intelligence, (AAAI-2022)*, pages 3724–3732, 2022.
- 23 Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, Marcin Pilipczuk, and Roohani Sharma. Parameterized complexity classification for interval constraints. *CoRR*, abs/2305.13889, 2023.

- 24 Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, and Magnus Wahlström. Almost consistent systems of linear equations. In *Proc. 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-2023)*, pages 3179–3217, 2023.
- 25 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.
- 26 Pascal Denis and Philippe Muller. Predicting globally-coherent temporal structures from texts via endpoint inference and graph decomposition. In *Proc. 22nd International Joint Conference on Artificial Intelligence (IJCAI-2011)*, pages 1788–1793, 2011.
- 27 Eduard Eiben, Clément Rambaud, and Magnus Wahlström. On the parameterized complexity of symmetric directed multicut. In *Proc. 17th International Symposium on Parameterized and Exact Computation (IPEC-2022)*, volume 249, pages 11:1–11:17, 2022.
- 28 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1998.
- 29 Alexander Göke, Dániel Marx, and Matthias Mnich. Hitting long directed cycles is fixed-parameter tractable. In *Proc. 47th International Colloquium on Automata, Languages, and Programming (ICALP-2020)*, volume 168, pages 59:1–59:18, 2020.
- 30 Alexander Göke, Dániel Marx, and Matthias Mnich. Parameterized algorithms for generalizations of directed feedback vertex set. *Discrete Optimization*, 46:100740, 2022.
- 31 Martin Charles Golumbic and Ron Shamir. Complexity and algorithms for reasoning about time: A graph-theoretic approach. *Journal of the ACM*, 40(5):1108–1133, 1993.
- 32 Venkatesan Guruswami, Rajsekar Manokaran, and Prasad Raghavendra. Beating the random ordering is hard: inapproximability of maximum acyclic subgraph. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS-2008)*, pages 573–582, 2008.
- 33 Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David P. Williamson. The approximability of constraint satisfaction problems. *SIAM Journal on Computing*, 30(6):1863–1920, 2000.
- 34 Subhash Khot. On the power of unique 2-prover 1-round games. In *Proc. 24th Annual ACM Symposium on Theory of Computing (STOC-2002)*, pages 767–775, 2002.
- 35 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Solving hard cut problems via flow-augmentation. In *Proc. 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA-2021)*, pages 149–168, 2021.
- 36 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Directed flow-augmentation. In *Proc. 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC-2022)*, pages 938–947, 2022.
- 37 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Flow-augmentation III: complexity dichotomy for Boolean CSPs parameterized by the number of unsatisfied constraints. In *Proc. 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-2023)*, pages 3218–3228, 2023.
- 38 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Flow-augmentation I: directed graphs. *CoRR*, abs/2111.03450, 2023.
- 39 Eun Jung Kim, Tomáš Masařík, Marcin Pilipczuk, Roohani Sharma, and Magnus Wahlström. On weighted graph separation problems and flow-augmentation. *CoRR*, abs/2208.14841, 2022.
- 40 Vladimir Kolmogorov, Andrei A. Krokhin, and Michal Rolínek. The complexity of general-valued CSPs. *SIAM Journal on Computing*, 46(3):1087–1110, 2017.
- 41 Andrei A. Krokhin, Peter Jeavons, and Peter Jonsson. Reasoning about temporal relations: The tractable subalgebras of Allen’s interval algebra. *Journal of the ACM*, 50(5):591–640, 2003.
- 42 Andrei A. Krokhin and Jakub Oprsal. An invitation to the promise constraint satisfaction problem. *ACM SIGLOG News*, 9(3):30–59, 2022.
- 43 Victor Lagerkvist. A new characterization of restriction-closed hyperclones. In *Proc. 50th IEEE International Symposium on Multiple-Valued Logic (ISMVL-2020)*, pages 303–308, 2020.

- 44 Daniel Lokshtanov, Pranabendu Misra, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Fpt-approximation for fpt problems. In *Proc. 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA-2021)*, pages 199–218, 2021.
- 45 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. Linear time parameterized algorithms for subset feedback vertex set. *ACM Transactions on Algorithms*, 14(1):7:1–7:37, 2018.
- 46 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. When recursion is better than iteration: A linear-time algorithm for acyclicity with few error vertices. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-2018)*, pages 1916–1933, 2018.
- 47 Dániel Marx and Igor Razgon. Constant ratio fixed-parameter approximation of the edge multicut problem. *Information Processing Letters*, 109(20):1161–1166, 2009.
- 48 Lenka Mudrová and Nick Hawes. Task scheduling for mobile robots using interval algebra. In *Proc. 2015 IEEE International Conference on Robotics and Automation (ICRA-2015)*, pages 383–388, 2015.
- 49 Richard N. Pelavin and James F. Allen. A model for concurrent actions having temporal extent. In *Proc. 6th National Conference on Artificial Intelligence (AAAI-1987)*, pages 246–250, 1987.
- 50 Prasad Raghavendra. Optimal algorithms and inapproximability results for every CSP? In *Proc. 40th Annual ACM Symposium on Theory of Computing (STOC-2008)*, pages 245–254, 2008.
- 51 Igor Razgon and Barry O’Sullivan. Almost 2-SAT is fixed-parameter tractable. *Journal of Computer and System Sciences*, 75(8):435–450, 2009.
- 52 Fei Song and Robin Cohen. The interpretation of temporal relations in narrative. In *Proc. 7th National Conference on Artificial Intelligence (AAAI-1988)*, pages 745–750, 1988.
- 53 Marc B. Vilain and Henry A. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proc. 5th National Conference on Artificial Intelligence (AAAI-1986)*, pages 377–382, 1986.
- 54 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *Journal of the ACM*, 67(5):30:1–30:78, 2020.

An FPT Algorithm for Temporal Graph Untangling

Riccardo Dondi  

Università degli studi di Bergamo, Italy

Manuel Lafond  

Université de Sherbrooke, Canada

Abstract

Several classical combinatorial problems have been considered and analysed on temporal graphs. Recently, a variant of VERTEX COVER on temporal graphs, called MINTIMELINECOVER, has been introduced to summarize timeline activities in social networks. The problem asks to cover every temporal edge while minimizing the total span of the vertices (where the span of a vertex is the length of the timestamp interval it must remain active in). While the problem has been shown to be NP-hard even in very restricted cases, its parameterized complexity has not been fully understood. The problem is known to be in FPT under the span parameter only for graphs with two timestamps, but the parameterized complexity for the general case is open. We settle this open problem by giving an FPT algorithm that is based on a combination of iterative compression and a reduction to the DIGRAPH PAIR CUT problem, a powerful problem that has received significant attention recently.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph theory; Theory of computation → Design and analysis of algorithms

Keywords and phrases Temporal Graphs, Vertex Cover, Graph Algorithms, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.12

Acknowledgements We thank the referees of the paper that help us to improve the presentation.

1 Introduction

Temporal graphs are emerging as one of the main models to describe the dynamics of complex networks. They describe how relations (edges) change in a discrete time domain [12, 11], while the vertex set is not changing. The development of algorithms on temporal graphs has mostly focused on finding paths or walks and on analyzing graph connectivity [12, 20, 21, 7, 22, 8, 3, 17, 1, 5]. However, several classical problems in computer science have been recently extended to temporal graphs and one of the most relevant problems in graph theory and theoretical computer science, VERTEX COVER, has been considered in this context [2, 10, 19].

In particular, here we study a variant of VERTEX COVER, called NETWORK UNTANGLING, introduced in [19]. NETWORK UNTANGLING has applications in discovering event timelines and summarizing temporal networks. It considers a sequence of temporal interactions between entities (e.g. discussions between users in a social network) and aims to explain the observed interactions with few (and short) *activity intervals* of entities, such that each interaction is covered by at least one of the two entities involved (i.e. at least one of the two entities is active when an interaction between them is observed).

NETWORK UNTANGLING can be seen as a variant of VERTEX COVER, where we search for a minimum cover of the interactions, called temporal edges. The size of this temporal vertex cover is based on the definition of *span* of a vertex, that is the length of vertex activity. In particular, the span of a vertex is defined as the difference between the maximum and minimum timestamp where the vertex is active. Hence, if a vertex is active in exactly one



© Riccardo Dondi and Manuel Lafond;

licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 12; pp. 12:1–12:16

Leibniz International Proceedings in Informatics



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

timestamp, it has a span equal to 0. This models the idea that each vertex is present in the network because we know that they interacted at least once, but that sustained periods of interaction are relatively rare.

Four combinatorial formulations of NETWORK UNTANGLING have been defined in [19], varying the definition of vertex activity (a single interval or $h \geq 2$ intervals) and the objective function (minimization of the sum of vertex spans or minimization of the maximum vertex span). Here we consider the formulation, denoted by MINTIMELINECOVER, where vertex activity is defined as a single interval and the objective function is the minimization of the sum of vertex spans. Hence, given a temporal graph, MINTIMELINECOVER asks for a cover of the temporal edges that has minimum span and such that each vertex is active in one time interval.

We focus on this specific problem, since it is not known to be FPT or not, while the variant of the problem where vertex activity is defined as two intervals is known to be NP-hard when the span is equal to 0 [9]. Hence it is unlikely that this problem variant admits an FPT algorithm for parameter the span. The MINTIMELINECOVER problem is known to be NP-hard also in very restricted cases, when each timestamp contains at most one temporal edge [4], when each vertex has at most two incident temporal edges in each timestamp and the temporal graph is defined over three timestamps [4], and when the temporal graph is defined over two timestamps [9]. MINTIMELINECOVER is also known to be approximable within factor $O(T \log n)$, where n is the number of vertices and T is the number of timestamps of the temporal graph [6]. Note that, since the span of a vertex activity in exactly one timestamp is equal to 0, MINTIMELINECOVER is trivially in P when the temporal graph is defined on a single timestamp, since in this case any solution of the problem has span 0. Furthermore, deciding whether there exists a solution of MINTIMELINECOVER that has span equal to 0 can be decided in polynomial time via a reduction to 2-SAT [19].

MINTIMELINECOVER has been considered also in the parameterized complexity framework. The definition of span leads to a problem where the algorithmic approaches applied to VERTEX COVER cannot be easily extended for the parameter span of the solution. Indeed, in VERTEX COVER for each edge we are sure that at least one of the endpoints must be included in the solution, thus at least one of the vertices contributes to the cost of the solution. This leads to the textbook FPT algorithm of branching over the endpoints of any edge. For MINTIMELINECOVER, a vertex with span 0 may cover a temporal edge, as the vertex can be active only in the timestamp where the temporal edge is defined. This makes it more challenging to design FPT algorithms when the parameter is the span of the solution. In this case, MINTIMELINECOVER is known to admit a parameterized algorithm only when the input temporal graph is defined over two timestamps [9], with a parameterized reduction to the ALMOST 2-SAT problem. However, the parameterized complexity of MINTIMELINECOVER for the span parameter on general instances has been left open [9, 4]. The authors of [9] have also analyzed the parameterized complexity of the variants of NETWORK UNTANGLING proposed in [19], considering other parameters in addition to the span of the solution: the number of vertices of the temporal graph, the length of the time domain, and the number of intervals of vertex activity.

Our contributions. We solve the open question on the parameterized complexity of MINTIMELINECOVER by showing that the problem is FPT in parameter k , the span of a solution, even if the number of timestamps is unbounded. Our algorithm takes time $O^*(2^{5k \log k})$, where the O^* notation hides polynomial factors. Our algorithm is divided into two phases, each using a different technique. First, given a temporal graph G , we use a variant of

iterative compression, where we start from a solution S of span at most k on a subgraph of G induced by a subset of vertices (taken across all timestamps), and then try to maintain such a solution after adding a new vertex of G to the graph under consideration. This requires us to reorganize which vertices involved in S should be in the solution or not, and in which timestamps. One challenge is that since the number of such timestamps is unbounded, there are too many ways to choose how to include or not include the vertices that are involved in S . We introduce the notion of a *feasible assignment*, which allows us to compute how the vertices in S can be reorganized (see Def. 8 for the formal definition). There are only $2^{O(k \log k)}$ ways of reorganizing the vertices in S . We try each such feasible assignments X , and we must then find a temporal cover of the whole graph G that “agrees” with X .

This leads to the second phase of the algorithm, which decides if such an agreement cover exists through a reduction to a variant of a problem called DIGRAPH PAIR CUT. In this problem, we receive a directed graph and forbidden pairs of vertices, and we must delete at most k arcs so that a specified source vertex does not reach both vertices from a forbidden pair. It is known that the problem can be solved in time $O^*(2^k)$. In this work, we need a version where the input specifies a set of deletable and undeletable arcs, which we call CONSTRAINED DIGRAPH PAIR CUT. The DIGRAPH PAIR CUT problem and its variants have played an important role in devising randomized kernels using matroids [16] and, more recently, in establishing a dichotomy in the complexity landscape of constraint satisfaction problems [13, 15]. Here, the problem is useful since it can model the implications of including a vertex in the solution or not and, in a more challenging way, allows implementing the notion of cost using our definition of span. We hope that the techniques developed for this reduction can be useful for other variants of temporal graph cover.

Overview of the algorithm. Our approach is loosely inspired by some ideas from the FPT algorithm for two timestamps, which is a reduction to ALMOST 2-SAT [9]. In the latter, one is given a set of clauses with at most two variables each and must delete a minimum number of clauses so that those remaining are satisfiable. We do not use ALMOST 2-SAT directly, but its usage for two timestamps may help understand the origins of our techniques and the relevance of our reduction to DIGRAPH PAIR CUT.

The reduction from MINTIMELINECOVER on two timestamps to ALMOST 2-SAT associates each vertex v_i with a variable $x(v_i)$, which is true when one should include v_i in a temporal cover and false otherwise; each edge $u_i v_i$ is associated with a clause $x(u_i) \vee x(v_i)$ (here, v_i represents the occurrence of vertex v at timestamp $i \in \{1, 2\}$). This corresponds to enforcing the inclusion of u_i or v_i in our vertex cover, and we can include enough copies of this clause to make it undeletable. Since our goal is to minimize the number of base vertices v with both v_1 and v_2 in the cover, we also add a clause $\neg x(v_1) \vee \neg x(v_2)$. Then there is a temporal cover of G of span at most k if and only if one can delete at most k clauses of the latter form to make all remaining clauses satisfiable.

For $T \geq 3$ timestamps, the clauses of the form $x(u_i) \vee x(v_i)$ can still be used to model the vertex cover requirements, but there seems to be no obvious way to model the span of a cover. One would need to devise a set of clauses of size two such that choosing an interval of t vertices in a cover corresponds to deleting $t - 1$ negative clauses. Our idea is to extend current FPT algorithms for ALMOST 2-SAT to accommodate our cost function. In [18], the authors propose an iterative compression FPT algorithm that starts from a solution that deletes $k + 1$ clauses, and modifies it into a solution with k clauses, if possible. The algorithm relies on several clever, but complicated properties of the dependency graph of the clauses (in which vertices are literals and arcs are implications implied by the clauses). This

algorithm seems difficult to adapt to our problem. To our knowledge, the only other FPT algorithm for ALMOST 2-SAT is that of [16]. The algorithm of [16] employs a parameterized reduction to DIGRAPH PAIR CUT. At a high level, the idea is to start from an initial guess of assignment for a well-chosen subset of variables, then to construct the dependency graph of the clauses. A certain chain of implications is enforced by our initial guess, the vertex pairs to separate correspond to contradictory literals, and deleting arcs corresponds to deleting clauses. It turns out that, with some work, we can skip the ALMOST 2-SAT formulation and reduce MINTIMELINECOVER to (a variant of) DIRECTED PAIR CUT directly by borrowing some ideas from this reduction. This is not immediate though. The first challenge is that the aforementioned “well-chosen initial guess” idea cannot be used in our context, and we must develop new tools to enumerate a bounded number of initial guesses from a partial solution (which we call feasible assignment). The second challenge is that our reduction to our variant of DIRECTED PAIR CUT needs a specific gadget to enforce our cost scheme, while remaining consistent with the idea of modeling the dependency graph of the SAT instance corresponding to the vertex cover problem at hand.

Some of the proofs are omitted due to page limit.

2 Preliminaries

For an integer n , we denote $[n] = \{1, \dots, n\}$ and for two integers i, j , we denote $[i, j] = \{i, i+1, \dots, j-1, j\}$ (which is the empty set if $i > j$). Temporal graphs are defined over a discrete time domain \mathcal{T} , which is a sequence $1, 2, \dots, T$ of timestamps. A temporal graph is also defined over a set of vertices, called *base vertices*, that do not change in the time domain and are defined in all timestamps, and are associated with *vertices*, which are base vertices defined in specific timestamps. We use subscripts to denote the timestamp to which a vertex belongs to, so, for a base vertex v and $t \in [T]$, we use v_t to denote the occurrence of v in timestamp t . A *temporal edge* connects two vertices, associated with distinct base vertices, that belong to the same timestamp.

► **Definition 1.** A temporal graph $G = (V_B, E, \mathcal{T})$ consists of

1. A time domain $\mathcal{T} = \{1, 2, \dots, T\}$;
2. A set V_B of base vertices; V_B has a corresponding set $V(G)$ of vertices, which consists of base vertices in specific timestamps, defined as follows:

$$V(G) = \{v_t : v \in V_B \wedge t \in [T]\}.$$

3. A set $E = E(G)$ of temporal edges, which satisfies:

$$E \subseteq \{u_t v_t : u, v \in V_B, t \in [T] \wedge u \neq v\}.$$

For a directed (static) graph H , we denote by (u, v) an arc from vertex u to vertex v (we consider only directed static graphs, not directed temporal graphs).

Given a temporal graph $G = (V_B, E, \mathcal{T})$ and a set of base vertices $B \subseteq V_B$, we define the set $\tau(B)$ of all vertices of B across all times:

$$\tau(B) = \{v_t : v \in B \wedge t \in [T]\}.$$

If $B = \{v\}$, we may write $\tau(v)$ instead of $\tau(\{v\})$.

For a subset $W_B \subseteq V_B$ of base vertices, we denote by $G[W_B]$ the subgraph induced by $\tau(W_B)$, that is, the graph whose vertex set is $\tau(W_B)$ and whose edge set is $\{u_t v_t \in E : u_t, v_t \in \tau(W_B)\}$. We also use the notation $G - W_B = G[V_B \setminus W_B]$. Observe that $G[W_B]$ and $G - W_B$ are temporal graphs over the same time domain as G .

In order to define the problem we are interested in, we need to define the *assignment* of a set of base vertices.

► **Definition 2.** Consider a temporal graph $G = (V_B, E, \mathcal{T})$ and a set $W_B \subseteq V_B$ of base vertices. An assignment of W_B is a subset $X \subseteq \tau(W_B)$ such that if $u_p \in X$ and $u_q \in X$, with $p, q \in [T]$, then $u_t \in X$, for each $t \in [p, q]$. For a base vertex $u \in W_B$ such that there exists $t \in [T]$ with $u_t \in X$, we denote by $\delta(u, X)$, $\Delta(u, X)$, respectively, the minimum and maximum timestamp, respectively, such that $u_{\delta(u, X)}, u_{\Delta(u, X)} \in X$. If u_t does not exist, then $\delta(u, X) = \Delta(u, X) = 0$.

If W_B is clear from the context or not relevant, then we may say that X is an assignment, without specifying W_B . Note that, given an assignment X and a set $\tau(v)$, for some $v \in V_B$, then $X \cap \tau(v) = \{v_t : v_t \in X \wedge v_t \in \tau(v)\}$ contains vertices for v that belong to a contiguous interval of timestamps. Consider a set $I \subseteq [T]$ of timestamps. An assignment X intersects I if there exists $v_t \in X$ such that $t \in I$.

Now, we give the definition of *temporal cover*.

► **Definition 3.** Given a temporal graph $G = (V_B, E, \mathcal{T})$ a temporal cover of G is an assignment X of V_B such that the following properties hold:

1. For each $v \in V_B$ there exists at least one $v_t \in X$, for some $t \in \mathcal{T}$.
2. For each $u_t v_t \in E$, with $t \in [T]$, at least one of u_t, v_t is in X .

For a temporal cover X of G , the *span* of v in X is defined as: $sp(v, X) = \Delta(v, X) - \delta(v, X)$. Note that if a temporal cover X contains, for a base vertex $v \in V_B$, a single vertex v_t , then $sp(v, X) = 0$. The span of X , denoted by $sp(X)$, is then defined as:

$$sp(X) = \sum_{v \in V_B} sp(v, X).$$

The definition of temporal cover requires that for each base vertex at least one of its associated vertices belongs to the cover. This is not strictly necessary, since it might be possible to cover every temporal edge without this condition. However, this condition simplifies some of the definitions and proofs below. Note that if an assignment of a base vertex is not needed to cover temporal edges, we can assign the vertex to some timestamp without increasing the span.

Now, we are able to define `MINTIMELINECOVER` (an example is presented in Fig. 1).

► **Problem 4.** (`MINTIMELINECOVER`)

Input: A temporal graph $G = (V_B, \mathcal{T}, E)$, an integer k .

Question: Does there exist a temporal cover of G of span at most k ?

A temporal cover $S \subseteq V(G)$ of span at most k will sometimes be called a *solution*. Our goal is to determine whether `MINTIMELINECOVER` is FPT in parameter k .

3 An FPT Algorithm

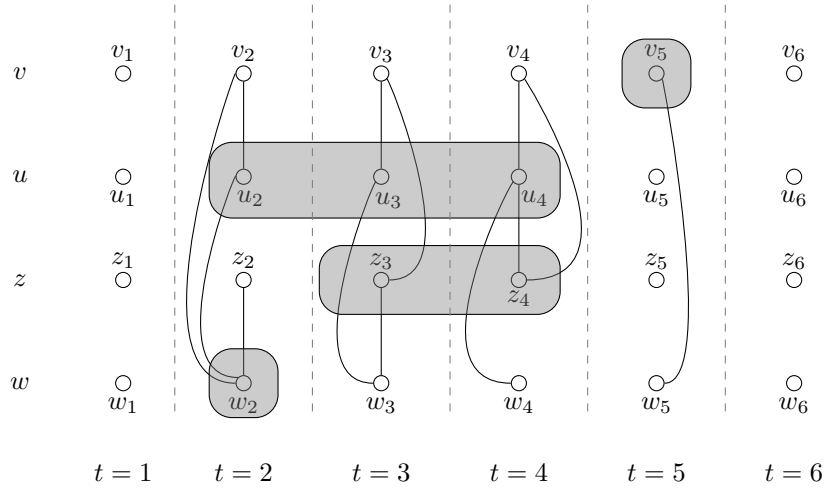
In this section we present our FPT algorithm, which consists of two parts:

1. The iterative compression technique.
2. A reduction to the `CONSTRAINED DIGRAPH PAIR CUT` problem.

Before presenting the main steps of our algorithm, we present the main idea and some definitions. Recall that our parameter, that is the span of a solution of `MINTIMELINECOVER`, is denoted by k .

Consider a temporal graph G and assume we have a temporal cover S of span at most k of the subgraph $G - \{w\}$, for some base vertex $w \in V_B$. The idea of the iterative compression step is, starting from S , to show how to decide in FPT time whether there

12:6 An FPT Algorithm for Temporal Graph Untangling



■ **Figure 1** An example of MINTIMELINECOVER on a temporal graph G consisting of four base vertices and six timestamps. For each timestamp, we draw the temporal edges of G , for example for $t = 2$, the temporal edges are $v_2u_2, v_2w_2, u_2w_2, z_2w_2$. Also note that in $t = 1$ and $t = 6$ no temporal edge is defined. A temporal cover $X = \{v_5, u_2, u_3, u_4, z_3, z_4, w_2\}$ is represented with grey rectangles. Note that $\delta(v, X) = \Delta(v, X) = 5$, $\delta(u, X) = 2$, $\Delta(u, X) = 4$, $\delta(z, X) = 3$, $\Delta(z, X) = 4$, $\delta(w, X) = \Delta(w, X) = 2$. It follows that $sp(X) = 3$.

exists a solution of MINTIMELINECOVER for G . This is done by solving a subproblem, called $\text{RESTRICTED TIMELINE COVER}$, where we must modify S to consider w . A solution to this subproblem is computed by branching on the assignments of base vertices having a positive span in S and on w , and then reducing the problem to $\text{CONSTRAINED DIGRAPH PAIR CUT}$. $\text{RESTRICTED TIMELINE COVER}$ is defined as follows.

► **Problem 5.** (*RESTRICTED TIMELINE COVER*)

Input: A temporal graph $G = (V_B, E, \mathcal{T})$, a vertex $w \in V_B$, an integer k , a temporal cover S of $G - \{w\}$ of span at most k .

Output: Does there exist a temporal cover of G of span at most k ?

For technical reasons that will become apparent later, we will assume that the temporal graph contains no edge at timestamps 1 and T , i.e. for every $u_tv_t \in E$, we have $t \in [2, T - 1]$ (as in Fig. 1). In particular, this avoids us to consider different gadget definitions in the reduction to $\text{CONSTRAINED DIGRAPH PAIR CUT}$, as the cases where a base vertex is assigned the first or the last of its associated vertex behaves somehow differently. It is easy to see that if this is not already the case, we can add two such “dummy” timestamps, where G does not contain any temporal edge. Indeed, since there are no temporal edges in these two timestamps, then G has a temporal cover of span at most k if and only if the same graph with dummy timestamps has a temporal cover of span at most k .

Informally, if we are able to solve $\text{RESTRICTED TIMELINE COVER}$ in FPT time, then we can obtain an FPT algorithm for MINTIMELINECOVER as well. Indeed, we can first compute a temporal cover on a small subset of base vertices (for example a single vertex), and then we can add, one at a time, the other vertices of the graph. This requires at most $|V_B|$ iterations, and each time a vertex is added, we compute a solution of $\text{RESTRICTED TIMELINE COVER}$ to check whether it is possible to find a temporal cover of span at most k after the addition of a vertex.

Iterative Compression

We now present our approach based on iterative compression to solve the RESTRICTED TIMELINE COVER problem. Given a solution S for $G - \{w\}$, we focus on the vertices of V_B that have a positive span in S and vertex w . An example of our approach, that illustrates the sets of base vertices and vertices used by the algorithm, is presented in Fig. 2.

Consider the input of RESTRICTED TIMELINE COVER that consists of a temporal graph $G = (V_B, E, \mathcal{T})$, a vertex $w \in V_B$, and a temporal cover S of $G - \{w\}$ of span at most k . Define the following sets associated with S :

$$V_S = \{v \in V_B : \exists p, q \in [T], p < q, \text{ such that } v_p, v_q \in S\} \cup \{w\}$$

$$V'_S = \{v_t : v_t \in S, v \in V_S \setminus \{w\}\} \cup \{w_t : t \in [T]\}.$$

The set V_S is defined as the set of base vertices having span greater than 0 in S , plus the vertex w . V'_S contains the vertices in $V(G)$ associated with V_S , in particular: (1) the vertices corresponding to the base vertices in $V_S \setminus \{w\}$ that are included in S and (2) vertices corresponding to the base vertex w in every timestamp.

Define the following set I_S of timestamps associated with $V_S \setminus \{w\}$:

$$I_S = \{t \in [T] : u_t \in V'_S \text{ for some } u \in V_S \setminus \{w\}\}.$$

Essentially, I_S contains those timestamps where the base vertices of $V_S \setminus \{w\}$, that is of span greater than zero, have associated vertices in S . These timestamps are essential for computing a solution of RESTRICTED TIMELINE COVER, that is to compute whether there exists a temporal cover of G of span at most k starting from S . We define now the sets of base vertices and vertices associated with S having a span equal to 0:

$$Z_S = V_B \setminus V_S \quad Z'_S = S \setminus V'_S.$$

First, we show two easy properties of S and I_S on the temporal graph $G - \{w\}$.

► **Lemma 6.** *Let S be a solution of MINTIMELINECOVER on instance $G - \{w\}$ and let I_S be the associated set of timestamps. Then $|I_S| \leq 2k$.*

► **Lemma 7.** *Let S be a solution of MINTIMELINECOVER on instance $G - \{w\}$. Then, $sp(Z'_S) = 0$. Moreover, Z'_S covers each temporal edge of $G - \{w\}$ not covered by $V'_S \setminus \tau(w)$.*

Now, we introduce the concept of feasible assignment, which is used to “guess” how S is rearranged in a solution of RESTRICTED TIMELINE COVER. Recall that an assignment X intersects a set I_S of timestamps if there exists $v_t \in X$ such that $t \in I_S$.

► **Definition 8 (Feasible assignment).** *Consider an instance of RESTRICTED TIMELINE COVER that consists of a temporal graph $G = (V_B, \mathcal{T}, E)$, a vertex $w \in V_B$, a temporal cover S of $G - \{w\}$ of span at most k , and sets V_S, V'_S and I_S associated with S . We say that an assignment $X \subseteq \tau(V_S)$ of V_S is a feasible assignment (with respect to G, S , and I_S) if all of the following conditions hold:*

1. *the span of X is at most k ;*
2. *every edge of $G[V_S]$ is covered by X ;*
3. *$X \cap \tau(w)$ is a non-empty assignment of $\{w\}$;*
4. *for every $v \in V_S \setminus \{w\}$, at least one of the following holds: (1) $X \cap \tau(v)$ is empty; (2) $X \cap \tau(v)$ is an assignment of $\{v\}$ that intersects with I_S ; or (3) $X \cap \tau(v)$ contains a vertex v_t such that $v_t w_t \in E$ and $w_t \notin X \cap \tau(w)$.*

12:8 An FPT Algorithm for Temporal Graph Untangling

Given a feasible assignment X , we denote

$$M_S(X) = \{v \in V_S : X \cap \tau(v) \neq \emptyset\} \quad N_S(X) = \{v \in V_S : X \cap \tau(v) = \emptyset\}$$

Informally, point 4 considers the possible cases for a feasible assignment of the vertices of a base vertex $v \in V_S \setminus \{w\}$: none of the associated vertices in I_S belongs to the computed solution (case 4.(1)); some of its associated vertices in I_S belongs to the solution (case 4.(2)); or some of the v_t vertices are forced, since they belong to an edge $v_t w_t$ with $t \in I_S$, that we know is not covered by w_t (case 4.(3)). Note that these cases are not necessarily mutually exclusive.

Note that $M_S(X)$ and $N_S(X)$ form a partition of V_S . Also note that G, S , and I_S are fixed in the remainder, so we assume that all feasible assignments are with respect to G, S , and I_S without explicit mention. We now relate feasible assignments to temporal covers.

► **Definition 9.** Let X^* be a temporal cover of G and let X be a feasible assignment. We say that X^* agrees with X if:

- for each $v \in M_S(X)$, $X^* \cap \tau(v) = X \cap \tau(v)$;
- for each $v \in N_S(X)$ and each $t \in I_S$, X^* contains every neighbor u_t of v_t such that $u_t \in \tau(Z_S)$.

The intuition of X^* agreeing with X is as follows. For $v \in M_S(X)$, X “knows” which vertices of $\tau(v)$ should be in the solution, and we require X^* to contain exactly those. For $v \in N_S(X)$, we interpret that X does not want any vertex v_t with $t \in I_S$. Thus, to cover the edges incident to v_t that go outside of V_S , we require X^* to contain the other endpoint. Note an important subtlety: we act “as if” X^* should not contain v_t or other vertices of $N_S(X)$ with timestamp in I_S , but the definition does not forbid it. Hence, X^* can contain a vertex of $N_S(X)$ in some timestamps of I_S , as long as X^* contains also its neighbors (in I_S) outside V_S .

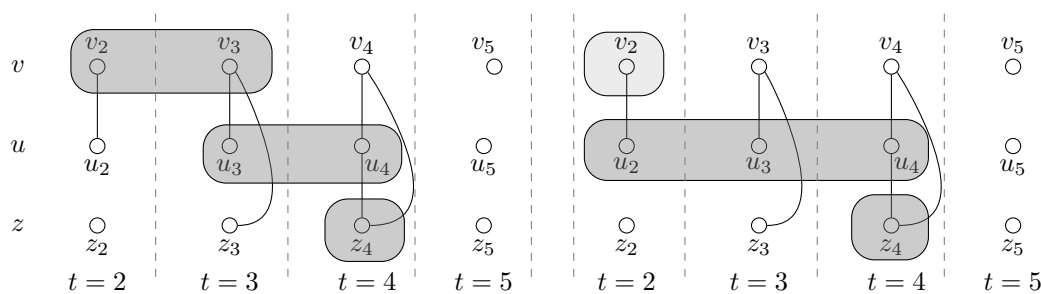
The main purpose of feasible assignments and agreement is as follows.

► **Lemma 10.** Let X^* be a temporal cover of G of span at most k . Then there exists a feasible assignment X such that X^* agrees with X .

Proof. Construct $X \subseteq X^*$ as follows: add $X^* \cap \tau(w)$ to X , and for $v \in V_S \setminus \{w\}$, add $X^* \cap \tau(v)$ to X if and only if $X^* \cap \tau(v)$ intersects with the set I_S , or if it contains a vertex v_t incident to an edge $v_t w_t \in E$ such that $w_t \notin X^* \cap \tau(w)$. Note that since X^* is an assignment of V_B , X is an assignment of V_S .

We first focus on arguing that X satisfies each condition of a feasible assignment (Definition 8). For Condition 1, since X^* has span at most k and $X \subseteq X^*$, it is clear that X also has span at most k . For Condition 3, $X^* \cap \tau(w)$ is non-empty by the definition of a temporal cover, and we added $X^* \cap \tau(w)$ to X . For Condition 4, we explicitly require in our construction of X that for each $v \in V_S \setminus \{w\}$, if $X \cap \tau(v)$ is non-empty, then it is equal to $X^* \cap \tau(v)$ and it either intersects with I_S or covers an edge not covered by $X \cap \tau(w) = X^* \cap \tau(w)$.

Let us focus on Condition 2. Let $u_t v_t \in E(G[V_S])$. If $u = w$, then if we did not add w_t to X , X^* must contain v_t and we added $X^* \cap \tau(v)$ to X , thereby covering the edge. The same holds if $v = w$. Assume $u \neq w, v \neq w$, and suppose without loss of generality that X^* contains u_t to cover the edge. Suppose for contradiction that X does not cover $u_t v_t$. Then we did not add $X^* \cap \tau(u)$ to X , which implies that $X^* \cap \tau(u)$ does not intersect with I_S . In particular, $t \notin I_S$. Recall that S , the temporal cover of $G - \{w\}$, only intersects with $\tau(u)$ and $\tau(v)$ in timestamps contained in I_S . Hence, S cannot cover $u_t v_t$, a contradiction. We deduce that X covers every edge. Therefore, X is a feasible assignment.



■ **Figure 2** An example of application of iterative compression (timestamps 1 and 6 are not shown as they are edgeless, also vertex w is not shown, its assignment is defined as in Fig. 1). In the left part, we represent solution $S = \{v_2, v_3, u_3, u_4, z_4\}$, where the vertices in S are highlighted with grey rectangles. Note that $I_S = \{2, 3, 4\}$, $V_S = \{v, u\}$, $V'_S = \{v_2, v_3, u_3, u_4\}$, $Z_S = \{z\}$, $Z'_S = \{z_4\}$. In the right part, we represent in grey a feasible assignment X associated with S , containing vertices u_2, u_3, u_4 ; in light grey we highlight $N'_S = \{v_2\}$. The sets associated with S and X are: $M_S = \{u\}$, $N_S = \{v\}$, $N'_S = \{v_2\}$, $N''_S = \{v_2, v_3, v_4\}$. The reduction to CONstrained DIGRAPH PAIR CUT eventually leads to the solution of MINTIMELINECOVER represented in Fig. 1.

It remains to show that X^* agrees with X . For $v \in M_S(X)$, $X^* \cap \tau(v) = X \cap \tau(v)$ by the construction of X . For $v \in N_S(X)$, there is no $v_t \in X^*$ with $t \in I_S$, as otherwise we would have added $X^* \cap \tau(v)$ to X . For every such v_t , X^* must contain all of its neighbors in $\tau(Z_S)$ to cover the edges, as required by the definition of agreement. ◀

It remains to show that the number of feasible assignments has bounded size and can be enumerated efficiently. We first show the latter can be achieved through the following steps. Start with X as an empty set and then apply the following steps (checking that the overall span is at most k):

- (1) Branch into every non-empty assignment X_w of $\{w\}$ of span at most k . In each branch, add the chosen subset X_w to X ;
- (2) For every edge $v_t w_t \in E(G[V_S])$ such that $w_t \notin X_w$, add v_t to X ;
- (3) For every $v \in V_S \setminus \{w\}$, such that $X \cap \tau(v) = \emptyset$ at this moment, branch into $|I_S| + 1$ options: either add no vertex of $\tau(v)$ to X , or choose a vertex v_t and add it to X , where $t \in I_S$;
- (4) For every $v \in V_S \setminus \{w\}$ such that $X \cap \tau(v) \neq \emptyset$ at this moment, branch into every assignment X_v of $\{v\}$ of span at most k that contains every vertex of $X \cap \tau(v)$ (if no such assignment exists, abort the current branch). For each such branch, add every vertex of $X_v \setminus X$ to X .

► **Theorem 11.** *The above steps enumerate every feasible assignment in time $O(2^{4k \log^k T^2 kn})$, where $n = |V_B|$.*

Reducing to Constrained Digraph Pair Cut

Our objective is now to list every feasible assignment and, for each of them, to verify whether there is a temporal cover that agrees with it. More specifically, consider a feasible assignment $X \subseteq \tau(V_S)$. Our goal is to decide whether there is a temporal cover X^* of span at most k that agrees with X . Since we branch over every possible feasible assignment X , if there is a temporal cover X^* of G of span at most k , then by Theorem 11 our enumeration will eventually consider an X that X^* agrees with, and hence we will be able to decide of the existence of X^* .

12:10 An FPT Algorithm for Temporal Graph Untangling

We show that finding X^* reduces to the CONSTRAINED DIGRAPH PAIR CUT problem, as we define it below. For a directed graph H , we denote its set of arcs by $A(H)$ (to avoid confusion with $E(G)$, which is used for the edges of an undirected graph G). For $F \subseteq A(H)$, we write $H - F$ for the directed graph with vertex set $V(H)$ and arc set $A(H) \setminus F$.

► **Problem 12.** (CONSTRAINED DIGRAPH PAIR CUT)

Input: A directed graph $H = (V(H), A(H))$, a source vertex $s \in V(H)$, a set of vertex pairs $P \subseteq \binom{V(H)}{2}$ called forbidden pairs, a subset of arcs $D \subseteq A(H)$ called deletable arcs, and an integer k' .

Output: Does there exist a set of arcs $F \subseteq D$ of H such that $|F| \leq k'$ and such that, for each $\{u, v\} \in P$, at least one of u, v is not reachable from s in $H - F$?

It is known that CONSTRAINED DIGRAPH PAIR CUT can be solved in time $O^*(2^{k'})$ [16], but a few remarks are needed before proceeding. In [16], the authors only provide an algorithm for the *vertex-deletion* variant, and do not consider deletable/undeletable arcs. It is easy to make an arc undeletable by adding enough parallel paths between the two endpoints, and we show at the end of the section that our formulation of CONSTRAINED DIGRAPH PAIR CUT reduces to the simple vertex-deletion variant. The vertex-deletion variant also admits a randomized polynomial kernel, and other FPT results are known for weighted arc-deletion variants [14].

So let us fix a feasible assignment X for the remainder of the section. We will denote $M_S = M_S(X)$ and $N_S = N_S(X)$. We also consider the following set of vertices associated with N_S :

$$N'_S = \{v_2 : v \in N_S\} \quad N''_S = \{v_t \in \tau(N_S) : t \in I_S\}.$$

For each base vertex $v \in N_S$, we need N'_S to contain any vertex of $\tau(v)$ that belongs to the time interval $[2, T - 1]$, so we choose v_2 arbitrarily. Then, N''_S contains those vertices v_t , with $t \in I_S$, not chosen by the feasible assignment X . Note that according to our definition of agreement, a solution X^* should contain all the neighbors of N''_S vertices that are in Z_S . Recall that we have defined $Z_S = V_B \setminus V_S$ and $Z'_S = S \setminus V'_S$. By Lemma 7 we know that Z'_S covers each temporal edge of $G[V_B \setminus \{w\}]$ not covered by $S \cap V'_S$, and that $sp(Z'_S) = 0$. We may assume that for each $v \in Z_S$, there is exactly one $t \in [T]$ such that $v_t \in Z'_S$ (there cannot be more than one since Z'_S has span 0, and if there is no such t , we can add any v_t without affecting the span). Furthermore, we will assume that for each $v \in Z_S$, the vertex v_t in Z'_S is not v_1 nor v_T . Indeed, since we assume that the first and last timestamps of G have no edges, if $v_t = v_1$ or $v_t = v_T$, then v_t covers no edge and we may safely change v_p to another vertex of $\tau(v)$.

The following observation will be useful for our reduction to CONSTRAINED DIGRAPH PAIR CUT.

► **Observation 13.** Let $u_t v_t \in E(G)$ such that $u \in N_S$ and $v \notin M_S$. Then $v \in Z_S$ and, if $u_t \notin N''_S$, we have $v_t \in Z'_S$.

Now, given a feasible assignment $X \subseteq \tau(V'_S)$, sets $M_S, N_S, N'_S, N''_S, Z_S$, and Z'_S , we present our reduction to the CONSTRAINED DIGRAPH PAIR CUT problem. We construct an instance of this problem that consists of the directed graph $H = (V(H), A(H))$, the set of forbidden (unordered) pairs $P \subseteq \binom{V(H)}{2}$, and the deletable arcs $D \subseteq A(H)$ by applying the following steps. The second step in the construction is the most important and is shown in Figure 3. The intuition of these steps is provided afterwards.

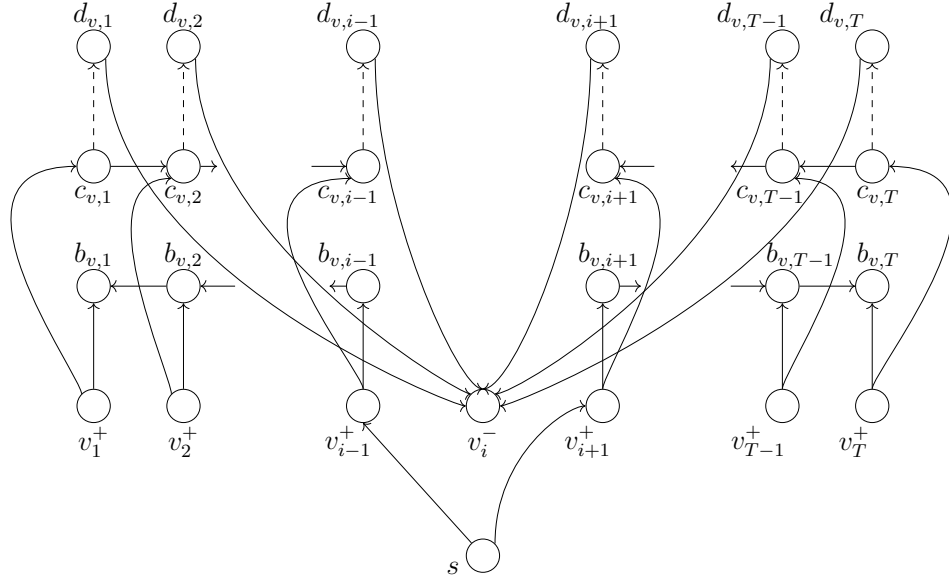
1. add to H the source vertex s ;
2. for each $v \in Z_S \cup N_S$, let v_i be the vertex of $Z'_S \cup N'_S$, where $i \in [2, T-1]$. Add to H the vertices $v_1^+, \dots, v_{i-1}^+, v_i^-, v_{i+1}^+, \dots, v_T^+$, the vertices $b_{v,j}, c_{v,j}, d_{v,j}$, for $j \in [T] \setminus \{i\}$, and the set of arcs shown in Figure 3, that is there are arcs $(v_j^+, b_{v,j}), (v_j^+, c_{v,j}), (c_{v,j}, d_{v,j}), (d_{v,j}, v_j^-)$, for each $j \in [T] \setminus \{i\}$ and four directed paths: (1) from $b_{v,i-1}$ to $b_{v,1}$, (2) from $c_{v,1}$ to $c_{v,i-1}$, (3) from $b_{v,i+1}$ to $b_{v,T}$ and (4) from $c_{v,T}$ to $c_{v,i+1}$. Add to D the set of deletable arcs $(c_{v,j}, d_{v,j})$, for $j \in [T] \setminus \{i\}$. Then add the following pairs to P :
 - a. $\{d_{v,h}, b_{v,j}\}$, with $1 \leq h < j \leq i-1$;
 - b. $\{d_{v,h}, b_{v,j}\}$, with $i+1 \leq j < h \leq T$;
 - c. $\{c_{v,h}, d_{v,j}\}$, with $1 \leq h \leq i-1 \leq i+1 \leq j \leq T$;
 - d. $\{c_{v,h}, d_{v,j}\}$, with $1 \leq j \leq i-1 \leq i+1 \leq h \leq T$.
 Note that we have created $T + 3(T-1) = 4T - 3$ vertices in H in this step. The subgraph of H induced by these vertices will be called the *gadget corresponding to v* .
3. for each temporal edge $u_t v_t \in E(G)$ such that $u_t, v_t \in \tau(Z_S) \cup (\tau(N_S) \setminus N''_S)$, there are three cases. First note that at least one of u_t or v_t is in Z'_S . Indeed, if $u, v \in Z_S$, this is because an element of Z'_S must cover the temporal edge, and if $u \in N_S$, then $v_t \in Z'_S$ by Observation 13 (or if $v \in N_S, u_t \in Z'_S$). The subcases are then:
 - a. if $u_t, v_t \in Z'_S \cup N'_S$, add the pair $\{u_t^-, v_t^-\}$ to P ;
 - b. if $u_t \in Z'_S \cup N'_S, v_t \notin Z'_S \cup N'_S$, add the arc (u_t^-, v_t^+) to H ;
 - c. if $v_t \in Z'_S \cup N'_S, u_t \notin Z'_S \cup N'_S$, add the arc (v_t^-, u_t^+) to H ;
4. for each temporal edge $u_t v_t \in E(G)$ such that $u_t \in (\tau(M_S) \setminus X) \cup N''_S$ and $v_t \in \tau(Z_S)$, there are two cases:
 - a. if $v_t \notin Z'_S$, add the arc (s, v_t^+) to H ;
 - b. if $v_t \in Z'_S$, add the pair $\{s, v_t^-\}$ to P .

Define $k' = k - sp(X)$. This concludes the construction. We will refer to the elements 1, 2, 3, 4 of the above enumeration as the *Steps* of the construction. Note that the only deletable arcs in D are the arcs $(c_{v,j}, d_{v,j})$ introduced in Step 2.

From here, the interpretation of H is that if we delete arc set F , then

- (p1) For $v_t \notin Z'_S \cup N'_S$ we should include v_t in X^* if and only if s reaches v_t^+ in $H - F$;
- (p2) For $v_t \in Z'_S \cup N'_S$ we should include v_t in X^* if and only if s does *not* reach v_t^- in $H - F$.

The idea behind the steps of the construction is then as follows (and is somewhat easier to describe in the reverse order of steps). Step 4 describes an initial set of vertices that s is forced to reach, which correspond to vertices that are forced in X^* . A vertex v_t in $\tau(Z_S)$ is forced in X^* if there is in an edge $u_t v_t$ and $u_t \in \tau(M_S)$ but $u_t \notin X$. By our definition of agreement, v_t is also forced if $u_t \in N''_S$. Step 4 handles both situations: if $v_t \notin Z'_S$, we force s to reach v_t^+ with the arc (s, v_t^+) , which is not deletable. If $v_t \in Z'_S$, then $v_t^- \in V(H)$, and s is forced to *not* reach v_t^- by adding $\{s, v_t^-\}$ to P . By (p1) and (p2), both cases correspond to including v_t in X^* . Then, Step 3 ensures that each temporal edge is “covered”: for a temporal edge $u_t v_t$, a pair of the form $\{u_t^-, v_t^-\}$ in P requires that s does not reach one of the two, i.e. that we include one in X^* , and an undeletable arc of the form (u_t^-, v_t^+) enforces that if s reaches u_t^- (i.e. $u_t \notin X^*$), then s reaches v_t^+ (i.e. $v_t \in X^*$). The reason why Z'_S is needed in our construction is that each edge has at least one negative corresponding vertex, so that no other case needs to be considered in Step 3.



■ **Figure 3** Gadget for $v_i \in Z'_S \cup N'_S$, where $i \in [2, T - 1]$. We assume that there exist temporal edges $u_t v_t \in E(G)$, where $t \in \{i - 1, i + 1\}$, such that $u_t \in (\tau(M_S) \setminus X) \cup N''_S$, $v_t \in \tau(Z_S)$ and $v_t \notin Z'_S$, thus arcs from s to v_t^+ are added. The dashed arcs represent deletable arcs.

Finally, Step 2 enforces the number of deleted arcs to correspond to the span of a solution. That is, it ensures that if we want to add to X^* a set of h vertices of base vertex $v \in Z_S$ to our solution of RESTRICTED TIMELINE COVER (so with a span equal to $h - 1$), then we have to delete $h - 1$ deletable arcs of the corresponding gadget of H in order to obtain a solution to CONSTRAINED DIGRAPH PAIR CUT (and vice-versa). Indeed, consider the gadget in Fig. 3. If v_i is not included in X^* , then in the gadget s reaches h positive vertices v_1^+, \dots, v_r^+ (and v_i^-). It follows that vertices $b_{v,l}, \dots, b_{v,r}$, $c_{v,l}, \dots, c_{v,r}$ and $d_{v,l}, \dots, d_{v,r}$ are all reachable from s . The pairs $\{d_{v,x}, b_{v,y}\}$ defined at Step 2, where either $l \leq x \leq y \leq r - 1$ if $r < i$, or $l + 1 \leq x \leq y \leq r$ if $l > i$, ensures that arcs $(c_{v,j}, d_{v,j})$, with $j \in [l, r - 1]$ in the former case or with $j \in [l + 1, r]$ in the latter case, are deleted.

If v_i is included in X^* , then in the gadget s reaches $h - 1$ positive vertices v_1^+, \dots, v_r^+ , with $i \in [l, r]$, and must not reach negative vertex v_i^- . It follows that vertices $b_{v,l}, \dots, b_{v,r}$, $c_{v,l}, \dots, c_{v,r}$ and $d_{v,l}, \dots, d_{v,r}$ are all reachable from s . Then $h - 1$ arcs $(c_{v,j}, d_{v,j})$, with $j \in [l, r] \setminus \{i\}$, must be deleted, due to the pairs $\{d_{v,x}, b_{v,y}\}$, $\{c_{v,x}, d_{v,y}\}$ defined at Step 2.

Note that Step 2 is the reason we added dummy timestamps 1 and T . If v_1 or v_T were allowed to be in $Z'_S \cup N'_S$, we would need a different gadget for these cases.

► **Lemma 14.** *There exists a solution of RESTRICTED TIMELINE COVER that agrees with X if and only if there is $F \subseteq D$ with $|F| \leq k'$ such that s does not reach a forbidden pair in $H - F$. Moreover, given such a set F , a solution of RESTRICTED TIMELINE COVER can be computed in polynomial time.*

Sketch of the proof. (\Rightarrow) Suppose that there exists a solution X^* of RESTRICTED TIMELINE COVER that agrees with X . By definition of RESTRICTED TIMELINE COVER, X^* has span at most k . Note that for $v \in M_S$, the agreement requires that $X^* \cap \tau(v) = X \cap \tau(v)$, and so the span of v in X^* is the same as the span of v in X . Thus

$$\sum_{v \in Z_S \cup N_S} sp(v, X^*) \leq k - sp(X) = k'.$$

We may assume that for every $v \in V_B$, at least one of v_2, \dots, v_{T-1} is in X^* , as otherwise we add one arbitrarily without affecting the span (if only v_1 or v_T is in X^* , remove it first). For each $v \in Z_S \cup N_S$, consider the gadget corresponding to v in H and delete some of its dashed arcs as follows (we recommend referring to Figure 3).

First, if only one of $\tau(v)$ is in X^* , no action is required on the gadget. So assume that $X^* \cap \tau(v)$ has at least two vertices; in the following we denote $v_l = v_{\delta(v, X^*)}$ and $v_r = v_{\Delta(v, X^*)}$ the vertices associated with v having minimum and maximum timestamp, respectively, contained in X^* . We assume that $l, r \in [2, T-1]$ and $l < r$. Note that $X^* \cap \tau(v) = \{v_l, v_{l+1}, \dots, v_r\}$.

Let $v_i \in Z'_S \cup N'_S$, where $i \in [2, T-1]$. Then

- suppose that $l, r \in [2, i-1]$, then: delete every arc $(c_{v,q}, d_{v,q})$, with $l \leq q \leq r-1$
- suppose that with $l, r \in [i+1, T-1]$, then: delete every arc $(c_{v,q}, d_{v,q})$, with $l+1 \leq q \leq r$
- suppose that $l \in [2, i]$ and $r \in [i, T-1]$, then: delete every arc $(c_{v,q}, d_{v,q})$, with $l \leq q \leq i-1$, and delete every arc $(c_{v,q}, d_{v,q})$, with $i+1 \leq q \leq r$.

We see that by construction for all $v \in Z_S \cup N_S$, the number of arcs deleted in the gadget corresponding to v is equal to the number of vertices in $X^* \cap \tau(v)$ minus one, that is the span of v in X^* . Since these vertices have span at most k' , it follows that we deleted at most k' arcs from H . Denote by H' the graph obtained after deleting the aforementioned arcs. We argue that in H' , s does not reach a forbidden pair. To this end, we claim the following.

▷ **Claim 15.** For $v \in Z_S \cup N_S$ and $t \in [T]$, if s reaches v_t^+ in H' , then $v_t \in X^*$, and if s reaches v_t^- in H' , then $v_t \notin X^*$.

Now, armed with the above claim, we can prove that in H' , s does not reach both vertices of a forbidden pair $q \in P$, thus concluding this direction of the proof.

(\Leftarrow) Suppose that there is a set $F \subseteq D$ with at most k' arcs such that s does not reach a forbidden pair in $H - F$. Denote $H' = H - F$. We construct X^* from F , which will also show that it can be reconstructed from F in polynomial time. Define $X^* \subseteq V(G)$ as follows:

- for each $v \in M_S$, add every element of $X \cap \tau(M_S)$ to X^* ;
- for each $v_t \in V(G) \setminus \tau(M_S)$, we add v_t to X^* if and only if one of the following holds: (1) $v_t^+ \in V(H)$ and s reaches v_t^+ in H' ; or (2) $v_t^- \in V(H)$, and s does *not* reach v_t^- in H' ;
- for each $v_j, v_h \in X^*$ with $j < h$, add v_t to X^* for each $t \in [j+1, h-1]$.

Note that X^* agrees with X . Indeed, for $v \in M_S$, there is no gadget corresponding to v in the construction and thus we only add $X \cap \tau(v)$ to X^* . For $u \in N_S$, consider $u_t \in N''_S$ and a neighbor v_t of u_t in $\tau(Z_S)$. If $v_t \notin Z'_S$, Step 4 adds an undeletable arc from s to v_t^+ , hence s reaches that vertex and we put v_t in X^* . If $v_t \in Z'_S$, Step 4 adds $\{s, v_t^-\}$ to P , and thus s does not reach v_t^- in H' , and again we add v_t to X^* . Therefore, we add all the $\tau(Z_S)$ neighbors of u_t to X^* , and so it agrees with X . We can prove that X^* covers every temporal edge of G and that $sp(X^*) \leq k$. ◀

Wrapping up

Before concluding, we must show that we are able to use the results of [16] to get an FPT algorithm for CONSTRAINED DIGRAPH PAIR CUT, as we have presented it. As we mentioned, the FPT algorithm in [16] studied the vertex-deletion variant and does not consider undeletable elements, but this is mostly a technicality. Roughly speaking, in our variant, it suffices to replace each vertex with enough copies of the same vertex, and replace each deletable arc (u, v) with a new vertex, adding arcs from the u copies to that vertex, and arcs from that vertex to the v copies. Deleting (u, v) corresponds to deleting that new vertex. For undeletable arcs, we apply the same process but repeat it $k' + 1$ times.

► **Lemma 16.** *The CONstrained DIGRAPH PAIR CUT problem can be solved in time $O^*(2^k)$, where k is the number of arcs to delete.*

We are able now to prove the main result of our contribution.

► **Theorem 17.** *MINTIMELINECOVER on a temporal graph $G = (V_B, E, \mathcal{T})$ can be solved in time $O^*(2^{5k \log k})$.*

Proof. First, we discuss the correctness of the algorithm we presented. Assume that we have an ordering on the base vertices of G and that v is the first vertex of this ordering. A solution S of MINTIMELINECOVER on $G[\{v\}]$ is equal to $S = \emptyset$.

Then for i , with $i \in [2, |V_B|]$, let G_i be the temporal graph induced by the first i vertices and let w be the $i + 1$ -th vertex. Given a solution S of MINTIMELINECOVER on instance G_i of span at most k , we can decide whether there exists a solution of MINTIMELINECOVER on instance G_{i+1} by computing whether there exists a solution X^* of the RESTRICTED TIMELINE COVER problem on instance G_i, w, S . By Lemma 10 and by Theorem 11 if there exists such an X^* , then there exists a feasible assignment X such that X^* agrees with X . By Lemma 14 we can compute, via the reduction to CONstrained DIGRAPH PAIR CUT, whether there exists a solution of RESTRICTED TIMELINE COVER on instance on instance G_i, w, S , and if so obtain such a solution (if no such solution X^* exists, then Lemma 14 also says that we will never return a solution, since every feasible assignment X that we enumerate will lead to a negative instance of CONstrained DIGRAPH PAIR CUT). Thus the RESTRICTED TIMELINE COVER subproblem is solved correctly, and once it is solved on $G_{|V_B|}$, we have a solution to MINTIMELINECOVER.

Now, we discuss the complexity of the algorithm. We must solve RESTRICTED TIMELINE COVER $|V_B|$ times. For each iteration, by Theorem 11 we can enumerate the feasible assignments in $O(2^{4k \log k} T^3 n)$ time. For each such assignment, the reduction from RESTRICTED TIMELINE COVER to CONstrained DIGRAPH PAIR CUT requires polynomial time, and each generated instance can be solved in time $O^*(2^k)$. The time dependency on k is thus $O^*(2^{4k \log k} \cdot 2^k)$, which we simplify to $O^*(2^{5k \log k})$. ◀

4 Conclusion

We have presented an FPT algorithm for the MINTIMELINECOVER problem, a variant of VERTEX COVER on temporal graphs recently considered for timeline activities summarizations. We point out some relevant future directions on this topic: (1) to improve, if possible, the time complexity of MINTIMELINECOVER by obtaining a single exponential time algorithm (of the form $O^*(c^k)$); (2) to establish whether MINTIMELINECOVER admits a polynomial kernel, possibly randomized (which it might, since CONstrained DIGRAPH PAIR CUT famously admits a randomized polynomial kernel).

References

- 1 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Christoforos L. Raptopoulos. The temporal explorer who returns to the base. *J. Comput. Syst. Sci.*, 120:179–193, 2021. doi:10.1016/j.jcss.2021.04.001.
- 2 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. *J. Comput. Syst. Sci.*, 107:108–123, 2020. doi:10.1016/j.jcss.2019.08.002.
- 3 Benjamin Merlin Bumpus and Kitty Meeks. Edge exploration of temporal graphs. In Paola Flocchini and Lucia Moura, editors, *Combinatorial Algorithms - 32nd International Workshop, IWOCA 2021, Ottawa, ON, Canada, July 5-7, 2021, Proceedings*, volume 12757 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2021. doi:10.1007/978-3-030-79987-8_8.

- 4 Riccardo Dondi. Untangling temporal graphs of bounded degree. *Theor. Comput. Sci.*, 969:114040, 2023. doi:10.1016/j.tcs.2023.114040.
- 5 Riccardo Dondi and Mohammad Mehdi Hosseinzadeh. Finding colorful paths in temporal graphs. In Rosa María Benito, Chantal Cherifi, Hocine Cherifi, Esteban Moro, Luis M. Rocha, and Marta Sales-Pardo, editors, *Complex Networks & Their Applications X - Volume 1, Proceedings of the Tenth International Conference on Complex Networks and Their Applications COMPLEX NETWORKS 2021, Madrid, Spain, November 30 - December 2, 2021*, volume 1015 of *Studies in Computational Intelligence*, pages 553–565. Springer, 2021. doi:10.1007/978-3-030-93409-5_46.
- 6 Riccardo Dondi and Alexandru Popa. Timeline cover in temporal graphs: Exact and approximation algorithms. In Sun-Yuan Hsieh, Ling-Ju Hung, and Chia-Wei Lee, editors, *Combinatorial Algorithms - 34th International Workshop, IWOCA 2023, Tainan, Taiwan, June 7-10, 2023, Proceedings*, volume 13889 of *Lecture Notes in Computer Science*, pages 173–184. Springer, 2023. doi:10.1007/978-3-031-34347-6_15.
- 7 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *J. Comput. Syst. Sci.*, 119:1–18, 2021. doi:10.1016/j.jcss.2021.01.005.
- 8 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.
- 9 Vincent Froese, Pascal Kunz, and Philipp Zschoche. Disentangling the computational complexity of network untangling. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 2037–2043. ijcai.org, 2022. doi:10.24963/ijcai.2022/283.
- 10 Thekla Hamm, Nina Klobas, George B. Mertzios, and Paul G. Spirakis. The complexity of temporal vertex cover in small-degree graphs. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelfth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 10193–10201. AAAI Press, 2022.
- 11 Petter Holme. Modern temporal network theory: a colloquium. *The European Physical Journal B*, 88(9):234, 2015.
- 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 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Directed flow-augmentation. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 938–947, 2022.
- 14 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Directed flow-augmentation. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 938–947. ACM, 2022. doi:10.1145/3519935.3520018.
- 15 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Flow-augmentation iii: Complexity dichotomy for boolean csps parameterized by the number of unsatisfied constraints. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3218–3228. SIAM, 2023.
- 16 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.
- 17 Andrea Marino and Ana Silva. Königsberg sightseeing: Eulerian walks in temporal graphs. In Paola Flocchini and Lucia Moura, editors, *Combinatorial Algorithms - 32nd International Workshop, IWOCA 2021, Ottawa, ON, Canada, July 5-7, 2021, Proceedings*, volume 12757 of *Lecture Notes in Computer Science*, pages 485–500. Springer, 2021. doi:10.1007/978-3-030-79987-8_34.

12:16 An FPT Algorithm for Temporal Graph Untangling

- 18 Igor Razgon and Barry O’Sullivan. Almost 2-sat is fixed-parameter tractable. *J. Comput. Syst. Sci.*, 75(8):435–450, 2009. doi:10.1016/j.jcss.2009.04.002.
- 19 Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis. The network-untangling problem: from interactions to activity timelines. *Data Min. Knowl. Discov.*, 35(1):213–247, 2021. doi:10.1007/s10618-020-00717-5.
- 20 Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path problems in temporal graphs. *Proc. VLDB Endow.*, 7(9):721–732, 2014. doi:10.14778/2732939.2732945.
- 21 Huanhuan Wu, James Cheng, Yiping Ke, Silu Huang, Yuzhen Huang, and Hejun Wu. Efficient algorithms for temporal path computation. *IEEE Trans. Knowl. Data Eng.*, 28(11):2927–2942, 2016. doi:10.1109/TKDE.2016.2594065.
- 22 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.

Budgeted Matroid Maximization: a Parameterized Viewpoint

Ilan Doron-Arad ✉

Computer Science Department, Technion, Haifa, Israel

Ariel Kulik ✉

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Hadas Shachnai ✉

Computer Science Department, Technion, Haifa, Israel

Abstract

We study budgeted variants of well known maximization problems with multiple matroid constraints. Given an ℓ -matchoid \mathcal{M} on a ground set E , a profit function $p : E \rightarrow \mathbb{R}_{\geq 0}$, a cost function $c : E \rightarrow \mathbb{R}_{\geq 0}$, and a budget $B \in \mathbb{R}_{\geq 0}$, the goal is to find in the ℓ -matchoid a feasible set S of maximum profit $p(S)$ subject to the budget constraint, i.e., $c(S) \leq B$. The *budgeted ℓ -matchoid* (BM) problem includes as special cases budgeted ℓ -dimensional matching and budgeted ℓ -matroid intersection. A strong motivation for studying BM from parameterized viewpoint comes from the APX-hardness of unbudgeted ℓ -dimensional matching (i.e., $B = \infty$) already for $\ell = 3$. Nevertheless, while there are known FPT algorithms for the unbudgeted variants of the above problems, the *budgeted* variants are studied here for the first time through the lens of parameterized complexity.

We show that BM parametrized by solution size is $W[1]$ -hard, already with a degenerate single matroid constraint. Thus, an exact parameterized algorithm is unlikely to exist, motivating the study of *FPT-approximation schemes* (FPAS). Our main result is an FPAS for BM (implying an FPAS for ℓ -dimensional matching and budgeted ℓ -matroid intersection), relying on the notion of representative set – a small cardinality subset of elements which preserves the optimum up to a small factor. We also give a lower bound on the minimum possible size of a representative set which can be computed in polynomial time.

2012 ACM Subject Classification Theory of computation

Keywords and phrases budgeted matching, budgeted matroid intersection, knapsack problems, FPT-approximation scheme

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.13

Related Version *Full Version*: <https://arxiv.org/abs/2307.04173> [7]

Funding *Ariel Kulik*: Research supported by the European Research Council (ERC) consolidator grant no. 725978 SYSTEMATICGRAPH.

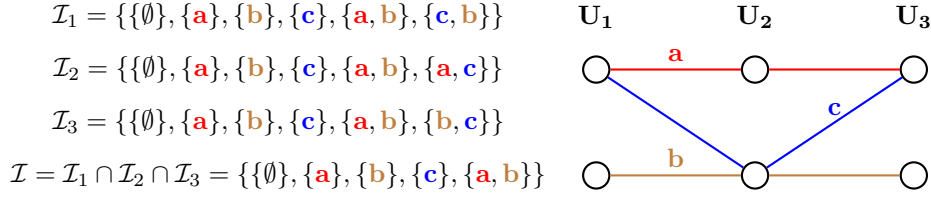
1 Introduction

Numerous combinatorial optimization problems can be interpreted as *constrained budgeted problems*. In this setting, we are given a ground set E of elements and a family $\mathcal{I} \subseteq 2^E$ of subsets of E known as the *feasible sets*. We are also given a cost function $c : E \rightarrow \mathbb{R}$, a profit function $p : E \rightarrow \mathbb{R}$, and a budget $B \in \mathbb{R}$. A *solution* is a feasible set $S \in \mathcal{I}$ of bounded cost $c(S) \leq B$.¹ Broadly speaking, the goal is to find a solution S of maximum profit. Notable examples include budgeted matching [1] and budgeted matroid intersection [3, 20], shortest weight-constrained path [18], and constrained minimum spanning trees [36].

¹ For a function $f : A \rightarrow \mathbb{R}$ and a subset of elements $C \subseteq A$, define $f(C) = \sum_{e \in C} f(e)$.



13:2 Budgeted Matroid Maximization: Parameterized Viewpoint



■ **Figure 1** A 3-dimensional matching viewed as a 3-matroid intersection. Each element in $E \subset U_1 \times U_2 \times U_3$ is represented by a path of three vertices, where the paths are distinguished by different colors, that is $E = \{a, b, c\}$. There is a matroid constraint (E, \mathcal{I}_i) for each $U_i, i = 1, 2, 3$. The feasible sets \mathcal{I} for the matching are exactly the common independent sets of $\mathcal{I}_i, i = 1, 2, 3$.

Despite the wide interest in constrained budgeted problems in approximation algorithms, not much is known about this intriguing family of problems in terms of parameterized complexity. In this work, we study budgeted maximization with the fairly general ℓ -dimensional matching, ℓ -matroid intersection, and ℓ -matchoid constraints.

An ℓ -dimensional matching constraint is a set system (E, \mathcal{I}) , where $E \subseteq U_1 \times \dots \times U_\ell$ for ℓ sets U_1, \dots, U_ℓ . The feasible sets \mathcal{I} are all subsets $S \subseteq E$ which satisfy the following. For any two distinct tuples $(e_1, \dots, e_\ell), (f_1, \dots, f_\ell) \in S$ and every $i \in [\ell]$ it holds that $e_i \neq f_i$.² Informally, the input for budgeted ℓ -dimensional matching is an ℓ -dimensional matching constraint (E, \mathcal{I}) , profits and costs for the elements in E , and a budget. The objective is to find a feasible set which maximizes the profit subject to the budget constraint (see below the formal definition).

We now define an ℓ -matroid intersection. A *matroid* is a set system (E, \mathcal{I}) , where E is a finite set and $\mathcal{I} \subseteq 2^E$, such that

- $\emptyset \in \mathcal{I}$.
- The *hereditary property*: for all $A \in \mathcal{I}$ and $B \subseteq A$ it holds that $B \in \mathcal{I}$.
- The *exchange property*: for all $A, B \in \mathcal{I}$ where $|A| > |B|$ there is $e \in A \setminus B$ such that $B \cup \{e\} \in \mathcal{I}$.

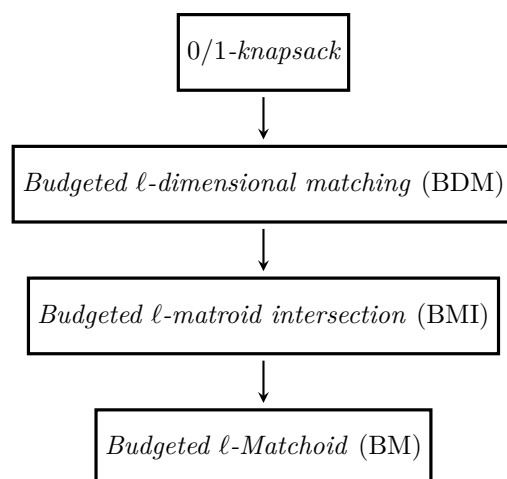
For a fixed $\ell \geq 1$, let $(E, \mathcal{I}_1), (E, \mathcal{I}_2), \dots, (E, \mathcal{I}_\ell)$ be ℓ matroids on the same ground set E . An ℓ -matroid intersection is a set system (E, \mathcal{I}) where $\mathcal{I} = \mathcal{I}_1 \cap \mathcal{I}_2 \cap \dots \cap \mathcal{I}_\ell$. Observe that ℓ -dimensional matching, where $E \subseteq U_1 \times \dots \times U_\ell$, is a special case of ℓ -matroid intersection: For each $i \in [\ell]$, define a partition matroid (E, \mathcal{I}_i) , where any feasible set $S \in \mathcal{I}_i$ may contain each element $e \in U_i$ in the i -th coordinate at most once, i.e.,

$$\mathcal{I}_i = \{S \subseteq E \mid \forall (e_1, \dots, e_\ell) \neq (f_1, \dots, f_\ell) \in S : e_i \neq f_i\}.$$

We give an illustration in Figure 1. It can be shown that (E, \mathcal{I}_i) is a matroid for all $i \in \ell$ (see, e.g., [37]).

The above constraint families can be generalized to the notion of ℓ -matchoid. Informally, an ℓ -matchoid is an intersection of an unbounded number of matroids, where each element belongs to at most ℓ of the matroids. Formally, for any $\ell \geq 1$, an ℓ -matchoid on a set E is a collection $\mathcal{M} = \{M_i = (E_i, \mathcal{I}_i)\}_{i \in [s]}$ of $s \in \mathbb{N}$ matroids, where for each $i \in [s]$ it holds that $E_i \subseteq E$, and every $e \in E$ belongs to at most ℓ sets in $\{E_1, \dots, E_s\}$, i.e., $|\{i \in [s] \mid e \in E_i\}| \leq \ell$. A set $S \subseteq E$ is *feasible* for \mathcal{M} if for all $i \in [s]$ it holds that $S \cap E_i \in \mathcal{I}_i$. Let $\mathcal{I}(\mathcal{M}) = \{S \subseteq E \mid \forall i \in [s] : S \cap E_i \in \mathcal{I}_i\}$ be all feasible sets of \mathcal{M} . For all $k \in \mathbb{N}$, we

² For any $k \in \mathbb{N}$ let $[k] = \{1, 2, \dots, k\}$.



■ **Figure 2** An overview of constrained budgeted problems. An arrow from problem A to problem B indicates that A is a special case of B .

use $\mathcal{M}_k \subseteq \mathcal{I}(\mathcal{M})$ to denote all feasible sets of \mathcal{M} of cardinality at most k . Clearly, ℓ -matroid intersection (and also ℓ -dimensional matching) is the special case of ℓ -matchoid where the $s(= \ell)$ matroids are defined over the same ground set E .

In the *budgeted ℓ -matchoid (BM)* problem, we are given an ℓ -matchoid along with a cost function, profit function, and a budget; our goal is to maximize the profit of a feasible set under the budget constraint. The *budgeted ℓ -matroid intersection (BMI)* and *budgeted ℓ -dimensional matching (BDM)* are the special cases where the ℓ -matchoid is an ℓ -matroid intersection and ℓ -dimensional matching, respectively. Each of these problems generalizes the classic 0/1-knapsack, where all sets are feasible. Figure 2 shows the relations between the problems. Henceforth, we focus on the BM problem.

Formally, a BM instance is a tuple $I = (E, \mathcal{M}, c, p, B, k, \ell)$, where E is a ground set of elements, \mathcal{M} is an ℓ -matchoid on E , $c : E \rightarrow \mathbb{N}_{>0}$ is a cost function, $p : E \rightarrow \mathbb{N}_{>0}$ is a profit function, $B \in \mathbb{N}_{>0}$ is a budget, and $k, \ell \in \mathbb{N}_{>0}$ are integer parameters.³ In addition, each matroid $(E_i, \mathcal{I}_i) \in \mathcal{M}$ has a *membership oracle*, which tests whether a given subset of E_i belongs to \mathcal{I}_i or not in a single query. Indeed, with no membership oracle, the representation size for a matroid over n elements may be exponential in n . A *solution* of I is a feasible set $S \in \mathcal{M}_k$ such that $c(S) \leq B$. The objective is to find a solution S of I such that $p(S)$ is maximized. We consider algorithms parameterized by k and ℓ (equivalently, $k + \ell$).

We note that even with no budget constraint (i.e., $c(E) \leq B$), where the ℓ -matchoid is restricted to be a 3-dimensional matching, BM is MAX SNP-complete [26], i.e., it cannot admit a *polynomial time approximation scheme (PTAS)* unless $P=NP$. On the other hand, the ℓ -dimensional matching and even the ℓ -matchoid problem (without a budget), parameterized by ℓ and the solution size k , are *fixed parameter tractable (FPT)* [19, 22]. This motivates our study of BM through the lens of parameterized complexity. We first observe that BM parameterized by the solution size is W[1]-hard, already with a *uniform* matroid where all sets are feasible (i.e., knapsack parametrized by the cardinality of the solution, k).

► **Theorem 1.** BM is W[1]-hard.

³ We assume integral values for simplicity; our results can be generalized also for real values.

13:4 Budgeted Matroid Maximization: Parameterized Viewpoint

By the hardness result in Theorem 1, the best we can expect for BM in terms of parameterized algorithms, is an *FPT-approximation scheme (FPAS)*. An FPAS with parameterization κ for a maximization problem Π is an algorithm whose input is an instance I of Π and an $\varepsilon > 0$, which produces a solution S of I of value $(1 - \varepsilon) \cdot \text{OPT}(I)$ in time $f(\varepsilon, \kappa(|I|)) \cdot |I|^{O(1)}$ for some computable function f , where $|I|$ denotes the encoding size of I and $\text{OPT}(I)$ is the optimum value of I . We refer the reader to [34, 14] for comprehensive surveys on parameterized approximation schemes and parameterized approximations in general. To derive an FPAS for BM, we use a small cardinality *representative set*, which is a subset of elements containing the elements of an *almost* optimal solution for the instance. The representative set has a cardinality depending solely on $\ell, k, \varepsilon^{-1}$ and is constructed in FPT time. Formally,

► **Definition 2.** Let $I = (E, \mathcal{M}, c, p, B, k, \ell)$ be a BM instance, $0 < \varepsilon < \frac{1}{2}$ and $R \subseteq E$. Then R is a representative set of I and ε if there is a solution S of I such that the following holds.

1. $S \subseteq R$.
2. $p(S) \geq (1 - 2\varepsilon) \cdot \text{OPT}(I)$.

We remark that Definition 2 slightly resembles the definition of *lossy kernel* [31]. Nonetheless, the definition of lossy kernel does not apply to problems in the oracle model, including BM (see Section 6 for further details).

The main technical contribution of this paper is the design of a small cardinality representative set for BM. Our representative set is constructed by forming a collection of $f(\ell, k, \varepsilon^{-1})$ *profit classes*, where the elements of each profit class have roughly the same profit. Then, to construct a representative set for the instance, we define a residual problem for each profit class which enables to circumvent the budget constraint. These residual problems can be solved efficiently using a construction of [22]. We show that combining the solutions for the residual problems, we obtain a representative set. In the following, we use $\tilde{O}(n)$ for $O(n \cdot \text{poly}(\log(n)))$.

► **Theorem 3.** There is an algorithm that given a BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$ and $0 < \varepsilon < \frac{1}{2}$, returns in time $|I|^{O(1)}$ a representative set $R \subseteq E$ of I and ε such that $|R| = \tilde{O}(\ell^{(k-1) \cdot \ell} \cdot k^2 \cdot \varepsilon^{-2})$.

Given a small cardinality representative set, it is easy to derive an FPAS. Specifically, using an exhaustive enumeration over the representative set as stated in Theorem 3, we can construct the following FPAS for BM, which naturally applies also for BMI and BDM.

► **Theorem 4.** For any BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$ and $0 < \varepsilon < \frac{1}{2}$, there is an FPAS whose running time is $|I|^{O(1)} \cdot \tilde{O}(\ell^{k^2 \cdot \ell} \cdot k^{O(k)} \cdot \varepsilon^{-2k})$.

Our FPAS cannot be significantly improved even for very restricted ℓ -matchoids. Namely, even if the ℓ -matchoid is a single matroid there cannot be an FPAS for BMI with running time polynomial in $\frac{1}{\varepsilon}$, where ε is the given error parameter [11]. To complement the above construction of a representative set, we show that even for the special case of an ℓ -dimensional matching constraint, it is unlikely that a representative set of significantly smaller cardinality can be constructed in polynomial time. The next result applies to the special case of BDM.

► **Theorem 5.** For any function $f : \mathbb{N} \rightarrow \mathbb{N}$, and $c_1, c_2 \in \mathbb{R}$ such that $c_2 - c_1 < 0$, there is no algorithm which finds for a given BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$ and $0 < \varepsilon < \frac{1}{2}$ a representative set of size $O(f(\ell) \cdot k^{\ell - c_1} \cdot \frac{1}{\varepsilon^{c_2}})$ of I and ε in time $|I|^{O(1)}$, unless $\text{coNP} \subseteq \text{NP/poly}$.

In the proof of Theorem 1, we use a lower bound on the kernel size of the *Perfect 3-Dimensional Matching* (3-PDM) problem, due to Dell and Marx [5, 6].⁴ In our hardness result, we are able to efficiently construct a kernel for 3-PDM using a representative set for BM, already for the special case of 3-dimensional matching constraint, uniform costs, and uniform profits.

Related Work

While BM is studied here for the first time, special cases of the problem have been extensively studied from both parameterized and approximative points of view. For maximum weighted ℓ -matchoid without a budget constraint, Huang and Ward [22] obtained a deterministic FPT algorithm, and algorithms for a more general problem, involving a *coverage function* objective rather than a linear objective. Their result differentiates the ℓ -matchoid problem from the matroid ℓ -parity problem which cannot have an FPT algorithm in general matroids [32, 24]. Interestingly, when the matroids are given a linear representation, the matroid ℓ -parity problem admits a randomized FPT algorithm [35, 15] and a deterministic FPT algorithm [30]. We use a construction of [22] as a building block of our algorithm.

The ℓ -dimensional k -matching problem (i.e., the version of the problem with no budget parametrized by k and ℓ) has received considerable attention in previous studies. Goyal et al. [19] presented a deterministic FPT algorithm whose running time is $O^*(2.851^{(\ell-1)\cdot k})$ for the weighted version of ℓ -dimensional k -matching, where O^* is used to suppress polynomial factor in the running time. This result improves a previous result of [4]. For the unweighted version of ℓ -dimensional k -matching, the state of the art is a randomized FPT algorithm with running time $O^*(2^{(\ell-2)\cdot k})$ [2], improving a previous result for the problem [27].

Budgeted problems are well studied in approximation algorithms. As BM is a generalization of classic 0/1-knapsack, it is known to be NP-hard. However, while knapsack admits a *fully PTAS (FPTAS)* [33], BM is unlikely to admit a PTAS, even for the special case of 3-dimensional matching with no budget constraint [26]. Consequently, there has been extensive research work to identify special cases of BM which admit approximation schemes.

For the budgeted matroid independent set (i.e., the special case of BM where the ℓ -matchoid consists of a single matroid), Doron-Arad et al. [9] developed an *efficient PTAS (EPTAS)* using the representative set based technique. This algorithm was later generalized in [8] to tackle budgeted matroid intersection and budgeted matching (both are special cases of BM where $\ell = 2$), improving upon a result of Berger et al. [1]. For the special case where the matroid is a *laminar matroid*, there is an FPTAS [10]. We generalize some of the technical ideas of [9, 8] to the setting of ℓ -matchoid and parametrized approximations.

Organization of the paper. Section 2 describes our construction of a representative set. In Section 3 we present our FPAS for BM. Section 4 contains the proofs of the hardness results given in Theorem 1 and in Theorem 1. In Section 5 we present an auxiliary approximation algorithm for BM. We conclude in Section 6 with a summary and some directions for future work.

⁴ We refer the reader e.g., to [16], for the formal definition of kernels.

2 Representative Set

In this section we construct a representative set for BM. Our first step is to round the profits of a given instance, and to determine the low profit elements that can be discarded without incurring significant loss of profit. We find a small cardinality representative set from which an almost optimal solution can be selected via enumeration yielding an FPAS (see Section 3).

We proceed to construct a representative set whose cardinality depends only on ε^{-1} , k , and ℓ . This requires the definition of *profit classes*, namely, a partition of the elements into groups, where the elements in each group have similar profits. Constructing a representative set using this method requires an approximation of the optimum value of the input BM instance I . To this end, we use a $\frac{1}{2\ell}$ -approximation $\alpha = \text{ApproxBM}(I)$ of the optimum value $\text{OPT}(I)$ described below.

► **Lemma 6.** *Given a BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$, there is an algorithm ApproxBM which returns in time $|I|^{O(1)}$ a value α such that $\frac{\text{OPT}(I)}{2\ell} \leq \alpha \leq \text{OPT}(I)$.*

The proof of Lemma 6 is given in Section 5. The proof utilizes a known approximation algorithm for the unbudgeted version of BM [23, 25] which is then transformed into an approximation algorithm for BM using a technique of [29].

The first step in designing the profit classes is to determine a set of *profitable* elements required for obtaining an almost optimal solution. This set allows us to

construct only a small number of profit classes. We define the set of *profitable* elements w.r.t. I, α , and ε as

$$H[I, \alpha, \varepsilon] = \left\{ e \in E \mid \frac{\varepsilon \cdot \alpha}{k} < p(e) \leq 2 \cdot \ell \cdot \alpha \right\}. \quad (1)$$

The upper bound on $p(e)$ is purely technical; it is used later to upper bound the number of profit classes. When clear from the context, we simply use $H = H[I, \alpha, \varepsilon]$. Consider the non-profitable elements. The next lemma states that omitting these elements indeed has small effect on the profit of the solution set.

► **Lemma 7.** *For every BM instance $I = (E, \mathcal{C}, c, p, B, k, \ell)$, $\frac{\text{OPT}(I)}{2\ell} \leq \alpha \leq \text{OPT}(I)$, $0 < \varepsilon < \frac{1}{2}$, and $S \in \mathcal{M}_k$ it holds that $p(S \setminus H[I, \alpha, \varepsilon]) \leq \varepsilon \cdot \text{OPT}(I)$.*

Proof. We note that

$$p(S \setminus H[I, \alpha(I), \varepsilon]) \leq k \cdot \frac{\varepsilon \cdot \alpha}{k} = \varepsilon \cdot \alpha \leq \varepsilon \cdot \text{OPT}(I).$$

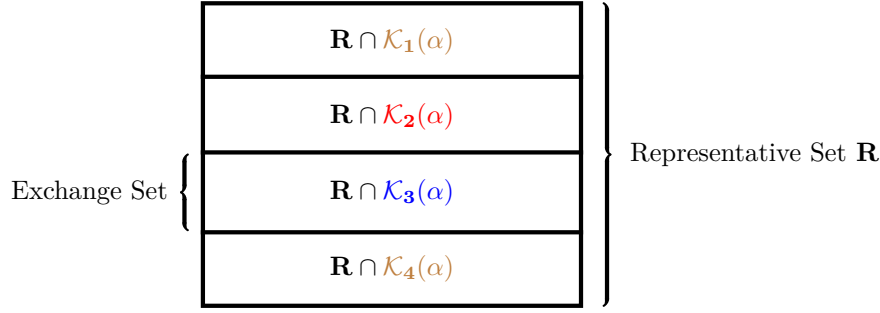
The first inequality holds since each element in $S \setminus H[I, \alpha(I), \varepsilon]$ has profit at most $\frac{\varepsilon \cdot \alpha}{k}$ by (1); in addition, since $S \in \mathcal{M}_k$ it follows that S contains at most k elements. The second inequality holds as $\alpha \leq \text{OPT}(I)$. ◀

Using Lemma 7, our representative set can be constructed exclusively from the profitable elements. We can now partition the profitable elements into a small number of *profit classes*. There is a profit class $\mathcal{K}_r(\alpha)$ for a suitable range of profit *boundaries* r and a constant factor approximation α for OPT. Specifically, let

$$D(I, \varepsilon) = \left\{ r \in \mathbb{N}_{>0} \mid (1 - \varepsilon)^{r-1} \geq \frac{\varepsilon}{2 \cdot \ell \cdot k} \right\} \quad (2)$$

be the set of *boundaries* for the profit classes (defined below). We simplify by $D = D(I, \varepsilon)$. For all $r \in D$, and $\frac{\text{OPT}(I)}{2\ell} \leq \alpha \leq \text{OPT}(I)$, define the r -*profit class* as

$$\mathcal{K}_r(\alpha) = \left\{ e \in E \mid \frac{p(e)}{2 \cdot \ell \cdot \alpha} \in ((1 - \varepsilon)^r, (1 - \varepsilon)^{r-1}] \right\}. \quad (3)$$



■ **Figure 3** An illustration of our construction of a representative set R , using a union of exchange sets, one for each profit class $\mathcal{K}_1(\alpha)$, $\mathcal{K}_2(\alpha)$, $\mathcal{K}_3(\alpha)$, $\mathcal{K}_4(\alpha)$.

In words, each profit class $r \in D$ contains profitable elements (and may contain some elements that are *almost* profitable due to our $\frac{1}{2\ell}$ -approximation for $\text{OPT}(I)$), where the profits of any two elements that belong to the r -profit class can differ by at most a multiplicative factor of $(1 - \varepsilon)$. We use the following simple upper bound on the number of profit classes.

► **Lemma 8.** *For every BM instance I and $0 < \varepsilon < \frac{1}{2}$ there are $O(k \cdot \ell \cdot \varepsilon^{-2})$ profit classes.*

Proof. We note that

$$\log_{1-\varepsilon} \left(\frac{\varepsilon}{2\ell \cdot k} \right) \leq \frac{\ln \left(\frac{2\ell \cdot k}{\varepsilon} \right)}{-\ln(1-\varepsilon)} \leq \frac{2\ell \cdot k \cdot \varepsilon^{-1}}{\varepsilon}. \quad (4)$$

The second inequality follows from $x < -\ln(1-x)$, $\forall x > -1, x \neq 0$, and $\ln(y) < y, \forall y > 0$. By (2) the number of profit classes is bounded by

$$|D| \leq \log_{1-\varepsilon} \left(\frac{\varepsilon}{2\ell \cdot k} \right) + 1 = O(k \cdot \ell \cdot \varepsilon^{-2}). \quad (5)$$

The last inequality follows from (4). ◀

Next, we define an *exchange set* for each profit class. This facilitates the construction of a representative set. Intuitively, a subset of elements X forms an exchange set for a profit class $\mathcal{K}_r(\alpha)$ if any feasible set Δ and element $a \in (\Delta \cap \mathcal{K}_r(\alpha)) \setminus X$ can be replaced (while maintaining feasibility) by some element $b \in (X \cap \mathcal{K}_r(\alpha)) \setminus \Delta$ such that the cost of b is upper bounded by the cost of a . Formally,

► **Definition 9.** *Let $I = (E, \mathcal{M}, c, p, B, k, \ell)$ be a BM instance, $0 < \varepsilon < \frac{1}{2}$, $\frac{\text{OPT}(I)}{2\ell} \leq \alpha \leq \text{OPT}(I)$, $r \in D(I, \varepsilon)$, and $X \subseteq \mathcal{K}_r(\alpha)$. We say that X is an exchange set for I, ε, α , and r if: For all $\Delta \in \mathcal{M}_k$ and $a \in (\Delta \cap \mathcal{K}_r(\alpha)) \setminus X$ there is $b \in (\mathcal{K}_r(\alpha) \cap X) \setminus \Delta$ satisfying*

- $c(b) \leq c(a)$.
- $\Delta - a + b \in \mathcal{M}_k$.

The key argument in this section is that if a set $R \subseteq E$ satisfies that $R \cap \mathcal{K}_r(\alpha)$ is an exchange set for any $r \in D$, then R is a representative set. This allows us to construct a representative set using a union of disjoint exchange sets, one for each profit class. We give an illustration in Figure 3.

► **Lemma 10.** *Let $I = (E, \mathcal{M}, c, p, B, k, \ell)$ be a BM instance, $0 < \varepsilon < \frac{1}{2}$, $\frac{\text{OPT}(I)}{2\ell} \leq \alpha \leq \text{OPT}(I)$, and $R \subseteq E$. If for all $r \in D = D(I, \varepsilon)$ it holds that $R \cap \mathcal{K}_r(\alpha)$ is an exchange set for I, ε, α , and r , then R is a representative set of I and ε .*

13:8 Budgeted Matroid Maximization: Parameterized Viewpoint

For the proof of Lemma 10, we define a *substitution* of some feasible set $G \in \mathcal{M}_k$. We will use G later only as an optimal solution; however, we can state the following claims for a general $G \in \mathcal{M}_k$. We require that a substitution preserves the number of profitable elements in G from each profit class, so a substitution guarantees a profit similar to the profit of G .

► **Definition 11.** For $G \in \mathcal{M}_k$ and $Z_G \subseteq \bigcup_{r \in D} \mathcal{K}_r(\alpha)$, we say that Z_G is a substitution of G if the following holds.

1. $Z_G \in \mathcal{M}_k$.
2. $c(Z_G) \leq c(G)$.
3. For all $r \in D$ it holds that $|\mathcal{K}_r(\alpha) \cap Z_G| = |\mathcal{K}_r(\alpha) \cap G|$.

Proof of Lemma 10. We first show that every set $G \in \mathcal{M}_k$ has a substitution which is a subset of R .

▷ **Claim 12.** For any $G \in \mathcal{M}_k$ there is a substitution Z_G of G such that $Z_G \subseteq R$.

Proof. Let $G \in \mathcal{M}_k$ and let Z_G be a substitution of G such that $|Z_G \cap R|$ is maximal among all substitutions of G ; formally, let $\mathcal{S}(G)$ be all substitutions of G and let

$$Z_G \in \{Z \in \mathcal{S}(G) \mid |Z \cap R| = \max_{Z' \in \mathcal{S}(G)} |Z' \cap R|\}.$$

Since $G \cap \bigcup_{r \in D} \mathcal{K}_r(\alpha)$ is in particular a substitution of G it follows that $\mathcal{S}(G) \neq \emptyset$; thus, Z_G is well defined. Assume towards a contradiction that there is $a \in Z_G \setminus R$; then, by Definition 11 there is $r \in D$ such that $a \in \mathcal{K}_r(\alpha)$. Because $R \cap \mathcal{K}_r(\alpha)$ is an exchange set for I, ε, α , and r , by Definition 9 there is $b \in (\mathcal{K}_r(\alpha) \cap R) \setminus Z_G$ such that $c(b) \leq c(a)$ and $Z_G - a + b \in \mathcal{M}_k$. Then, the properties of Definition 11 are satisfied for $Z_G - a + b$ by the following.

1. $Z_G - a + b \in \mathcal{M}_k$ by the definition of b .
2. $c(Z_G - a + b) \leq c(Z_G) \leq c(G)$ because $c(b) \leq c(a)$.
3. for all $r' \in D$ it holds that $|\mathcal{K}_{r'}(\alpha) \cap (Z_G - a + b)| = |\mathcal{K}_{r'}(\alpha) \cap Z_G| = |\mathcal{K}_{r'}(\alpha) \cap G|$ because $a, b \in \mathcal{K}_r(\alpha)$.

By the above, and using and Definition 11, we have that $Z_G + a - b$ is a substitution of G ; that is, $Z_G + a - b \in \mathcal{S}(G)$. Moreover,

$$|R \cap (Z_G - a + b)| > |R \cap Z_G| = \max_{Z \in \mathcal{S}(G)} |Z \cap R|. \quad (6)$$

The first inequality holds since $a \in Z_G \setminus R$ and $b \in R$. Thus, we have found a substitution of G which contains more elements in R than $Z_G \in \mathcal{S}(G)$. A contradiction to the definition of Z_G as a substitution of G having a maximum number of elements in R . Hence, $Z_G \subseteq R$, as required. ◁

Let G be an optimal solution for I . We complete the proof of Lemma 10 by showing that a substitution of G which is a subset of R yields a profit at least $(1 - 2\varepsilon) \cdot \text{OPT}(I)$. Let $H[I, \alpha, \varepsilon] = H$ be the set of profitable elements w.r.t. I, α and ε (as defined in (1)). By Claim 12, as $G \in \mathcal{M}_k$, it has a substitution $Z_G \subseteq R$. Then,

$$\begin{aligned} p(Z_G) &\geq \sum_{r \in D} p(\mathcal{K}_r(\alpha) \cap Z_G) \\ &\geq \sum_{r \in D \text{ s.t. } \mathcal{K}_r(\alpha) \neq \emptyset} |\mathcal{K}_r(\alpha) \cap Z_G| \cdot \min_{e \in \mathcal{K}_r(\alpha)} p(e) \\ &\geq \sum_{r \in D \text{ s.t. } \mathcal{K}_r(\alpha) \neq \emptyset} |\mathcal{K}_r(\alpha) \cap G| \cdot (1 - \varepsilon) \cdot \max_{e \in \mathcal{K}_r(\alpha)} p(e) \\ &\geq (1 - \varepsilon) \cdot p(G \cap H). \end{aligned} \quad (7)$$

The third inequality follows from (3), and from Property 3 in Definition 11. The last inequality holds since for every $e \in H$ there is $r \in D$ such that $e \in \mathcal{K}_r(\alpha)$, by (1) and (3). Therefore,

$$\begin{aligned}
p(Z_G) &\geq (1 - \varepsilon) \cdot p(G \cap H) \\
&= (1 - \varepsilon) \cdot (p(G) - p(G \setminus H)) \\
&\geq (1 - \varepsilon) \cdot p(G) - p(G \setminus H) \\
&\geq (1 - \varepsilon) \cdot p(G) - \varepsilon \cdot \text{OPT}(I) \\
&= (1 - \varepsilon) \cdot \text{OPT}(I) - \varepsilon \cdot \text{OPT}(I) \\
&= (1 - 2\varepsilon) \cdot \text{OPT}(I).
\end{aligned} \tag{8}$$

The first inequality follows from (7). The last inequality holds by Lemma 7. The second equality holds since G is an optimal solution for I . To conclude, by Properties 1 and 2 in Definition 11, it holds that $Z_G \in \mathcal{M}_k$, and $c(Z_G) \leq c(G) \leq B$; thus, Z_G is a solution for I . Also, by (8), it holds that $p(Z_G) \geq (1 - 2\varepsilon) \cdot \text{OPT}(I)$ as required (see Definition 2). ◀

By Lemma 10, our end goal of constructing a representative set is reduced to efficiently finding exchange sets for all profit classes. This can be achieved by the following result, which is a direct consequence of Theorem 3.6 in [22]. As the result of [22] refers to a *maximization* version of exchange sets, we first present an analogue to Definition 9 for maximization exchange sets (as in [22]), using our notation.

► **Definition 13.** Let $I = (E, \mathcal{M}, c, p, B, k, \ell)$ be a BM instance, $0 < \varepsilon < \frac{1}{2}$, $\frac{\text{OPT}(I)}{2\ell} \leq \alpha \leq \text{OPT}(I)$, $r \in D(I, \varepsilon)$, and $X \subseteq \mathcal{K}_r(\alpha)$. We say that X is a maximization exchange set for I, ε, α , and r if: For all $\Delta \in \mathcal{M}_k$ and $a \in (\Delta \cap \mathcal{K}_r(\alpha)) \setminus X$ there is $b \in (\mathcal{K}_r(\alpha) \cap X) \setminus \Delta$ satisfying

- $c(a) \leq c(b)$.
- $\Delta - a + b \in \mathcal{M}_k$.

We remark that the same construction and the proof of Theorem 3.6 in [21] hold for our exchange sets (in Definition 9) as well. Hence, we have the following.

► **Lemma 14.** Given a BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$, $0 < \varepsilon < \frac{1}{2}$, $\frac{\text{OPT}(I)}{2\ell} \leq \alpha \leq \text{OPT}(I)$, and $r \in D(I, \varepsilon)$, there is an algorithm `ExSet` which returns in time $\tilde{O}(\ell^{(k-1) \cdot \ell} \cdot k)$. $|I|^{O(1)}$ an exchange set X for I, ε, α , and r , such that $|X| = \tilde{O}(\ell^{(k-1) \cdot \ell} \cdot k)$.

■ **Algorithm 1** `RepSet`($I = (E, \mathcal{M}, c, p, B, k, \ell), \varepsilon$).

input : A BM instance I , and an error parameter $0 < \varepsilon < \frac{1}{2}$.
output : A representative set R of I and ε .

- 1 **if** $\ell^{(k-1) \cdot \ell} \cdot k^2 \cdot \varepsilon^{-2} > |I|$ **then**
- 2 | Return E
- 3 Compute $\alpha \leftarrow \text{ApproxBM}(I)$.
- 4 **for** $r \in D(I, \varepsilon)$ **do**
- 5 | $R \leftarrow R \cup \text{ExSet}(I, \varepsilon, \alpha, r)$.
- 6 Return R

13:10 Budgeted Matroid Maximization: Parameterized Viewpoint

Using Lemmas 10 and 14, a representative set of I can be constructed as follows. If the parameters ℓ and k are too high w.r.t. $|I|$, return the trivial representative set E in polynomial time. Otherwise, compute an approximation for $\text{OPT}(I)$, and define the profit classes. Then, the representative set is constructed by finding an exchange set for each profit class. The pseudocode of the algorithm is given in Algorithm 1.

► **Lemma 15.** *Given a BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$, and $0 < \varepsilon < \frac{1}{2}$, Algorithm 1 returns in time $|I|^{O(1)}$ a representative set $R \subseteq E$ of I and ε such that $|R| = \tilde{O}(\ell^{(k-1)\cdot\ell} \cdot k^2 \cdot \varepsilon^{-2})$.*

Proof. Clearly, if $\ell^{(k-1)\cdot\ell} \cdot k^2 \cdot \varepsilon^{-2} > |I|$, then by Step 2 the algorithm runs in time $|I|^{O(1)}$ and returns the trivial representative set E . Thus, we may assume below that $\ell^{(k-1)\cdot\ell} \cdot k^2 \cdot \varepsilon^{-2} \leq |I|$. The running time of Step 3 is $|I|^{O(1)}$ by Lemma 6. Each iteration of the **for** loop in Step 4 can be computed in time $\tilde{O}(\ell^{(k-1)\cdot\ell} \cdot k) \cdot |I|^{O(1)}$, by Lemma 14. Hence, as we have $|D| = |D(I, \varepsilon)|$ iterations of the **for** loop, the running time of the algorithm is bounded by

$$|D| \cdot \tilde{O}(\ell^{(k-1)\cdot\ell} \cdot k) \cdot |I|^{O(1)} \leq (2\ell \cdot k \cdot \varepsilon^{-2} + 1) \cdot \tilde{O}(\ell^{(k-1)\cdot\ell} \cdot k) \cdot |I|^{O(1)} = \tilde{O}(\ell^{(k-1)\cdot\ell+1} \cdot k^2 \cdot \varepsilon^{-2}) \cdot |I|^{O(1)}.$$

The first inequality follows from (4) and (5). As in this case $\ell^{(k-1)\cdot\ell} \cdot k^2 \cdot \varepsilon^{-2} \leq |I|$, we have the desired running time.

For the cardinality of R , note that by Lemma 6 $\text{OPT}(I) \geq \alpha \geq \frac{\text{OPT}(I)}{2\ell}$. Thus, by Lemma 14, for all $r \in D$, $\text{ExSet}(I, \varepsilon, \alpha, r)$ is an exchange set satisfying $|\text{ExSet}(I, \varepsilon, \alpha, r)| = \tilde{O}(\ell^{(k-1)\cdot\ell} \cdot k)$. Then,

$$|R| \leq |D| \cdot \tilde{O}(\ell^{(k-1)\cdot\ell} \cdot k) \leq (2\ell \cdot k \cdot \varepsilon^{-2} + 1) \cdot \tilde{O}(\ell^{(k-1)\cdot\ell} \cdot k) = \tilde{O}(\ell^{(k-1)\cdot\ell+1} \cdot k^2 \cdot \varepsilon^{-2}).$$

The second inequality follows from (4) and (5).

To conclude, we show that R is a representative set. By Lemma 14, for all $r \in D$, it holds that $\text{ExSet}(I, \varepsilon, \alpha, r)$ is an exchange set for I, ε, α , and r . Therefore, $R \cap \mathcal{K}_r(\alpha)$ is an exchange set for I, ε, α , for all $r \in D$. Hence, by Lemma 10, R is a representative set of I and ε . ◀

► **Theorem 3.** *There is an algorithm that given a BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$ and $0 < \varepsilon < \frac{1}{2}$, returns in time $|I|^{O(1)}$ a representative set $R \subseteq E$ of I and ε such that $|R| = \tilde{O}(\ell^{(k-1)\cdot\ell} \cdot k^2 \cdot \varepsilon^{-2})$.*

Proof of Theorem 3. The statement of the lemma follows from Lemma 15. ◀

3 An FPT Approximation Scheme

In this section we use the representative set constructed by Algorithm 1 to obtain an FPAS for BM. For the discussion below, fix a BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$ and an error parameter $0 < \varepsilon < \frac{1}{2}$. Given the representative set R for I and ε output by algorithm `RepSet`, we derive an FPAS by exhaustive enumeration over all solutions of I within R . The pseudocode of our FPAS is given in Algorithm 2.

► **Lemma 16.** *Given a BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$ and $0 < \varepsilon < \frac{1}{2}$, Algorithm 2 returns in time $|I|^{O(1)} \cdot \tilde{O}(\ell^{k^2 \cdot \ell} \cdot k^{2k} \cdot \varepsilon^{-2k})$ a solution for I of profit at least $(1 - 2\varepsilon) \cdot \text{OPT}(I)$.*

We give the proof at the end of this section. We can now prove our main result.

► **Theorem 4.** *For any BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$ and $0 < \varepsilon < \frac{1}{2}$, there is an FPAS whose running time is $|I|^{O(1)} \cdot \tilde{O}(\ell^{k^2 \cdot \ell} \cdot k^{O(k)} \cdot \varepsilon^{-2k})$.*

■ **Algorithm 2** $\text{FPAS}(I = (E, \mathcal{M}, c, p, B, k, \ell), \varepsilon)$.

input : A BM instance I and an error parameter $0 < \varepsilon < \frac{1}{2}$.
output : A solution for I .

- 1 Initialize an empty solution $A \leftarrow \emptyset$.
- 2 Construct $R \leftarrow \text{RepSet}(I, \varepsilon)$.
- 3 **for** $F \subseteq R$ s.t. $|F| \leq k$ and F is a solution of I **do**
- 4 **if** $p(F) > p(A)$ **then**
- 5 Update $A \leftarrow F$
- 6 **Return** A .

Proof of Theorem 4. The statement of the lemma follows from Lemma 16 by using in Algorithm 2 an error parameter $\varepsilon' = \frac{\varepsilon}{2}$. ◀

For the proof of Lemma 16, we use the next auxiliary lemmas.

► **Lemma 17.** *Given a BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$ and $0 < \varepsilon < \frac{1}{2}$, Algorithm 2 returns a solution for I of profit at least $(1 - 2\varepsilon) \cdot \text{OPT}(I)$.*

Proof. By Lemma 15, it holds that $R = \text{RepSet}(I, \varepsilon)$ is a representative set of I and ε . Therefore, by Definition 2, there is a solution S for I such that $S \subseteq R$, and

$$p(S) \geq (1 - 2\varepsilon) \cdot \text{OPT}(I). \quad (9)$$

Since S is a solution for I , it follows that $S \in \mathcal{M}_k$ and therefore $|S| \leq k$. Thus, there is an iteration of Step 3 in which $F = S$, and therefore the set A returned by the algorithm satisfies $p(A) \geq p(S) \geq (1 - 2\varepsilon) \cdot \text{OPT}(I)$. Also, the set A returned by the algorithm must be a solution for I : If $A = \emptyset$ the claim trivially follows since \emptyset is a solution for I . Otherwise, the value of A has been updated in Step 5 of Algorithm 2 to be some set $F \subseteq R$, but this step is reached only if F is a solution for I . ◀

► **Lemma 18.** *Given a BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$ and $0 < \varepsilon < \frac{1}{2}$, the running time of Algorithm 2 is $|I|^{O(1)} \cdot \tilde{O}\left(\ell^{k^2 \cdot \ell} \cdot k^{2k} \cdot \varepsilon^{-2k}\right)$.*

Proof. Let $W' = \{F \subseteq R \mid F \in \mathcal{M}_k, c(F) \leq B\}$ be the solutions considered in Step 3 of Algorithm 2, and let $W = \{F \subseteq R \mid |F| \leq k\}$. Observe that the number of iterations of Step 3 of Algorithm 2 is bounded by $|W|$, since $W' \subseteq W$ and for each $F \in W$ we can verify in polynomial time if $F \in W'$. Thus, it suffices to upper bound W .

By a simple counting argument, we have that

$$\begin{aligned} |W| &\leq (|R| + 1)^k \\ &\leq \tilde{O}\left(\left(\ell^{(k-1) \cdot \ell + 1} \cdot k^2 \cdot \varepsilon^{-2}\right)^k\right) \\ &= \tilde{O}\left(\ell^{k^2 \cdot \ell} \cdot k^{2k} \cdot \varepsilon^{-2k}\right) \end{aligned} \quad (10)$$

The first equality follows from Lemma 15. Hence, by (10), the number of iterations of the **for** loop in Step 3 is bounded by $\tilde{O}\left(\ell^{k^2 \cdot \ell} \cdot k^{2k} \cdot \varepsilon^{-2k}\right)$. In addition, the running time of each iteration is at most $|I|^{O(1)}$. Finally, the running time of the steps outside the **for** loop is $|I|^{O(1)}$, by Lemma 15. Hence, the running time of Algorithm 2 can be bounded by $|I|^{O(1)} \cdot \tilde{O}\left(\ell^{k^2 \cdot \ell} \cdot k^{2k} \cdot \varepsilon^{-2k}\right)$. ◀

Proof of Lemma 16. The proof follows from Lemmas 17 and 18. ◀

4 Hardness Results

In this section we prove Theorem 1 and Theorem 1. In the proof of Theorem 1, we use a reduction from the k -subset sum (KSS) problem. The input for KSS is a set $X = \{x_1, \dots, x_n\}$ of strictly positive integers and two positive integers $T, k > 0$. We need to decide if there is a subset $S \subseteq [n], |S| = k$ such that $\sum_{i \in S} x_i = T$, where the problem is parameterized by k . KSS is known to be W[1]-hard [12].

► **Theorem 1.** BM is W[1]-hard.

Proof of Theorem 1. Let U be a KSS instance with the set of numbers $E = [n]$, target value T , and k . We define the following BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$.

1. \mathcal{M} is a 1-matchoid $\mathcal{M} = \{(E, \mathcal{I})\}$ such that $\mathcal{I} = 2^E$. That is, \mathcal{M} is a single uniform matroid whose independent sets are all possible subsets of E .
2. For any $i \in E = [n]$ define $c(i) = p(i) = x_i + 2 \cdot \sum_{j \in [n]} x_j$.
3. Define the budget as $B = T + 2k \cdot \sum_{j \in [n]} x_j$.

▷ **Claim 19.** If there is a solution for U then there is a solution for I of profit B .

Proof. Let $S \subseteq [n], |S| = k$ such that $\sum_{i \in S} x_i = T$. Then,

$$c(S) = p(S) = \sum_{i \in S} \left(x_i + 2 \cdot \sum_{j \in [n]} x_j \right) = T + |S| \cdot 2 \cdot \sum_{j \in [n]} x_j = T + 2k \cdot \sum_{j \in [n]} x_j = B.$$

By the above, and as $S \in \mathcal{M}_k$, S is also a solution for I of profit exactly B . ◁

▷ **Claim 20.** If there is a solution for I of profit at least B then there is a solution for U .

Proof. Let F be a solution for I of profit at least B . Then, $p(F) = c(F) \leq B$, since F satisfies the budget constraint. As $p(F) \geq B$, we conclude that

$$p(F) = c(F) = B. \tag{11}$$

We now show that F is also a solution for U . First, assume towards contradiction that $|F| \neq k$. If $|F| < k$ then

$$p(F) = \sum_{i \in F} x_i + |F| \cdot 2 \cdot \sum_{j \in [n]} x_j \leq \sum_{i \in F} x_i + (k-1) \cdot 2 \cdot \sum_{j \in [n]} x_j \leq 2k \cdot \sum_{j \in [n]} x_j < B.$$

We reach a contradiction to (11). Since F is a solution for I it holds that $F \in \mathcal{M}_k$; thus, $|F| \leq k$. By the above, $|F| = k$. Therefore,

$$\sum_{i \in F} x_i = c(F) - |F| \cdot 2 \cdot \sum_{j \in [n]} x_j = c(F) - 2k \cdot \sum_{j \in [n]} x_j = B - 2k \cdot \sum_{j \in [n]} x_j = T. \tag{12} \quad \triangleleft$$

By Claims 19 and 20, there is a solution for U if and only if there is a solution for I of profit at least B . Furthermore, the construction of I can be done in polynomial time in the encoding size of U . Hence, an FPT algorithm which finds an optimal solution for I can decide the instance U in FPT time. As KSS is known to be W[1]-hard [12], we conclude that BM is also W[1]-hard. ◀

In the proof of Theorem 5 we use a lower bound on the kernel size of *Perfect ℓ -Dimensional Matching* (ℓ -PDM), due to Dell and Marx [5, 6]. The input for the problem consists of the finite sets U_1, \dots, U_ℓ and $E \subseteq U_1 \times \dots \times U_\ell$. Also, we have an ℓ -dimensional matching constraint (E, \mathcal{I}) to which we refer as the associated set system of the instance (i.e., \mathcal{I} contains all subsets $S \subseteq E$ such that for any two distinct tuples $(e_1, \dots, e_\ell), (f_1, \dots, f_\ell) \in S$ and every $i \in [\ell]$ it holds that $e_i \neq f_i$). The instance is associated also with the parameter $k = \frac{n}{\ell}$, where $n = \sum_{j=1}^{\ell} |U_j|$. We refer to $|E|$ as the *number of tuples* in the instance. The objective is to find $S \in \mathcal{I}$ such that $|S| = k$. Let $J = (U_1, \dots, U_\ell, E)$ denote an instance of ℓ -PDM. We say J is a “yes” instance if such a set S exists; otherwise, J is a “no” instance. Observe that the parameter k is set such that if $S \in \mathcal{I}$ and $|S| = k$ then every element in $U_1 \cup \dots \cup U_\ell$ appears in exactly one of the tuples in S .

► **Lemma 21** (Theorem 1.2 cf. [6]). *Let $\ell \geq 3$ and $\varepsilon > 0$. If $\text{coNP} \not\subseteq \text{NP/poly}$ then ℓ -PDM does not have a kernel in which the number of tuples is $O(k^{\ell-\varepsilon})$.*

► **Theorem 5.** *For any function $f : \mathbb{N} \rightarrow \mathbb{N}$, and $c_1, c_2 \in \mathbb{R}$ such that $c_2 - c_1 < 0$, there is no algorithm which finds for a given BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$ and $0 < \varepsilon < \frac{1}{2}$ a representative set of size $O(f(\ell) \cdot k^{\ell-c_1} \cdot \frac{1}{\varepsilon^{c_2}})$ of I and ε in time $|I|^{O(1)}$, unless $\text{coNP} \subseteq \text{NP/poly}$.*

Proof of Theorem 5. Assume $\text{coNP} \not\subseteq \text{NP/poly}$. Furthermore, assume towards a contradiction that there is a function $f : \mathbb{N} \rightarrow \mathbb{N}$, constants c_1, c_2 , where $c_2 - c_1 < 0$, and an algorithm \mathcal{A} that, given a BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$ and $0 < \varepsilon < \frac{1}{2}$, finds in time $|I|^{O(1)}$ a representative set of I and ε of size $O(f(\ell) \cdot k^{\ell-c_1} \cdot \frac{1}{\varepsilon^{c_2}})$. We use \mathcal{A} to construct a kernel for 3-PDM.

Consider the following kernelization algorithm for 3-PDM. Let $J = (U_1, U_2, U_3, E)$ be the 3-PDM input instance. Define $n = |U_1| + |U_2| + |U_3|$, $\ell = 3$, and $k = \frac{n}{\ell}$. Furthermore, let (E, \mathcal{I}) be the set system associated with the instance, and let \mathcal{M} be an ℓ -matchoid representing the set system (E, \mathcal{I}) . Run \mathcal{A} on the BM instance $I = (E, \mathcal{M}, p, B, k, \ell)$ with $\varepsilon = \frac{1}{3k}$, where $c(e) = p(e) = 1$ for all $e \in E$ and $B = k$. Let $R \subseteq E$ be the output of \mathcal{A} . Return the 3-PDM instance $J' = (U_1, U_2, U_3, R)$.

Since \mathcal{A} runs in polynomial time, the above algorithm runs in polynomial time as well. Moreover, as $k = \frac{n}{3}$ and $R \subseteq E$, it follows that the returned instance can be encoded using $O(k^4)$ bits. Let (R, \mathcal{I}') be the set system associated with J' . Since $R \subseteq E$, it follows that $\mathcal{I}' \subseteq \mathcal{I}$. Hence, if there is $S \in \mathcal{I}'$ such that $|S| = k$, then $S \in \mathcal{I}$ as well. That is, if J' is a “yes” instance, so is J .

For the other direction, assume that J is a “yes” instance. That is, there is $S \in \mathcal{I}$ such that $|S| = k$. Then S is a solution for the BM instance I (observe that $c(S) = |S| = k = B$). Therefore, as R is a representative set of I and $\varepsilon = \frac{1}{3k}$, there is a solution T for I such that $T \subseteq R$, and

$$p(T) \geq (1 - 2\varepsilon) \cdot \text{OPT}(I) \geq (1 - 2\varepsilon) \cdot p(S) = \left(1 - \frac{2}{3k}\right) \cdot p(S) = \left(1 - \frac{2}{3k}\right) \cdot k = k - \frac{2}{3}.$$

Since the profits are integral we have that $|T| = p(T) \geq k$. Furthermore $|T| \leq k$ (since T is a solution for I), and thus $|T| = k$. Since $T \in \mathcal{I}$ (as T is a solution for I) and $T \subseteq R$, it trivially holds that $T \in \mathcal{I}'$. That is, $T \in \mathcal{I}'$ and $|T| = k$. Hence, J' is a “yes” instance. We have showed that the above procedure is indeed a kernelization for 3-PDM.

Now, consider the size of R . Since \mathcal{A} returns a representative set of size $O(f(\ell) \cdot k^{\ell-c_1} \cdot \frac{1}{\varepsilon^{c_2}})$ it follows that

$$|R| = O(f(3) \cdot k^{3-c_1} \cdot (3k)^{c_2}) = O(k^{3-c_1+c_2}).$$

13:14 Budgeted Matroid Maximization: Parameterized Viewpoint

As $c_2 - c_1 < 0$, we have a contradiction to Lemma 21. Thus, for any function $f : \mathbb{N} \rightarrow \mathbb{N}$ and constants c_1, c_2 satisfying $c_2 - c_1 < 0$, there is no algorithm which finds for a given BM instance $I = (E, \mathcal{M}, c, p, B, k, \ell)$ and $0 < \varepsilon < \frac{1}{2}$ a representative set of I and ε of size $O(f(\ell) \cdot k^{\ell-c_1} \cdot \frac{1}{\varepsilon^{c_2}})$ in time $|I|^{O(1)}$. \blacktriangleleft

5 A Polynomial-time $\frac{1}{2\ell}$ -Approximation for BM

In this section we prove Lemma 6. The proof combines an existing approximation algorithm for the unbudgeted version of BM [23, 25] with the Lagrangian relaxation technique of [29]. As the results in [23, 25] are presented in the context of ℓ -extendible set systems, we first define these systems and use a simple argument to show that such systems are generalizations of matchoids. We refer the reader to [13] for further details about ℓ -extendible systems.

► **Definition 22.** *Given a finite set E , $\mathcal{I} \subseteq 2^E$, and $\ell \in \mathbb{N}$, we say that (E, \mathcal{I}) is an ℓ -extendible system if for every $S \in \mathcal{I}$ and $e \in E \setminus S$ there is $T \subseteq S$, where $|T| \leq \ell$, such that $(S \setminus T) \cup \{e\} \in \mathcal{I}$.*

The next lemma shows that an ℓ -matchoid is in fact an ℓ -extendible set system.

► **Lemma 23.** *For any $\ell \in \mathbb{N}_{>0}$ and an ℓ -Matchoid $\mathcal{M} = \{M_i = (E_i, \mathcal{I}_i)\}_{i \in [s]}$ on a set E , it holds that $(E, \mathcal{I}(\mathcal{M}))$ is an ℓ -extendible set system.*

Proof. Let $S \in \mathcal{I}(\mathcal{M})$ and $e \in E \setminus S$. As \mathcal{M} is an ℓ -matchoid, there is $H \subseteq [s]$ of cardinality $|H| \leq \ell$ such that for all $i \in [s] \setminus H$ it holds that $e \notin E_i$ and for all $i \in H$ it holds that $e \in E_i$. Since for all $i \in H$ it holds that (E_i, \mathcal{I}_i) is a matroid, either $(S \cap E_i) \cup \{e\} \in \mathcal{I}_i$, or there is $a_i \in S \cap E_i$ such that $((S \cap E_i) \setminus \{a_i\}) \cup \{e\} \in \mathcal{I}_i$ (this follows by repeatedly adding elements from $S \cap E_i$ to $\{e\}$ using the exchange property of the matroid (E_i, \mathcal{I}_i)). Let $L = \{i \in H \mid (S \cap E_i) \cup \{e\} \notin \mathcal{I}_i\}$. Then, there are $|L|$ elements $T = \{a_i\}_{i \in L}$ such that for all $i \in L$ it holds that $((S \cap E_i) \setminus \{a_i\}) \cup \{e\} \in \mathcal{I}_i$ and for all $i \in H \setminus L$ it holds that $(S \cap E_i) \cup \{e\} \in \mathcal{I}_i$. Thus, it follows that $(S \setminus T) \cup \{e\} \in \mathcal{I}(\mathcal{M})$ by the definition of a matchoid. Since $|T| = |L| \leq |H| \leq \ell$, we have the statement of the lemma. \blacktriangleleft

Proof of Lemma 6. Consider the BM problem with no budget constraint (equivalently, $c(E) \leq B$) that we call the *maximum weight matchoid maximization (MWM)* problem. By Lemma 23, MWM is a special case of the *maximum weight ℓ -extendible system maximization* problem, which admits $\frac{1}{\ell}$ -approximation [23, 25].⁵ Therefore, using Theorem 3.1 in [29], we have the following. There is an algorithm that, given some $\varepsilon > 0$, returns a solution for the BM instance I of profit at least $\left(\frac{1}{\frac{1}{\ell}+1} - \varepsilon\right) \cdot \text{OPT}(I)$, and whose running time is $|I|^{O(1)} \cdot O(\log(\varepsilon^{-1}))$. Now, we can set $\varepsilon = \frac{1}{\frac{1}{\ell}+1} - \frac{1}{2\ell}$; then, the above algorithm has a running time $|I|^{O(1)}$, since ε^{-1} is polynomial in ℓ and $\ell \leq |I|$. Moreover, the algorithm returns a solution S for I , such that

$$\text{OPT}(I) \geq p(S) \geq \left(\frac{1}{\frac{1}{\ell}+1} - \varepsilon\right) \cdot \text{OPT}(I) = \frac{1}{2\ell} \cdot \text{OPT}(I).$$

To conclude, we define the algorithm **ApproxBM** which returns $\alpha = p(S)$. By the above discussion, $\text{OPT}(I) \geq \alpha \geq \frac{\text{OPT}(I)}{2\ell}$, and the running time of **ApproxBM** is $|I|^{O(1)}$. \blacktriangleleft

⁵ The algorithm of [23] can be applied also in the more general setting of ℓ -systems. For more details on such set systems, see, e.g., [13].

6 Discussion

In this paper we present an FPT-approximation scheme (FPAS) for the budgeted ℓ -matchoid problem (BM). As special cases, this yields FPAS for the budgeted ℓ -dimensional matching problem (BDM) and the budgeted ℓ -matroid intersection problem (BMI). While the unbudgeted version of BM has been studied earlier from parameterized viewpoint, the budgeted version is studied here for the first time.

We show that BM parameterized by the solution size is $W[1]$ -hard already with a degenerate matroid constraint (Theorem 1); thus, an exact FPT time algorithm is unlikely to exist. Furthermore, the special case of unbudgeted ℓ -dimensional matching problem is APX-hard, already for $\ell = 3$, implying that PTAS for this problem is also unlikely to exist. These hardness results motivated the development of an FPT-approximation scheme for BM.

Our FPAS relies on the notion of representative set – a small cardinality subset of the ground set of the original instance which preserves the optimum value up to a small factor. We note that representative sets are not *lossy kernels* [31] as BM is defined in an oracle model; thus, the definitions of kernels or lossy kernels do not apply to our problem. Nevertheless, for some variants of BM in which the input is given explicitly (for instance, this is possible for BDM) our construction of representative sets can be used to obtain an approximate kernelization scheme.

Our results also include a lower bound on the minimum possible size of a representative set for BM which can be computed in polynomial time (Theorem 5). The lower bound is based on the special case of the budgeted ℓ -dimensional matching problem (BDM). We note that there is a significant gap between the size of the representative sets found in this paper and the lower bound. This suggests the following questions for future work.

- Is there a representative set for the special case of BDM whose size matches the lower bound given in Theorem 5?
- Can the generic structure of ℓ -matchoids be used to derive an improved lower bound on the size of a representative set for general BM instances?

The budgeted ℓ -matchoid problem can be naturally generalized to the d -budgeted ℓ -matchoid problem (d -BM). In the d -budgeted version, both the costs and the budget are replaced by d -dimensional vectors, for some constant $d \geq 2$. A subset of elements is feasible if it is an independent set of the ℓ -matchoid, and the total cost of the elements in each dimension is bounded by the budget in this dimension. The problem is a generalization of the d -dimensional knapsack problem (d -KP), the special case of d -BM in which the feasible sets of the matchoid are all subsets of E . A PTAS for d -KP was first given in [17], and the existence of an *efficient* polynomial time approximation scheme was ruled out in [28]. PTASs for the special cases of d -BM in which the matchoid is a single matroid, matroid intersection or a matching constraint were given in [3, 20]. It is likely that the lower bound in [28] can be used also to rule out the existence of an FPAS for d -BM. However, the question whether d -BM admits a $(1 - \varepsilon)$ -approximation in time $O(f(k + \ell) \cdot n^{g(\varepsilon)})$, for some functions f and g , remains open.

References

- 1 André Berger, Vincenzo Bonifaci, Fabrizio Grandoni, and Guido Schäfer. Budgeted matching and budgeted matroid intersection via the gasoline puzzle. *Mathematical Programming*, 128(1):355–372, 2011.
- 2 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *Journal of Computer and System Sciences*, 87:119–139, 2017.
- 3 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Multi-budgeted matchings and matroid intersection via dependent rounding. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1080–1097. SIAM, 2011.
- 4 Jianer Chen, Qilong Feng, Yang Liu, Songjian Lu, and Jianxin Wang. Improved deterministic algorithms for weighted matching and packing problems. *Theoretical computer science*, 412(23):2503–2512, 2011.
- 5 Holger Dell and Dániel Marx. Kernelization of packing problems. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 68–81. SIAM, 2012.
- 6 Holger Dell and Dániel Marx. Kernelization of packing problems. *arXiv preprint arXiv:1812.03155*, 2018.
- 7 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. Budgeted matroid maximization: a parameterized viewpoint. *arXiv preprint arXiv:2307.04173*, 2023.
- 8 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An FPTAS for budgeted matching and budgeted matroid intersection via representative sets. In *50th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2023.
- 9 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An FPTAS for budgeted matroid independent set. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 69–83, 2023.
- 10 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An FPTAS for budgeted laminar matroid independent set. *Operations Research Letters*, 51(6):632–637, 2023. doi:10.1016/j.orl.2023.10.005.
- 11 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. Tight lower bounds for weighted matroid problems. *arXiv preprint arXiv:2307.07773*, 2023.
- 12 Rod G Downey and Michael R Fellows. Fixed-parameter tractability and completeness ii: On completeness for W[1]. *Theoretical Computer Science*, 141(1-2):109–131, 1995.
- 13 Moran Feldman, Joseph Naor, Roy Schwartz, and Justin Ward. Improved approximations for k-exchange systems. In *Algorithms–ESA 2011: 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings 19*, pages 784–798. Springer, 2011.
- 14 Andreas Emil Feldmann, Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020.
- 15 Fedor V Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *Journal of the ACM (JACM)*, 63(4):1–60, 2016.
- 16 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.
- 17 Alan M Frieze, Michael RB Clarke, et al. Approximation algorithms for the m-dimensional 0-1 knapsack problem: worst-case and probabilistic analyses. *European Journal of Operational Research*, 15(1):100–109, 1984.
- 18 Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.
- 19 Prachi Goyal, Neeldhara Misra, Fahad Panolan, and Meirav Zehavi. Deterministic algorithms for matching and packing problems based on representative sets. *SIAM Journal on Discrete Mathematics*, 29(4):1815–1836, 2015.
- 20 Fabrizio Grandoni and Rico Zenklusen. Approximation schemes for multi-budgeted independence systems. In *European Symposium on Algorithms*, pages 536–548. Springer, 2010.

- 21 Chien-Chung Huang and Justin Ward. FPT-algorithms for the ℓ -matchoid problem with a coverage objective. *arXiv preprint arXiv:2011.06268*, 2020.
- 22 Chien-Chung Huang and Justin Ward. FPT-algorithms for the ℓ -matchoid problem with a coverage objective. *SIAM Journal on Discrete Mathematics*, 2023.
- 23 Th Jenkyns. The efficacy of the "greedy" algorithm. In *Proc. 7th Southeastern Conf. on Combinatorics, Graph Theory and Computing*, pages 341–350, 1976.
- 24 Per M Jensen and Bernhard Korte. Complexity of matroid property algorithms. *SIAM Journal on Computing*, 11(1):184–190, 1982.
- 25 Stasys Jukna. *Extremal combinatorics: with applications in computer science*, volume 571. Springer, 2011.
- 26 Viggo Kann. Maximum bounded 3-dimensional matching is max snp-complete. *Information Processing Letters*, 37(1):27–35, 1991.
- 27 Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. In *Automata, Languages and Programming: 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I 36*, pages 653–664. Springer, 2009.
- 28 Ariel Kulik and Hadas Shachnai. There is no EPTAS for two-dimensional knapsack. *Information Processing Letters*, 110(16):707–710, 2010.
- 29 Ariel Kulik, Hadas Shachnai, and Gal Tamir. On lagrangian relaxation for constrained maximization and reoptimization problems. *Discrete Applied Mathematics*, 296:164–178, 2021.
- 30 Daniel Lokshantov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. *ACM Transactions on Algorithms (TALG)*, 14(2):1–20, 2018.
- 31 Daniel Lokshantov, Fahad Panolan, MS Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 224–237, 2017.
- 32 László Lovász. Matroid matching and some applications. *Journal of Combinatorial Theory, Series B*, 28(2):208–236, 1980.
- 33 Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.
- 34 Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.
- 35 Dániel Marx. A parameterized view on matroid optimization problems. *Theoretical Computer Science*, 410(44):4471–4479, 2009.
- 36 Ram Ravi and Michel X Goemans. The constrained minimum spanning tree problem. In *Scandinavian Workshop on Algorithm Theory*, pages 66–75. Springer, 1996.
- 37 Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

Computing Complexity Measures of Degenerate Graphs

Pål Grønås Drange ✉ 

University of Bergen, Norway

Patrick Greaves ✉

Birkbeck, University of London, UK

Irene Muzi ✉ 

Birkbeck, University of London, UK

Felix Reidl ✉ 

Birkbeck, University of London, UK

Abstract

We show that the VC-dimension of a graph can be computed in time $n^{\lceil \log d+1 \rceil} d^{O(d)}$, where d is the degeneracy of the input graph. The core idea of our algorithm is a data structure to efficiently query the number of vertices that see a specific subset of vertices inside of a (small) query set. The construction of this data structure takes time $O(d2^d n)$, afterwards queries can be computed efficiently using fast Möbius inversion.

This data structure turns out to be useful for a range of tasks, especially for finding bipartite patterns in degenerate graphs, and we outline an efficient algorithm for counting the number of times specific patterns occur in a graph. The largest factor in the running time of this algorithm is $O(n^c)$, where c is a parameter of the pattern we call its *left covering number*.

Concrete applications of this algorithm include counting the number of (non-induced) bicliques in linear time, the number of co-matchings in quadratic time, as well as a constant-factor approximation of the ladder index in linear time.

Finally, we supplement our theoretical results with several implementations and run experiments on more than 200 real-world datasets – the largest of which has 8 million edges – where we obtain interesting insights into the VC-dimension of real-world networks.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases vc-dimension, datastructure, degeneracy, enumerating

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.14

Related Version *Full Version*: <https://arxiv.org/abs/2308.08868>

Supplementary Material

Software (Source code): <https://github.com/microgravitas/mantis-shrimp>
archived at [swh:1:dir:74c67c5dfffc03a1c1383be7947d7def7cc0d4798](https://swh.1:dir:74c67c5dfffc03a1c1383be7947d7def7cc0d4798)

Funding *Pål Grønås Drange*: Supported by the Research council of Norway, grant number 329745: *Machine Teaching for Explainable AI*.



© Pål Grønås Drange, Patrick Greaves, Irene Muzi, and Felix Reidl;
licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 14; pp. 14:1–14:21

Leibniz International Proceedings in Informatics



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

1 Introduction

Our work began with the simple question: What is the Vapnik–Chervonenkis (VC) dimension of real-world networks? That is, what is the largest vertex set X such that every subset $X' \subseteq X$ is the neighbourhood (when restricted to X) of some vertex in the network?

This parameter, developed in the context of learning theory, happens to be extremely useful in the theory of sparse graphs (e.g. [11]) and is one possible method of capturing the “complexity” of an object. It is therefore a natural statistic to consider when a) trying to categorise networks and b) identifying structural properties that can be leveraged to design efficient algorithms.

As the best-known general algorithm to compute the VC-dimension of a graph takes $O(n^{\log n})$ time, and in fact the problem being LOGNP-complete [16], we investigated whether a better algorithm is possible if we assume our input graph to be sparse, more precisely, to be d -degenerate. This choice is motivated by the observation that the degeneracy for most real-world networks is small (see e.g. [7] and our results in the full version [9]).

Our first achievement is an algorithm that computes the VC-dimension of a d -degenerate graph in time $O(n^{\lceil \log d+1 \rceil} d^{O(d)})$. A core concept is a novel data structure which enables us to efficiently query the size of the intersection of several neighbourhoods for a small set of vertices, described in Section 3, which we use to quickly determine whether a given candidate set is shattered by its neighbours.

But the general idea of this algorithm can be generalised to other bipartite “patterns” like bicliques, co-matchings, and ladders (defined in Section 2.2). These objects are also closely related to notions of “complexity” of graphs. They appear, for example, in the study of graph width measures [10] and algorithm design for sparse classes [13] (see also there for connections to stability theory). Our general pattern-finding algorithm presented in Section 3 can count bicliques in linear time, co-matchings in quadratic time and find partial ladders in linear time, see Section 4 for these and further results.

Dense structures like cliques or bicliques are famously important in the analysis of networks, and we suggest that co-matchings and ladders might be of similar interest – but without a program to compute them, we cannot hope for these statistics to be trialled in practice. We therefore implemented algorithms to compute the VC-dimension, ladder index, maximum biclique¹ and maximum co-matching of a graph. To establish their practicality, we ran these four algorithms on 206 real-world networks from various sources, see Section 5. The VC-dimension algorithm in our experiments terminated within 10 minutes on networks with up to $\sim 33\text{K}$ vertices, the other three on networks up to $\sim 93\text{K}$ vertices. This is already squarely in the region of “practical” for certain types of networks and we believe that with further engineering – in particular to improve space efficiency – our implementation can be used to compute these statistics on much larger networks.

Prior work. We briefly mention a few relevant previous articles on the subject. Eppstein, Löffler, and Strash [12] gave an algorithm for enumerating maximal cliques in d -degenerate graphs in $O(dn3^{d/3})$ time, i.e., fixed-parameter tractable time when parameterized by the degeneracy. They also give experimental results showing that their algorithm works well on large real-world networks. Bera, Pashanasangi, and Seshadhri [1], extending the classic result

¹ There are probably faster programs to compute bicliques in practice, we compute this statistic here as a baseline.

by Chiba and Nishizeki [6], show that for all patterns H of size less than six, we can count the number of appearances of H in a d -degenerate graph G in time $O(m \cdot d^{k-2})$, where m is the number of edges in G and k is the number of vertices in H . Recently, Bressan, and Roth [3] gave algorithms for counting copies of a graph H in a d -degenerate graph G in time $f(d, k) \cdot n^{\mathbf{im}(H)} \log n$, for some function f , where k again is the number of vertices in H , n the number of vertices in G , and $\mathbf{im}(H)$ is the size of a largest induced matching in H .

2 Preliminaries

For an integer k , we use $[k]$ as a short-hand for the set $\{0, 1, 2, \dots, k-1\}$. We use black-board bold letters like \mathbb{X} to denote sets X associated with a total order $<_{\mathbb{X}}$. The *index function* $\iota_{\mathbb{X}}: X \rightarrow \mathbb{N}$ maps elements of X to their corresponding position in \mathbb{X} . We extend this function to sets via $\iota_{\mathbb{X}}(S) = \{\iota_{\mathbb{X}}(s) \mid s \in S\}$. For any integer $i \in [|\mathbb{X}|]$ we write $\mathbb{X}[i]$ to mean the i th element in the ordered set. An *index set* I for \mathbb{X} is simply a subset of $[|\mathbb{X}|]$ and we extend the index notation to sets via $\mathbb{X}[I] := \{\mathbb{X}[i] \mid i \in I\}$. We write $\pi(H)$ for the set of all permutations of H .

For a graph G we use $V(G)$ and $E(G)$ to refer to its vertex- and edge-set, respectively. We used the short hands $|G| := |V(G)|$ and $\|G\| := |E(G)|$.

An *ordered graph* is a pair $\mathbb{G} = (G, <)$ where G is a graph and $<$ a total ordering of $V(G)$. We write $<_{\mathbb{G}}$ to denote the ordering for a given ordered graph and extend this notation to the derived relations $\leq_{\mathbb{G}}$, $>_{\mathbb{G}}$, $\geq_{\mathbb{G}}$.

We use the same notations for graphs and ordered graphs, additionally we write $N^-(u) := \{v \in N(u) \mid v <_{\mathbb{G}} u\}$ for the *left neighbourhood* and $N^+(u) := \{v \in N(u) \mid v >_{\mathbb{G}} u\}$ for the *right neighbourhood* of a vertex $u \in \mathbb{G}$. We further use $d_{\mathbb{G}}^-(u)$ and $d_{\mathbb{G}}^+(u)$ for the left and right degree, as well as $\Delta^-(\mathbb{G}) := \max_{u \in \mathbb{G}} d_{\mathbb{G}}^-(u)$ and $\Delta^+(\mathbb{G}) := \max_{u \in \mathbb{G}} d_{\mathbb{G}}^+(u)$. We omit the graphs in the subscripts if clear from the context.

A graph G is d -degenerate if there exists an ordering \mathbb{G} such that $\Delta^-(\mathbb{G}) \leq d$. An equivalent definition is that a graph is d -degenerate if every subgraph has a vertex of degree at most d . The number of edges in a d -degenerate graph is bounded by dn and many important sparse graph classes – bounded treewidth, planar graphs, graphs excluding a minor – have finite degeneracy. The degeneracy ordering of a graph can be computed in time $O(n + m)$ [15], and $O(dn)$ for d -degenerate graphs.

Let $\mathcal{F} \subseteq 2^U$ be a set family over U . We define the intersection of a set family with set $X \subseteq U$ as $\mathcal{F} \cap X := \{F \cap X \mid F \in \mathcal{F}\}$. A set $X \subseteq U$ is then *shattered* by \mathcal{F} if $\mathcal{F} \cap X = 2^X$. The *graph representation* of a set family \mathcal{F} is the bipartite graph $G(\mathcal{F}) = (\mathcal{F}, U, E)$ where for each $F \in \mathcal{F}$ and $x \in U$ we have the edge $Fx \in E$ iff $x \in F$. In the other direction, we define for a graph G its *neighbourhood set system* $\mathcal{F}(G) := \{N(v) \mid v \in G\}$.

The Vapnik–Chervonenkis dimension (VC-dimension) of a set family $\mathcal{F} \subseteq 2^U$ is the size of the largest set in U that is shattered by \mathcal{F} and we write this quantity as $\mathbf{vc}(\mathcal{F})$. The VC-dimension of a graph G is defined as the VC-dimension of its neighbourhood set system, i.e. $\mathbf{vc}(G) := \mathbf{vc}(\mathcal{F}(G))$.

2.1 Set dictionaries

In the following we will make heavy use of data structures that model functions of the form $f: 2^U \rightarrow \mathbb{Z}$ for some universe U . Since the arguments in our use-case are assumed to be small, we use prefix-tries [17] in our theoretical analysis (see notes on practical implementations below):

14:4 Computing Complexity Measures of Degenerate Graphs

► **Definition 1** (Subset dictionary). Let U be a set and let \mathbb{U} be an arbitrary total order of U . A subset dictionary D over U associates a key $X \subseteq U$ with an integer $D[X]$ by storing the sequence \mathbb{X} of X under $<_{\mathbb{U}}$ in a prefix trie.

Accordingly, insertion/update/deletion of a value for a key X takes time $O(|X|)$ if we can assume the key X to be present in some canonical order. Our algorithms all work on graphs imbued with a (degeneracy) ordering and we will sort the left-neighbourhood $N^-(\bullet)$ of each vertex according to this global ordering, which we will simply call “sorting the left-neighbourhoods” for brevity. Subsets of these left-neighbourhoods are assumed to inherit this ordering, which covers all operations that we will need in our algorithms, which in conclusion means that we can assume that all sets used as keys in subset dictionaries have a canonical ordering.

Unless otherwise noted, we will use the convention that $D[X] = 0$ for all keys X that have not been inserted into D .

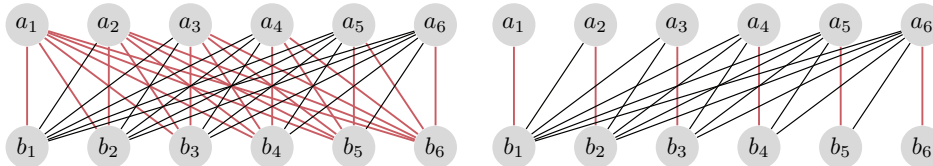
2.2 Bipartite patterns and left-covers

► **Definition 2** (Pattern). A pattern H is a complete graph whose edges are partitioned into sets B , R , and W (black, red and white). We say that a graph G contains H (or H appears in G) if there exists a vertex set $S \subseteq V(G)$ and a bijection $\phi: V(H) \rightarrow S$ such that $uv \in B \implies \phi(u)\phi(v) \in E(G)$ and $uv \in R \implies \phi(u)\phi(v) \notin E(G)$.

We say that a pattern H is bipartite if the vertex set of H can be partitioned into two sets X, Y such that all edges inside of X and inside of Y are white.

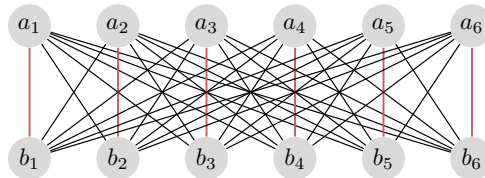
For a vertex $v \in V(H)$ we write $N(v)$ to denote its neighbours according to the black edge relation only. An *ordered* pattern \mathbb{H} is a pattern whose vertex set comes with a linear order $<_{\mathbb{H}}$. Given a vertex $v \in \mathbb{H}$, we write $N^-(u) := \{v \in N(u) \mid v <_{\mathbb{H}} u\}$.

A *ladder* (sometimes called a chain graph) L_n of size n is a bipartite pattern defined on two vertex sequences $A = (a_i)_{i \in [n]}$ and $B = (b_i)_{i \in [n]}$, where $a_i b_j \in B$ if $i > j$ and $a_i b_j \in R$ otherwise. Note that for any $1 \leq l \leq r \leq n$ the subgraph induced by the sequences $(a_i)_{i \in [l, r]}$ and $(b_i)_{i \in [l, r]}$ induces a ladder. A *semi-ladder* \tilde{L}_n has the same black edges, but only the edges $a_i b_i$, $i \in [n]$ are red. All the remaining edges are white:

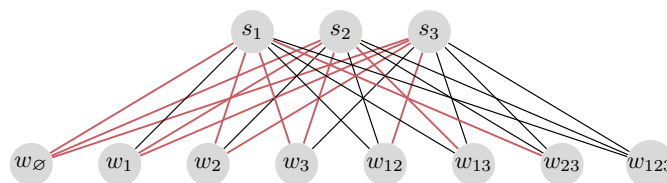


The *Ladder index* of a graph G is the largest n such that G contains the pattern L_n .

A *co-matching* \bar{M}_n (also called *crown*) has black edges $a_i b_j$ for $i \neq j$ and red edges $a_i b_i$ for $i \in [n]$.



Finally, the *shattered* pattern U_n of size n has a side S (the *shattered* set) of size n and a side W (the *witness* set) of size 2^n . We index the vertices of W by subsets $I \subseteq S$, then the vertex w_I has black edges into I and red edges into $S \setminus I$:



► **Definition 3** (Left-cover, left-covering number). Given an ordered bipartite pattern \mathbb{H} with bipartition (X, Y) , a left-cover is a set of vertices $C \subseteq V(\mathbb{H})$ such that either $X \subseteq N^-(C) \cup C$ or $Y \subseteq N^-(C) \cup C$. The left-covering number $\text{lc}(\mathbb{H})$ is the minimum size of a left cover of \mathbb{H} . For an (unordered) pattern H we define its left-covering number as

$$\text{lc}(H) := \max_{\mathbb{H} \in \pi(H)} \text{lc}(\mathbb{H}).$$

Note that we include the covering set C itself in the cover, this is necessary since for a given ordering of a pattern some vertices might not have right neighbours and can therefore not be covered by left neighbourhoods.

The left-covering number of a pattern is the first important measure that will influence the running time of the main algorithm presented later. The second important measure relates to the number of non-isomorphic “half”-ordered patterns we can obtain from a bipartite pattern, that is, how many distinct objects we find by ordering one partition. A useful tool to concretise this notion is the following function:

► **Definition 4** (Signature). Let H be a bipartite pattern with bipartition (X, Y) and let Z be an ordering of $Z \in \{X, Y\}$. Then the signature $\sigma_Z(H)$ is defined as the multiset

$$\sigma_Z(H) := \{\{\iota_Z(N(u)) \mid u \in (X \cup Y) \setminus Z\}\}.$$

For orderings $Z, Z' \in \pi(Z)$ we define the equivalence relation

$$Z \sim_H Z' \iff \sigma_Z(H) = \sigma_{Z'}(H).$$

► **Definition 5** (Half-ordering asymmetry). Given a bipartite pattern H with bipartition (X, Y) and a partite set $Z \in \{X, Y\}$, we define the half-ordering asymmetry $\text{hoa}(H, Z)$ as the number of equivalence classes under the \sim_H relation

$$\text{hoa}(H, Z) := |\pi(Z) / \sim_H|.$$

We further define the half-ordering asymmetry of H as

$$\text{hoa}(H) := \max\{\text{hoa}(H, X), \text{hoa}(H, Y)\}.$$

Alternatively, $\text{hoa}(H, Z) := |\{\sigma_Z(H) \mid Z \in \pi(Z)\}|$.

3 A general pattern-finding algorithm

We first describe a general-purpose algorithm for finding patterns in degenerate graphs. Afterwards, we will describe more specialised algorithms using similar ideas to find specific patterns.

► **Theorem 6.** Let G be a d -degenerate graph and let H be a bipartite pattern with bipartition (X, Y) where $|X| \geq |Y|$. Then after a preprocessing time of $O(|X|^{\text{lc}(H)} |H|! + d2^d n)$, we can in time $O(n^{\text{lc}(H)} (4d \text{lc}(H))^{|X|} d |X|^3 \text{hoa}(H))$ count how often H appears in G .

14:6 Computing Complexity Measures of Degenerate Graphs

The main ingredient of our algorithm will be the following data structure:

► **Theorem 7.** *Let \mathbb{G} be an ordered graph on n vertices with degeneracy d . After a preprocessing time of $O(d2^d n)$, we can, for any given $S \subseteq V(G)$, compute a subset dictionary Q_S in time $O(|S|2^{|S|} + d|S|^2)$ which for any $X \subseteq S \subseteq V(G)$ answers the query*

$$Q_S[X] := |\{v \in G \mid S \cap N(v) = X\}|$$

in time $O(|X|)$.

► **Lemma 8.** *Let \mathbb{G} be an ordered graph with degeneracy d . Then in time $O(d2^d n)$ we can compute a subset dictionary R over $V(G)$ which for any $X \subseteq V(G)$ answers the query*

$$R[X] := |\{v \in G \mid X \subseteq N^-(v)\}|$$

in time $O(|X|)$.

Proof. Given \mathbb{G} as input, we compute R as follows:

Initialize R as an empty trie storing integers;

for $u \in \mathbb{G}$ **do**

for $X \subseteq N^-(u)$ **do**
 | $R[X] \leftarrow R[X] + 1$ // Non-existing keys are treated as zero

return R ;

Note that every update of the data structure with key X takes time $O(|X|)$, since $|X| \leq d$ it follows that the total initialisation time is bounded by $O(d2^d n)$. ◀

► **Lemma 9.** *Let \mathbb{G} be an ordered graph with degeneracy d and let $S \subseteq V(G)$. If we assume the subset dictionary R of Lemma 8 is given, we can construct in time $O(|S|2^{|S|} + d|S|^2)$ a subset dictionary Q_S over S which for $X \subseteq S$ answer the query*

$$Q_S[X] := |\{v \in G \mid S \cap N(v) = X\}|$$

in time $O(|X|)$.

Proof. We first construct an auxiliary subset dictionary \hat{Q} which for $X \subseteq S$ answers the query

$$\hat{Q}_S[X] := |\{v \in G \mid S \cap N^-(v) = X\}|$$

in time $O(|X|)$. We first prove the following claim which implies that \hat{Q}_S is the (upwards) Möbius inversion of R over S and hence can be computed in time $O(|S|2^{|S|})$ using Yate's algorithm [18, 14, 2].

▷ **Claim 10.** $|\{v \in G \mid S \cap N^-(v) = X\}| = \sum_{X \subseteq Y \subseteq S} (-1)^{|Y \setminus X|} R[Y]$,

Proof. First consider $v \not\geq_{\mathbb{G}} X$. Then X cannot be contained in $N^-(v)$ and therefore v does not contribute to the left-hand side. Note that v is not counted by $R[Y]$ for any $Y \supseteq X$, therefore v does not contribute to the right-hand side.

Consider therefore $v \geq_{\mathbb{G}} X$. First, assume that $S \cap N^-(v) = X$ and therefore v contributes to the left-hand side. Then v is counted on the right-hand side exactly once by the term $R[X]$ which has a positive sign.

Consider now v with $S \cap N^-(v) \neq X$. If $X \not\subseteq N^-(v)$, then v does not contribute to the left-hand side and it is not counted by any term $R[Y]$, $Y \supseteq X$ on the right-hand side. We are therefore left with vertices v where $I := S \cap N^-(v)$ satisfies $X \subset I$. Note that I is counted by every term $R[Y]$ with $X \subseteq Y \subseteq I$. Since

$$\sum_{X \subseteq Y \subseteq I} (-1)^{|Y \setminus X|} = \sum_{0 \leq k \leq |I \setminus X|} (-1)^k \binom{|I \setminus X|}{k} = 0$$

we conclude that these counts of v cancel out and contribute a sum-total of zero to the right-hand side. This covers all cases and we conclude that the claim holds. \triangleleft

It remains to be shown how the query $Q_S[X]$ can be computed using $\hat{Q}_S[X]$. To this end, consider a vertex $v \in G$ where $S \cap N(v) \neq S \cap N^-(v)$ as these contribute to $\hat{Q}_S[X]$ but must not be counted by $Q_S[X]$. Note that any such vertex must be contained in $N^-(S)$ since v has at least one right-neighbour in S . Accordingly, we apply the following correction to $\hat{Q}_S[X]$:

Let $Q_S = \hat{Q}_S$
for $u \in N^-(S)$ **do**
 $Q_S[N^-(u) \cap S] \leftarrow Q_S[N^-(u) \cap S] - 1$
 $Q_S[N(u) \cap S] \leftarrow Q_S[N(u) \cap S] + 1$

This correction takes time $O(d|S|^2)$. \blacktriangleleft

We are now ready to describe the pattern-counting algorithm.

Proof of Theorem 6. The problem is trivial for $|X| = 1$ since then the pattern is either a single edge or anti-edge. Thus assume $|X| \geq 2$ in the following, in particular for the running time calculations.

We first compute the left-covering number $\text{lc}(H)$ by simply brute-forcing all orderings of H in time $O(|H|! \cdot \max\{|X|, |Y|\}^{\text{lc}(H)}) = O(|H|! |X|^{\text{lc}(H)})$. At the same time, whenever we find that a specific ordering \mathbb{H} has a minimal left-covering of X , then we add the signature $\sigma_{\mathbb{X}}(H)$ with $\mathbb{X} := \mathbb{H}[X]$ to a collection \mathcal{X} . Similarly, if we find that a minimal left-covering in \mathbb{H} covers Y we add the signature $\sigma_{\mathbb{Y}}(H)$ with $\mathbb{Y} := \mathbb{H}[Y]$ to a collection \mathcal{Y} . We will later use that $|\mathcal{X}| \leq \text{hoa}(H, X)$ and $|\mathcal{Y}| \leq \text{hoa}(H, Y)$.

We now compute an ordering \mathbb{G} for G of degeneracy d in time $O(dn)$, sort the left-neighbourhoods in time $O(d \log d \cdot n)$ time and compute the data structure R as per Lemma 8 in time $O(d2^d n)$. If we want to compute the number of times H appears in G , we further need to initialise a subset dictionary K .

We now iterate through all subsets $C \subseteq V(G)$ of size $\text{lc}(H)$ and for each such set we iterate through all subsets $Z \subseteq N_{\mathbb{G}}^-(C) \cup C$ of size $|X|$ or $|Y|$, in total this takes time $O(n^{\text{lc}(H)} ((d+1) \text{lc}(H))^{|X|})$. We describe the remainder of the algorithm for a set $X = Z$ of size $|X|$, the procedure for a set Y works analogously. Let \mathbb{X} be the ordering of X in \mathbb{G} .

To verify that X can be completed into a pattern H in G , we compute the data structure Q_X in time $O(|X|2^{|X|} + d|X|^2)$ as per Theorem 7. To check whether H exists in G , we iterate through all signatures $\sigma \in \mathcal{X}$ and test whether $Q_X[\mathbb{X}[A]] > 0$ for all index sets $A \in \sigma$, this takes time $O(|\mathcal{X}||X||Y|)$, in total the verification step for X takes time

$$O((|X|2^{|X|} + d|X|^2) \cdot |\mathcal{X}||X||Y|) = O(d|X|^3 2^{|X|} \text{hoa}(H))$$

where we used that $|X| \geq |Y|$ and $|X| \geq 2$. This bound also holds for checking Y since $|\mathcal{X}| + |\mathcal{Y}| \leq 2 \text{hoa}(H)$. Finally, if we exhaust all orderings of H without finding the pattern, we report that it does not exist in G .

14:8 Computing Complexity Measures of Degenerate Graphs

To *count* in how many ways X can be extended into the pattern H in G , we compute

$$c_{H,X} := \sum_{\sigma \in \mathcal{X}} \prod_{A^{(k)} \in \sigma} \binom{Q_X[\mathbb{X}[A]]}{k}$$

where k denotes the multiplicity of A in the multiset σ . Note, however, that we have to take care not to double-count the contribution of X to the overall count as we might encounter the set X multiple times. To that end, we record the intermediate result by setting $K[X] := c_{H,X}$ and we forgo the above computation if X exists already as a key in K . The computation of $c_{H,X}$ and this additional book keeping takes time $O(|X| + |\mathcal{X}||X||Y|)$, in total we arrive at the same running time $O(d|X|^3 2^{|X|} \text{hoa}(H))$ like for the decision variant. After exhausting all orderings of H we report back the number of times H appears in G as the sum of all entries of K .

The total running time of either variant of the algorithm is, as claimed,

$$\begin{aligned} & O\left(|X|^{\text{lc}(H)} |H|! + d2^d n + dn + n^{\text{lc}(H)} ((d+1) \text{lc}(H))^{|X|} \cdot d|X|^3 2^{|X|} \text{hoa}(H)\right) \\ & = O\left(|X|^{\text{lc}(H)} |H|! + d2^d n + n^{\text{lc}(H)} (4d \text{lc}(H))^{|X|} d|X|^3 \text{hoa}(H)\right). \quad \blacktriangleleft \end{aligned}$$

4 Concrete applications

4.1 Finding bicliques and co-matchings

We note that $\text{lc}(K_{t,t}) = 1$ and $\text{hoa}(K_{t,t}) = 1$, therefore the application of Theorem 6 gives the following:

► **Corollary 11.** *Let G be a d -degenerate graph. Then we can compute the number of biclique patterns $K_{s,t}$ ($s \geq t$) in time $O(s \cdot (2s)! + d2^d n + n(4d)^s ds^3)$.*

Let \bar{M}_t be a co-matching on $2t$ vertices. We will assume in the following that the partite sets of \bar{M}_t are $X := (x_1, \dots, x_t)$ and $Y := (y_1, \dots, y_t)$ so that the edges $x_i y_i$ for $i \in [t]$ are forbidden.

► **Lemma 12.** $\text{lc}(\bar{M}_t) = 2$ and $\text{hoa}(\bar{M}_t) = 1$.

Proof. Let \bar{M}_t be an ordering of \bar{M}_t and let z be the last vertex in that order. Then $N^-(z)$ covers all vertices of one partite set except one vertex z' . Thus $\{z, z'\}$ is a left-cover of \bar{M}_t .

To determine the half-ordering asymmetry, note that for *every* ordering \mathbb{Z} of $Z \in \{X, Y\}$ the signature $\sigma_{\mathbb{Z}}(\bar{M}_t)$ is simply the set $\binom{[t]}{t-1}$, so the total number of signatures is one. ◀

► **Corollary 13.** *Let G be a d -degenerate graph. Then we can compute the number of co-matching patterns \bar{M}_t in time $O(t^2(2t)! + d2^d n + n^2(8d)^t dt^3)$.*

4.2 Finding shattered sets

A direct application of Theorem 6 to locate a shattered pattern U_t is unsatisfactory as the running time will include a factor of n^t since $\text{lc}(U_t) = t$. By the following observation, we can bound t by the degeneracy of the graph, but we can greatly improve the running time by further adjusting the algorithm.

► **Observation 14.** *Let G be a d -degenerate graph. Then $\text{vc}(G) \leq d + 1$.*

Proof. Assume $S \subseteq V(G)$ is shattered by $W \subseteq V(G)$, with $S = |\mathbf{vc}(G)|$. Let $W' \subseteq W$ be those witnesses that have $|S| - 1$ neighbours in S . Then $G[W' \cup S]$ induces a graph of minimum degree $|S| - 1$ and we must have that $|S| - 1 \leq d$ and accordingly $\mathbf{vc}(G) = |S| \leq d + 1$. ◀

The core observation that allows further improvements is that many orderings of U_{d+1} have degeneracy *larger* than d and can therefore not appear in a d -degenerate graph. In particular, the ordering in which all witnesses of U_{d+1} appear before the shattered set has degeneracy 2^{d+1} and can therefore be ruled out. We refine this idea further in the following lemma.

► **Lemma 15.** *Let \mathbb{G} be a d -degenerate ordering of a graph G . Let G contain the shattered pattern U_t and let $\mathbb{U}_t := \mathbb{G}[U_t]$ be its ordering. Then $\text{lc}(\mathbb{U}_t) \leq \lceil \log d + 1 \rceil$. Specifically, we either have that $t \leq \lceil \log d + 1 \rceil$ or that \mathbb{U}_t can be covered by $\lceil \log d + 1 \rceil$ witness vertices.*

Proof. Let $S = (s_1, \dots, s_t)$ and $W = (w_1, \dots, w_{2^t})$ be the vertices of U_t in G and let the indices of the variables reflect the ordering of the corresponding vertices in \mathbb{U}_t .

Partition the set S into $p := \lceil \log d + 1 \rceil$ sets S_1, \dots, S_p such that each set has size at least $\lceil t/p \rceil$ and at most $\lfloor t/p \rfloor$. For each set S_i define the set of “apex”-witnesses $A_i := \{w \in W \mid N(w) \supset S_i\}$. Note that, for all $i \in [p]$,

$$|A_i| = 2^{|S \setminus S_i|} \geq 2^{t - \lceil t/p \rceil} = 2^{\lceil t \frac{p-1}{p} \rceil}.$$

We call a set A_i *good* if $\max_{\mathbb{G}} A_i > \max_{\mathbb{G}} S_i$, that is, at least one apex vertex from A_i can be found to the right of S_i . We now distinguish two cases:

Case 1. All A_i , $i \in [p]$, are good.

It follows that \mathbb{U}_t can be left-covered by taking one vertex from each A_i , $i \in [p]$. We conclude that $\text{lc}(\mathbb{U}_t) \leq p = \lceil \log d + 1 \rceil$.

Case 2. Some A_i , $i \in [p]$, is not good.

Let $u = \max_{\mathbb{G}} S_i$ be the last vertex in S_i , note that $A_i \leq_{\mathbb{G}} u$ and accordingly $A_i \subseteq N^-(u)$. But then we must have that $|A_i| \leq d$ and accordingly that

$$\begin{aligned} 2^{\lceil t \frac{p-1}{p} \rceil} \leq d &\iff \lceil t \frac{p-1}{p} \rceil \leq \log d \implies t \frac{p-1}{p} \leq \log d \iff t \leq \frac{p}{p-1} \log d \\ &\iff t \leq \frac{\lceil \log d + 1 \rceil}{\lceil \log d + 1 \rceil - 1} \log d = \frac{\log d}{\lceil \log d \rceil} \lceil \log d + 1 \rceil \leq \lceil \log d + 1 \rceil. \end{aligned}$$

We therefore find that $\text{lc}(\mathbb{U}_t) \leq |S| \leq \lceil \log d + 1 \rceil$. ◀

► **Theorem 16.** *Let G be a d -degenerate graph on n vertices. Then we can determine the VC-dimension of its neighbourhood set system $\mathcal{F}(G)$ in time $O(n^{\lceil \log d + 1 \rceil} d^{d+2} (2d \log d)^{d+1})$.*

Proof. We first compute an ordering \mathbb{G} of G with degeneracy d in time $O(dn)$ and sort all left-neighbourhoods in time $O(d \log d \cdot n)$. Let $p := \lceil \log d + 1 \rceil$ in the following.

Let $\mathbb{U}_t = (S, W)$ be a shattered set of size $t \leq d + 1$ in \mathbb{G} . By Lemma 15 we then have that $\text{lc}(\mathbb{U}_t) \leq p$. Therefore to locate the set S we first guess up to p vertices and then exhaustively search through their (closed) left-neighbourhoods in time

$$\binom{n}{p} \binom{dp}{t} \leq \left(\frac{en}{p}\right)^p \left(\frac{edp}{t}\right)^t = O\left(n^{\lceil \log d + 1 \rceil} (d \log d)^{d+1}\right).$$

14:10 Computing Complexity Measures of Degenerate Graphs

Now that we can locate S we apply Theorem 7 in order to verify that S is indeed shattered: For each candidate set S from the previous step, we compute a subset dictionary Q_S in time $O(|S|2^{|S|} + d|S|^2) = O(d2^d)$ and then check whether $Q_S[X] > 0$ for each $X \subseteq S$. This latter step takes time $O(|S|2^{|S|})$ and is therefore subsumed by the construction time of Q_S . We conclude that the algorithm runs in total time

$$O(d \log d \cdot n) + O(d2^d n) + O\left(n^{\lceil \log d + 1 \rceil} (d \log d)^{d+1} \cdot d2^d\right) = O\left(n^{\lceil \log d + 1 \rceil} d^{d+2} (2d \log d)^{d+1}\right)$$

as claimed. \blacktriangleleft

We note that the exponent of $\lceil \log d + 1 \rceil$ in the running time is almost tight:

► **Theorem 17.** *GRAPH VC-DIMENSION parameterized by the degeneracy d of the input graph cannot be solved in time $f(d) \cdot n^{o(\log d)}$ unless all problems in SNP can be solved in subexponential time.*

Proof. We adapt the $W[1]$ -hardness reduction from k -CLIQUE to VC-DIMENSION by Downey, Evans, and Fellows [8] and combine it with the result by Chen *et al.* [5, 4] which states that k -CLIQUE cannot be solved in time $f(k)n^{o(k)}$ unless all problems in SNP admit subexponential-time algorithms.

Given an instance (H, k) for k -CLIQUE, we construct a graph G as follows. We first create k copies V_1, \dots, V_k of $V(H)$. For $v \in H$, let us denote its copies by $v^{(1)}, \dots, v^{(k)}$ with $v^{(i)} \in V_i$ for $i \in [k]$. We now add the following vertices and edges:

- A single isolated vertex w_0 ,
- a vertex set W_1 which contains one pendant vertex for each $v^{(i)}$, $v \in H$ and $i \in [k]$,
- a vertex set W_2 which for each edge $uv \in H$ contains $\binom{k}{2}$ vertices w_{uv}^{ij} , $i, j \in [k]$, each of which $u^{(i)}$ and $v^{(j)}$ as its only neighbours, and
- a vertex set A which for each index set $I \subseteq [k]$ contains a vertex a_I which is connected to all vertices in V_i for each $i \in I$.

Note that the graph is bipartite with partite sets $\mathcal{V} := V_1 \cup \dots \cup V_k$ and $\mathcal{W} := W_1 \cup W_2 \cup A$.

Let us first show that if H contains a clique of size k then G contains a shattered set of size k . Let u_1, \dots, u_k be distinct vertices that form a complete graph in H . We claim that then the set $S := \{u_1^{(1)}, \dots, u_k^{(k)}\}$ is shattered in G . First, note that for every subset $X \subset S$, $|X| \geq 3$, there exists a witness vertex $a \in A$ such that $N(a) \cap S = X$. For the empty set we have the witness w_0 , for every singleton subset $\{u\} \subseteq S$ we have that the pendant vertex $p \in N(u) \cap W_1$ witnesses $\{u\}$. Therefore, only subsets of size exactly two need to be witnessed to shatter S . Consider $\{u_i^{(i)}, u_j^{(j)}\} \subseteq S$ for $i \neq j$. Since $u_i u_j \in H$, the vertex $w_{u_i u_j}^{ij}$ exists in W_2 and its neighbourhood in S is exactly $\{u_i^{(i)}, u_j^{(j)}\}$. We conclude that all subsets of size two in S are witnessed as well and therefore S is shattered.

In the other direction, assume that G contains a shattered set (S, W) of size k . Without loss of generality, assume that $k \geq 3$.

▷ **Claim 18.** $S \subseteq \mathcal{V}$ and $W \subseteq \mathcal{W}$.

Proof. Since G is bipartite we either have that $S \subseteq \mathcal{V}$ and $W \subseteq \mathcal{W}$ or that $S \subseteq \mathcal{W}$ and $W \subseteq \mathcal{V}$. Let us now show that the latter is impossible.

Since $k \geq 3$ we have that every vertex in S has degree at least four. Accordingly, W cannot contain vertices from W_1 or W_2 , which leaves us with $W \subseteq A$. However, all vertices in V_i , $i \in [k]$, have the exact same neighbours in A . Therefore only k subsets of A are witnessed by vertices in \mathcal{V} and therefore the largest shattered set in A has size at most $\log k$. We conclude that S cannot be contained in A and the claim holds. \blacktriangleleft

We now claim that $|S \cap V_i| = 1$ for all $i \in [k]$. Assume otherwise, so let $u^{(i)}, v^{(i)} \in S$ for some $i \in [k]$. But then the set $\{u^i, v^i\}$ cannot be witnessed: not by a vertex from W_1 , since it only contains vertices with one neighbour, not by a vertex from W_2 , since these vertices each have at most one neighbour in each set V_i , and not by a vertex from A since we need all $2^k - \binom{k}{2} - k - 1$ vertices of A to witness subsets of S of size at least three.

Therefore S intersects each V_i in exactly one vertex. Since S is shattered, every subset $\{u^{(i)}, v^{(j)}\}$, $i \neq j$, is shattered. By the same logic as above, this can only be due to a witness $w_{uv}^{ij} \in W_2$ and therefore $uv \in H$. We conclude that indeed u_1, \dots, u_k induce a complete graph in H , as claimed.

Finally, we need to determine the degeneracy of G . Consider the following elimination sequence: We first delete all of $\{w_0\} \cup W_1 \cup W_2$, all of which have degree at most two. Note now that all vertices in \mathcal{V} have at most $|A| < 2^k$ neighbours in A , so we delete \mathcal{V} and then A . In total, the maximum degree we encountered in this deletion sequence is $< 2^k$.

Assume we could solve GRAPH VC-DIMENSION in time $f(d)n^{o(\log d)}$. In the above reduction the degeneracy of the constructed graph is $d < 2^k$, thus this running time for GRAPH VC-DIMENSION would imply a running time of

$$f(d) \cdot n^{o(\log d)} = f(2^k) \cdot n^{o(\log 2^k)} = f(2^k) \cdot n^{o(k)}$$

for k -CLIQUE. We conclude that VC-DIMENSION parameterized by the degeneracy of the input graph cannot be solved in time $f(d)n^{o(\log d)}$ unless all problems in SNP can be solved in subexponential time. \blacktriangleleft

We note that Lemma 15 allows us to approximate the VC-dimension of degenerate graphs.

► Theorem 19. *Let G be a d -degenerate graph on n vertices. Then for any $0 < \varepsilon \leq 1$ we can approximate the VC-dimension of G in time $O(d2^d(2n)^{\lceil \varepsilon(1+\log d) \rceil})$ within a factor of ε .*

Proof. We first compute a d -degenerate ordering \mathbb{G} of G in time $O(dn)$ and sort its left-neighbourhoods in time $O(d \log d \cdot n)$. Let $U_t = (S, W)$ be the largest shattered set in G and let \mathbb{U}_t be its ordering in \mathbb{G} . We further prepare the use of Theorem 7 by computing the necessary data structure in time $O(d2^d n)$.

Let $c := \lceil \varepsilon(1 + \log d) \rceil$. The algorithm now iterates over all $C \subseteq V(G)$ of size c and searches the left-neighbourhood $L := N^-[C]$ for a shattered set by first computing a subset dictionary Q_L in time $O(d2^d)$ and then finding the largest shattered subset $S \subseteq L$ by brute-force in time $O(|L|2^{|L|}) = O(cd2^{cd})$.

We claim that this simple algorithm computes the claimed approximation of the VC-dimension. By Lemma 15 we either have that $t \leq \log d + 1$ or that \mathbb{U}_t can be left-covered by $\log d + 1$ witness vertices. In the first case, our algorithm will trivially locate an ε -fraction of a maximal solution since it tests every set of size c . In the second case, the shattered set S of \mathbb{U}_t is covered by the left-neighbourhood of witness vertices $w_1, \dots, w_p \in W$ for $p := \log d + 1$. Then by simple averaging, there exist c witnesses W' such that $|N^-[W'] \cap S| \geq c|S|/p = ct/(\log d + 1)$. Since the above algorithm will find the shattered set $N^-[W] \cap S$ when inspecting the left-neighbourhood of W , we conclude that it will output at least a value of $ct/(\log d + 1)$. In either case the approximation factor is $\frac{c}{1+\log d} \geq \varepsilon$, as claimed. \blacktriangleleft

We would like to highlight the special case of $c = 1$ of the above theorem as it provides us with a linear-time approximation of the VC-dimension, which is probably a good starting point for practical applications:

► Corollary 20. *Let G be a d -degenerate graph on n vertices. Then we can approximate the VC-dimension of G in time $O(d2^d n)$ within a factor of $\frac{1}{1+\log d}$.*

4.3 Approximating the ladder and semi-ladder index

Before we proceed, we note that degenerate graphs cannot contain arbitrarily long ladders:

► **Observation 21.** *If G is d -degenerate then G cannot contain a ladder of length $2d + 2$.*

Proof. Note that a ladder of length t contains a complete bipartite graph $K_{\lfloor t/2 \rfloor, \lfloor t/2 \rfloor}$, i.e. a subgraph of minimum degree $\lfloor t/2 \rfloor$. Therefore $t < 2d + 2$. ◀

Again we find that a direct application of Theorem 6 to ladder patterns does not yield a satisfying running time since $\text{lc}(L_t) \approx t/2$. However, we can always left-cover a large portion of a ladder with only one vertex:

► **Observation 22.** *Let (A, B) induce a ladder of length t in G . For every ordering \mathbb{G} of G there exists a vertex $u \in A \cup B$ such that $|N^-(u) \cap (A \cup B)| \geq \lfloor t/2 \rfloor$.*

Proof. Let $A' := (a_i)_{i \geq t/2}$ and $B' := (b_i)_{i \leq t/2}$, then $G[A' \cup B']$ contains a biclique with partite sets A', B' . Let $u \in A' \cup B'$ be the largest vertex according to $<_{\mathbb{G}}$, then $N^-(u) \cap (A', B')$ is either all of A' or all of B' . In either case the claim holds. ◀

► **Theorem 23.** *Let G be a d -degenerate graph on n vertices and let t be its ladder-index. Then we can in time $O(d^2 8^d \cdot n)$ decide whether G contains a ladder of size at least $\lfloor t/2 \rfloor$.*

Proof. We compute a degeneracy ordering \mathbb{G} of G and initialize the data structure R as per Lemma 8 in time $O(2^d n)$.

Let (A, B) induce a ladder of maximum size t in G , by Observation 21 we have that $t \leq 2d + 1$. By Observation 22, there exists a vertex $u \in A \cup B$ such that $N^-(u)$ contains either $A' := (a_i)_{i \geq t/2}$ or $B' := (b_i)_{i \geq t/2}$. Wlog assume $A' \subseteq N^-(u)$ and let $k := |A'|$. We guess u in $O(n)$ time and $A' \subseteq N^-(u)$ in time $O(2^d)$. To verify that A' can be completed into a ladder, we compute the data structure $Q_{A'}$ in time $O(k2^k + dk)$ using Lemma 9.

Finally, we verify that there exists a sequence of subsets $A'_1 \subset A'_2 \subset \dots \subset A'_k = A'$ where $Q_{A'}[A'_i] > 0$ for all $i \in [k]$; as each lookup in $Q_{A'}$ has cost equal to the size of the query set this will take time proportional to $\sum_{i=0}^k i \binom{k}{i} = k2^{k-1}$ in the worst case (where we have to query all subsets of A' before finding the sequence). Since $k := \lfloor t/2 \rfloor$, the total running time of this algorithm is

$$O(2^d n) + O\left(2^d n \cdot (k2^k + dk) \cdot k2^{k-1}\right) = O\left(2^d k 2^k (k2^k + dk) \cdot n\right).$$

We can simplify this expression further by using that $k \leq d$ which leads us to the claimed running time of $O(d^2 8^d \cdot n)$. ◀

5 Implementation and experiments

Based on the above theoretical ideas, we implemented algorithms² to compute the VC-dimension, find the largest biclique, co-matchings (within an additive error of 1) and ladder (within a factor 2). The last three algorithms all simply check the left-neighbourhood for the respective structure. Aside from optimisations of the involved data structures we will not describe these algorithms in further detail.

We observe that for practical purposes the data structure R can be computed progressively: if we know that our algorithm currently only needs to compute Q_S from R (as per Lemma 9) with $|S| = k$ ($k \leq d$), then it is enough to only count sets of size $\leq k$ in R . We can achieve this

² Source code available under <https://github.com/microgravitas/mantis-shrimp/>

in time $O(\binom{d}{k}n)$, which is far preferable to using $O(2^d n)$ time to insert all left-neighbourhood subsets into R . If k remains much smaller than d , this improves our running time and space consumption substantially.

The second important optimisation regards subset dictionaries. While tries are useful in our theoretical analysis, in practice we opted to use bitsets for the data structures Q_S , as their universe S can be assumed to be small. Bitsets also allow for a very concise and fast implementation of the fast Möbius inversion, which needs to happen very frequently inside the hot loop of the search algorithms.

The algorithm to compute the VC-dimension includes a few simple optimisations that vastly improved its performance. Note that if we are currently searching for a shattered set of size k , then a candidate vertex for a shattered set of size k must have at least $\binom{k-1}{i-1}$ neighbours of degree at least i , for $1 \leq i \leq k-1$. Our algorithm recomputes the set of remaining candidates each time it finds a larger shattered set. The (progressive) computation of the data structure R can then also be restricted to only those left-neighbourhood subsets which only contain candidate vertices.

Accordingly, the algorithm performs well if it finds large shattered sets fast. To that end, it first only looks at k -subsets of left-neighbourhoods of single vertices. Once that search is exhausted, it considers left-neighbourhoods of pairs, then triplets, *etc.* up to $\lceil \log d + 1 \rceil$ vertices (as per Lemma 15). As this search is very expensive once we need to consider the joint left-neighbourhood of several vertices, the algorithm estimates the work needed and compares it against simply brute-forcing all k -subsets of the remaining candidates. Since the number of candidates shrinks quite quickly in practice, the algorithm usually concludes with such a final exhaustive search.

5.1 Results

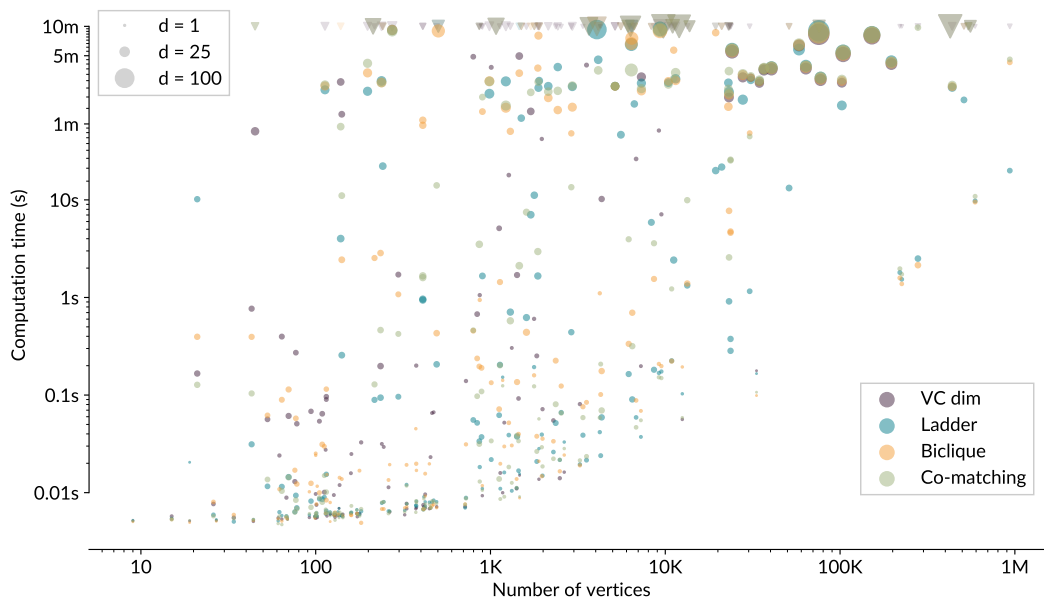


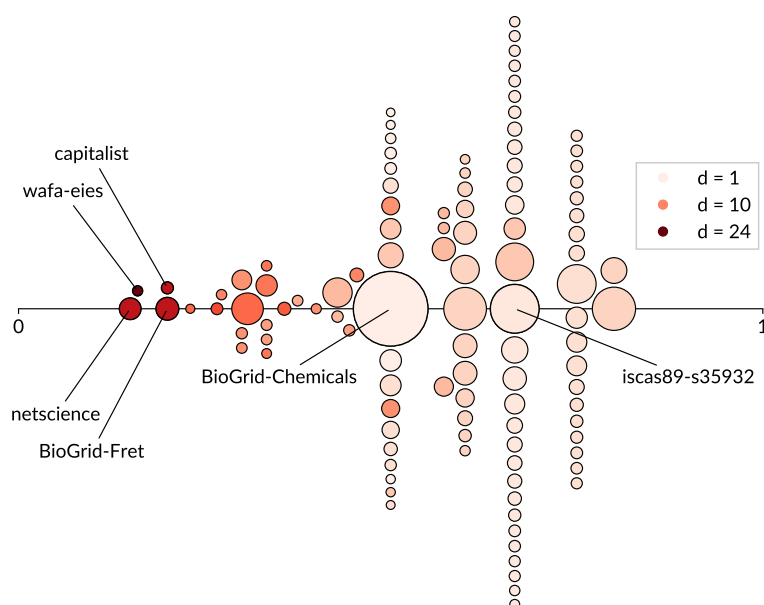
Figure 1 Running times of all four algorithms on a collection of 206 networks. The size of the circles indicates the degeneracy of the networks, triangles indicate that program timed out on the network after 10 minutes.

14:14 Computing Complexity Measures of Degenerate Graphs

We implemented all four algorithms in Rust and tested them on a diverse collection of 206 networks³, using a PC with a AMD Ryzen 3 2200G CPU and 24 GB RAM. The primary goal of our experiments was to verify that the data structures and algorithms in this paper could be of practical use, therefore we ran each algorithm only once per network⁴ and timed out after 10 minutes.

Of all the four measures, computing the VC-dimension is, unsurprisingly, the most computationally challenging and the program timed out or ran out of memory for networks larger than a few ten-thousand nodes or of degeneracy higher than 24. The broad summary of the results looks as follows (Figure 1 visualises these results in more detail):

Statistics	Completed	Max size (n)	Max degeneracy
VC-dimension	126	33266 (BioGrid-Chemicals)	24 (wafa-eies)
Biclique	176	935591 (teams)	191 (BioGrid-All)
Co-matching	179	935591 (teams)	255 (dogster_friendships)
Ladder index	187	935591 (teams)	191 (BioGrid-All)



■ **Figure 2** VC-dimension of networks normalized by their degeneracy + 1. Networks with large degeneracy tend towards the left, meaning that the VC-dimension does not increase proportionally to the degeneracy.

We are also interested in typical values of the VC-dimension of networks and how it compares to the degeneracy. This topic deserves a deeper investigation, but we can report some preliminary results here for those networks where our program terminated before the timeout. In Figure 2 we normalised the VC-dimension by the degeneracy-plus-one, so values close to one indicate that the VC-dimension is on the order of the degeneracy while values close to zero indicate that it is much smaller than the degeneracy. We see a clear tendency that networks with larger degeneracy tend towards zero, which we interpret as the VC-dimension “growing slower” than the degeneracy in typical networks.

³ <https://github.com/microgravitas/network-corpus>

⁴ The variance in running times was on the orders of seconds.

6 Conclusion

On the theoretical side, we outlined a general bipartite pattern-finding and -counting algorithm in degenerate graphs. Its running time crucially depends on two complexity measures of patterns, namely the left-covering number and the half-ordering asymmetry. These general algorithms can be further improved for specific patterns, which we exemplify for shattered set, ladder, co-matching and biclique patterns. Our results also include improved running times when the input graphs are of bounded degeneracy.

On the experimental side, we demonstrate that this style of algorithm is feasible and practical for computation on real-world networks, which often exhibit low degeneracy. The experiments also suggest that the VC-dimension of networks tends to be a very small parameter, which makes it an interesting target for the development of fast algorithms that exploit low VC-dimension.

References

- 1 Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Linear time subgraph counting, graph degeneracy, and the chasm at size six. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151, pages 38:1–38:20. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.38.
- 2 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set Partitioning via Inclusion-Exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009. doi:10.1137/070683933.
- 3 Marco Bressan and Marc Roth. Exact and Approximate Pattern Counting in Degenerate Graphs: New Algorithms, Hardness Results, and Complexity Dichotomies. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 276–285, 2022. doi:10.1109/FOCS52979.2021.00036.
- 4 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. On the computational hardness based on linear FPT-reductions. *Journal of Combinatorial Optimization*, 11(2):231–247, 2006. doi:10.1007/s10878-006-7137-6.
- 5 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346–1367, 2006. doi:10.1016/j.jcss.2006.04.007.
- 6 Norishige Chiba and Takao Nishizeki. Arboricity and Subgraph Listing Algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985. doi:10.1137/0214017.
- 7 Erik D. Demaine, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, Somnath Sikdar, and Blair D. Sullivan. Structural sparsity of complex networks: Bounded expansion in random models and real-world graphs. *Journal of Computer and System Sciences*, 105:199–241, 2019.
- 8 Rodney G. Downey, Patricia A. Evans, and Michael R. Fellows. Parameterized learning complexity. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 51–57, 1993.
- 9 Pål Grønås Drange, Patrick Greaves, Irene Muzi, and Felix Reidl. Computing complexity measures of degenerate graphs. *CoRR*, arXiv:2308.08868 [cs.DS], 2023. doi:10.48550/arXiv.2308.08868.
- 10 Eduard Eiben, Robert Ganian, Thekla Hamm, Lars Jaffke, and O-joung Kwon. A Unifying Framework for Characterizing and Computing Width Measures. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215, pages 63:1–63:23. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.63.
- 11 Kord Eickmeyer, Archontia C. Giannopoulou, Stephan Kreutzer, O-joung Kwon, Michał Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. Neighborhood Complexity and Kernelization for Nowhere Dense Classes of Graphs. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80, pages 63:1–63:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.63.

14:16 Computing Complexity Measures of Degenerate Graphs

- 12 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM Journal of Experimental Algorithmics*, 18:3.1:3.1–3.1:3.21, 2013. doi:10.1145/2543629.
- 13 Grzegorz Fabiański, Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. Progressive algorithms for domination and independence. In *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126, pages 27:1–27:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.27.
- 14 Donald E. Knuth. *Art of computer programming, volume 2: Seminumerical algorithms*. Addison–Wesley Professional, 2014.
- 15 David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983. doi:10.1145/2402.322385.
- 16 Christos H. Papadimitriou and Mihalis Yannakakis. On limited nondeterminism and the complexity of the VC dimension. *Journal of Computer and System Sciences*, 53(2):161–170, 1996.
- 17 Robert Sedgewick and Kevin Wayne. *Algorithms, 4th edition*. Addison–Wesley, 2011.
- 18 Frank Yates. The design and analysis of factorial experiments. *Imperial Bureau of Soil Science*, 1937.

A Complete results

For space reasons, some of the network names are abbreviated below. Abbreviated names are marked in gray. Some of the rows have been deferred to the full version of the paper [9].

Network	n	m	δ	\bar{d}	deg	Δ	Running time				Statistics			
							VC-dim	biclique	crown	ladder	VC-dim	biclique	crown	ladder
AS-oregon-1	11174	23409	2389	4.2	17	2389	600.09	342.08	175.43	2.42	[5,18]	12	[13,14]	[17,35]
AS-oregon-2	11461	32730	2432	5.7	31	2432	600.40	167.57	201.43	174.27	[6,32]	[7,31]	[7,32]	[7,63]
BG-AC-Luminescence	1840	2312	376	2.5	6	376	0.25	0.04	0.03	0.02	4	5	7	[6,13]
BG-AC-Rna	13765	42815	3572	6.2	54	3572	600.21	601.05	601.38	601.54	[4,55]	[5,54]	[5,55]	[5,109]
BG-AC-Western	21028	64046	535	6.1	17	535	600.08	600.30	600.29	21.70	[5,18]	[7,17]	[9,18]	[17,35]
BG-Biochemical-Activity	8620	17746	427	4.1	11	427	600.11	1.55	3.59	0.18	[5,12]	8	[7,8]	[11,23]
BG-Bos-Taurus	454	424	27	1.9	3	27	0.02	0.01	0.01	0.01	2	3	[3,4]	[3,7]
BG-C.-Elegans	6394	23646	522	7.4	64	522	600.21	399.05	214.18	390.57	[4,65]	[6,64]	[5,65]	[6,129]
BG-Canis-Familiaris	143	125	90	1.7	2	90	0.01	0.01	0.01	0.01	2	2	3	[2,5]
BG-Chemicals	33266	28093	413	1.7	1	413	0.18	0.10	0.11	0.17	1	[0,1]	2	[0,3]
BG-Co-Localization	3543	4452	63	2.5	6	63	240.56	0.08	0.02	0.02	3	5	7	[6,13]
BG-Co-Purification	4326	5970	1972	2.8	12	1972	10.21	0.18	0.07	0.06	4	6	13	[12,25]
BG-Cricetulus-Griseus	69	57	30	1.7	1	30	0.01	0.01	0.01	0.01	1	[0,1]	2	[0,3]
BG-D.-Discoideum-Ax4	27	20	4	1.5	1	4	0.01	0.01	0.01	0.01	1	[0,1]	2	[0,3]
BG-D.-Growth-Defect	1447	2193	213	3.0	5	213	0.09	0.04	0.03	0.02	4	4	6	[5,11]
BG-D.-Melanogaster	9330	60556	303	13.0	83	303	600.32	538.71	553.50	566.57	[4,84]	[5,83]	[5,84]	[5,167]
BG-E.-Nidulans-Fgsc-A4	64	62	44	1.9	2	44	0.01	0.01	0.00	0.01	2	2	3	[2,5]
BG-E.-Coli-K12-Mg1655	1273	1889	58	3.0	5	58	17.95	0.05	0.02	0.04	3	4	6	[5,11]
BG-Far-Western	1199	1089	60	1.8	3	60	0.05	0.03	0.01	0.02	3	3	4	[3,7]
BG-Fret	1700	2395	51	2.8	19	51	80.71	600.09	126.45	7.04	4	[11,19]	[17,18]	[19,39]
BG-Hepatitis-C-Virus	136	134	133	2.0	1	133	0.01	0.01	0.01	0.01	1	[0,1]	2	[0,3]
BG-Homo-Sapiens	24093	369767	2882	30.7	71	2882	329.05	338.13	343.83	346.77	[3,72]	[3,71]	[3,72]	[3,143]
BG-HSV-1	178	208	40	2.3	3	40	0.01	0.02	0.01	0.01	3	3	4	[3,7]

Continued on next page

Network	n	m	δ	\bar{d}	deg	Δ	Running time				Statistics			
							VC-dim	biclique	crown	ladder	VC-dim	biclique	crown	ladder
BG-HSV-5	121	107	27	1.8	1	27	0.01	0.01	0.01	0.01	1	[0,1]	2	[0,3]
BG-HSV-8	716	691	119	1.9	3	119	0.01	0.01	0.01	0.01	2	3	[3,4]	[3,7]
BG-HPV-16	173	186	93	2.2	2	93	0.01	0.01	0.01	0.01	2	2	[2,3]	[2,5]
Cannes2013	438089	835892	15169	3.8	27	15169	600.79	144.70	149.74	142.22	[5,28]	[6,27]	[6,28]	[6,55]
CoW-interstate	182	319	25	3.5	4	25	0.03	0.00	0.01	0.01	3	4	[3,4]	[4,9]
EU-email-core	986	16064	345	32.6	34	345	600.86	164.82	163.41	122.48	[5,35]	[6,34]	[6,35]	[6,69]
JDK_dependency	6434	53658	5923	16.7	65	5923	600.41	453.02	600.99	601.73	[4,66]	[5,65]	[5,66]	[5,131]
NZ_legal	2141	15739	429	14.7	25	429	600.36	110.96	128.55	146.23	[6,26]	[7,25]	[7,26]	[7,51]
Noordin-terror-loc	127	190	18	3.0	3	18	0.01	0.01	0.01	0.01	3	3	[3,4]	[3,7]
Noordin-terror-orgas	129	181	21	2.8	3	21	0.01	0.01	0.01	0.01	3	3	[3,4]	[3,7]
ODLIS	2900	16377	592	11.3	12	592	600.03	48.04	13.49	0.44	[5,13]	6	[9,10]	[12,25]
Opsahl-forum	899	7036	128	15.7	14	128	600.02	80.31	113.04	1.66	[5,15]	5	[6,7]	[14,29]
Y2H_union	1966	2705	89	2.8	4	89	42.11	0.01	0.03	0.04	3	4	5	[4,9]
Yeast	2361	7182	66	6.1	10	66	600.04	0.23	0.08	0.05	[4,11]	7	[9,10]	[10,21]
actor_movies	511463	1470404	646	5.7	14	646	600.36	600.51	600.46	105.76	[5,15]	[7,14]	[6,15]	[14,29]
airlines	235	1297	130	11.0	13	130	0.20	2.85	0.46	0.09	5	8	[11,12]	[13,27]
american_revolution	141	160	59	2.3	3	59	0.01	0.01	0.01	0.01	3	3	[2,3]	[3,7]
autobahn	374	478	5	2.6	2	5	0.01	0.05	0.01	0.01	2	2	3	[2,5]
bahamas	219856	246291	14902	2.2	6	14902	600.12	1.59	1.97	1.80	[3,7]	6	[3,4]	[6,13]
bergen	53	272	32	10.3	9	32	0.06	0.06	0.01	0.01	4	6	[8,9]	[9,19]
bitcoin-otc-positive	5573	18591	788	6.7	20	788	600.14	600.45	600.24	46.50	[6,21]	[8,20]	[10,21]	[20,41]
bn-fly-d._medulla_1	1781	8911	927	10.0	18	927	600.04	600.17	600.15	11.17	[5,19]	[8,18]	[8,19]	[18,37]
boards_gender_2m	4220	5598	45	2.7	4	45	600.10	1.10	0.06	0.04	[3,5]	4	[3,4]	[4,9]
ca-CondMat	23133	93439	279	8.1	25	279	600.05	600.65	208.73	600.65	[4,26]	[14,25]	26	[18,51]
ca-HepPh	12006	118489	491	19.7	238	491	600.21	600.33	600.13	600.17	[3,239]	[3,238]	[3,239]	[3,477]
celegans	297	2148	134	14.5	10	134	1.72	1.08	0.42	0.10	5	7	[8,9]	[10,21]
chess	7301	55899	181	15.3	29	181	182.00	130.97	138.03	157.14	[6,30]	[6,29]	[6,30]	[6,59]
cit-HepTh	27769	352285	2468	25.4	37	2468	180.07	189.90	194.29	106.40	[4,38]	[4,37]	[4,38]	[3,75]
codeminer	724	1015	55	2.8	4	55	0.14	0.03	0.01	0.01	3	4	[4,5]	[4,9]
columbia-mobility	863	4147	228	9.6	9	228	600.11	0.20	0.03	0.03	[4,10]	6	10	[9,19]

Continued on next page

Network	n	m	δ	\bar{d}	deg	Δ	Running time				Statistics			
							VC-dim	biclique	crown	ladder	VC-dim	biclique	crown	ladder
cora_citation	23166	89157	377	7.7	13	377	600.11	7.72	2.57	0.91	[5,14]	9	[10,11]	[13,27]
countries	592414	624402	110602	2.1	6	110602	600.14	9.49	10.88	9.75	[4,7]	5	[4,5]	[6,13]
diseasome	1419	2738	84	3.9	11	84	1.70	0.14	0.04	0.04	4	6	12	[11,23]
dogster_friendships	426820	8546581	46505	40.0	255	46505	600.38	600.57	600.00	600.44	[1,256]	[0,255]	[1,256]	[0,511]
dolphins	62	159	12	5.1	4	12	0.02	0.01	0.01	0.01	3	3	5	[4,9]
edinburgh_assoc._thes.	23132	297094	1062	25.7	34	1062	112.09	119.67	123.49	128.27	[3,35]	[3,34]	[3,35]	[3,69]
email-Enron	36692	183831	1383	10.0	43	1383	213.38	219.83	223.53	219.47	[4,44]	[4,43]	[4,44]	[4,87]
exnet-water	1893	2416	10	2.6	2	10	0.01	0.02	0.06	0.03	2	2	3	[2,5]
facebook-links	63731	817090	1098	25.6	52	1098	221.29	229.98	233.75	237.25	[3,53]	[3,52]	[3,53]	[3,105]
foldoc	13356	91471	728	13.7	12	728	600.10	1.39	9.93	1.33	[5,13]	12	[9,10]	[12,25]
foodweb-otago	141	832	45	11.8	14	45	75.41	2.43	11.06	0.26	4	12	[7,8]	[14,29]
football	115	613	12	10.7	8	12	0.10	0.12	0.01	0.02	4	4	9	[8,17]
haggle	274	2124	101	15.5	39	101	600.11	551.34	533.96	550.92	[4,40]	[8,39]	[8,40]	[8,79]
hex	331	930	6	5.6	3	6	0.01	0.02	0.01	0.01	3	2	[3,4]	[3,7]
hypertext_2009	113	2196	98	38.9	28	98	600.15	146.97	150.88	134.22	[5,29]	[9,28]	[9,29]	[9,57]
ia-infect-dublin	410	2765	50	13.5	17	50	600.10	65.59	1.60	0.94	[4,18]	9	[16,17]	[17,35]
ia-reality	6809	7680	261	2.3	5	261	26.27	0.09	0.05	0.06	4	4	[5,6]	[5,11]
iscas89-s1238	416	625	18	3.0	2	18	0.01	0.01	0.01	0.01	2	2	[2,3]	[2,5]
iscas89-s13207	2492	3406	37	2.7	4	37	0.01	0.02	0.02	0.02	4	4	[4,5]	[4,9]
iscas89-s1423	423	554	17	2.6	2	17	0.01	0.01	0.01	0.01	2	2	[2,3]	[2,5]
iscas89-s1494	473	796	56	3.4	3	56	0.07	0.01	0.01	0.01	3	3	[3,4]	[3,7]
iscas89-s15850	3247	4004	25	2.5	4	25	0.08	0.02	0.02	0.02	3	4	[3,4]	[4,9]
iscas89-s298	92	131	11	2.8	2	11	0.01	0.01	0.01	0.01	2	2	[2,3]	[2,5]
iscas89-s344	100	122	9	2.4	2	9	0.01	0.02	0.01	0.01	2	2	[2,3]	[2,5]
iscas89-s349	102	127	9	2.5	2	9	0.01	0.01	0.01	0.01	2	2	[2,3]	[2,5]
iscas89-s382	116	168	18	2.9	2	18	0.01	0.02	0.01	0.01	2	2	[2,3]	[2,5]
iscas89-s38417	9500	10635	39	2.2	4	39	7.11	0.20	0.15	0.17	4	2	[4,5]	[4,9]
iscas89-s400	121	182	19	3.0	2	19	0.01	0.01	0.01	0.01	2	2	[2,3]	[2,5]
iscas89-s420	129	145	9	2.2	2	9	0.01	0.01	0.01	0.01	2	2	[2,3]	[2,5]
iscas89-s444	134	206	19	3.1	2	19	0.01	0.01	0.01	0.01	2	2	3	[2,5]

Continued on next page

Network	n	m	δ	\bar{d}	deg	Δ	Running time				Statistics			
							VC-dim	biclique	crown	ladder	VC-dim	biclique	crown	ladder
iscas89-s526	160	270	12	3.4	3	12	0.02	0.01	0.01	0.01	3	3	[2,3]	[3,7]
iscas89-s526n	159	268	12	3.4	3	12	0.02	0.01	0.01	0.01	3	3	[3,4]	[3,7]
iscas89-s713	137	180	12	2.6	3	12	0.01	0.01	0.01	0.01	3	2	[2,3]	[3,7]
iscas89-s820	239	480	48	4.0	3	48	0.03	0.01	0.01	0.01	3	3	[3,4]	[3,7]
iscas89-s832	245	498	49	4.1	3	49	0.03	0.01	0.01	0.01	3	3	[3,4]	[3,7]
iscas89-s9234	1985	2370	18	2.4	4	18	0.07	0.04	0.02	0.01	3	4	[3,4]	[4,9]
iscas89-s953	332	454	12	2.7	2	12	0.01	0.01	0.01	0.01	2	2	[2,3]	[2,5]
lederberg	8324	41532	1103	10.0	15	1103	600.11	600.11	600.08	5.88	[5,16]	[7,15]	[9,16]	[15,31]
lesmiserables	77	254	36	6.6	9	36	0.27	0.06	0.01	0.01	3	6	10	[9,19]
link-pedigree	898	1125	14	2.5	2	14	0.01	0.01	0.01	0.01	2	2	[2,3]	[2,5]
livemocha	104103	2193083	2980	42.1	92	2980	308.64	318.47	325.14	326.65	[2,93]	[2,92]	[2,93]	[2,185]
loc-brightkite_edges	58228	214078	1134	7.4	52	1134	381.48	385.52	393.61	346.98	[5,53]	[5,52]	[5,53]	[5,105]
mg_casino	109	326	94	6.0	9	94	0.06	0.03	0.01	0.01	3	5	10	[9,19]
mg_forrestgump	94	271	89	5.8	8	89	0.07	0.01	0.01	0.01	3	4	9	[8,17]
mg_godfatherII	78	219	34	5.6	8	34	0.05	0.01	0.01	0.01	3	5	9	[8,17]
minnesota	2642	3303	5	2.5	2	5	0.02	0.02	0.03	0.03	2	2	3	[2,5]
moreno_health	2539	10455	27	8.2	7	27	600.10	0.12	0.07	0.07	[4,8]	5	[7,8]	[7,15]
movies	101	192	19	3.8	3	19	0.01	0.01	0.01	0.01	3	2	[3,4]	[3,7]
muenchen-bahn	447	578	13	2.6	2	13	0.01	0.02	0.01	0.01	2	[0,1]	[2,3]	[2,5]
munin	1324	1397	66	2.1	3	66	0.30	0.03	0.01	0.01	2	3	[2,3]	[3,7]
offshore	278877	505965	37336	3.6	13	37336	600.11	2.14	583.20	2.50	[5,14]	13	[9,10]	[13,27]
openflights	2939	15677	242	10.7	28	242	600.11	89.14	147.54	144.73	[5,29]	[7,28]	[7,29]	[7,57]
paradise	542102	794545	35359	2.9	23	35359	600.21	601.52	600.31	601.57	[5,24]	[22,23]	[6,24]	[23,47]
photoviz_dynamic	376	610	29	3.2	4	29	0.20	0.02	0.01	0.01	3	3	[3,4]	[4,9]
pigs	492	592	39	2.4	2	39	0.01	0.01	0.01	0.01	2	2	[2,3]	[2,5]
polbooks	105	441	25	8.4	6	25	0.05	0.01	0.01	0.01	4	5	[6,7]	[6,13]
pollination-carlinville	1500	15255	157	20.3	18	157	600.04	600.37	600.30	68.80	[5,19]	[6,18]	[6,19]	[18,37]
ratbrain	503	23030	497	91.6	67	497	600.50	538.67	601.38	600.83	[4,68]	[5,67]	[5,68]	[5,135]
reactome	6327	147547	855	46.6	191	855	600.17	600.17	600.21	600.23	[3,192]	[3,191]	[3,192]	[3,383]
residence_hall	217	1839	56	16.9	11	56	600.10	2.54	0.13	0.09	[4,12]	6	[10,11]	[11,23]


Continued on next page

Network	n	m	δ	\bar{d}	deg	Δ	Running time				Statistics			
							VC-dim	biclique	crown	ladder	VC-dim	biclique	crown	ladder
roget-thesaurus	1010	3648	28	7.2	6	28	228.14	0.13	0.03	0.02	4	3	[6,7]	[6,13]
slashdot_threads	51083	117378	2915	4.6	14	2915	600.13	600.17	600.14	13.23	[5,15]	[5,14]	[6,15]	[14,29]
soc-advogato	5167	39432	807	15.3	25	807	600.47	145.63	146.80	145.69	[5,26]	[6,25]	[6,26]	[6,51]
soc-gplus	23628	39194	2761	3.3	12	2761	600.04	4.74	25.84	0.28	[6,13]	9	[7,8]	[12,25]
soc-hamsterster	2426	16630	273	13.7	24	273	600.10	83.26	131.04	229.61	[5,25]	[11,24]	[23,25]	[24,49]
sp_data_school_day_2	238	5539	88	46.5	33	88	600.31	159.31	159.79	165.56	[5,34]	[6,33]	[7,34]	[7,67]
teams	935591	1366466	2671	2.9	9	2671	600.24	256.73	275.42	19.94	[6,10]	6	[6,7]	[9,19]
twittercrawl	3656	154824	1084	84.7	143	1084	600.46	600.68	600.37	600.47	[3,144]	[3,143]	[3,144]	[3,287]
unicode_languages	868	1255	141	2.9	4	141	1.06	0.94	0.01	0.02	3	4	[3,4]	[4,9]
wafa-ceos	26	93	22	7.2	5	22	0.01	0.01	0.01	0.01	3	3	[5,6]	[5,11]
wafa-hightech	21	159	20	15.1	12	20	0.17	0.40	0.13	10.16	3	7	[10,11]	[8,17]
wafa-padgett	15	27	8	3.6	3	8	0.01	0.01	0.01	0.01	2	2	[3,4]	[3,7]
web-google	1299	2773	59	4.3	17	59	600.02	50.40	0.58	0.71	[3,18]	9	18	[17,35]
wikipedia-norm	1881	15372	455	16.3	22	455	600.14	482.15	161.62	140.63	[6,23]	[10,22]	[11,23]	[11,45]
win95pts	99	112	9	2.3	2	9	0.01	0.01	0.01	0.01	2	2	3	[2,5]
word_adjacencies	112	425	49	7.6	6	49	0.01	0.03	0.01	0.01	4	4	[5,6]	[6,13]
zewail	6651	54182	331	16.3	18	331	600.09	601.22	601.20	96.19	[5,19]	[9,18]	[10,19]	[18,37]

An Improved Kernelization Algorithm for Trivially Perfect Editing

Maël Dumas 

Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans, France

Anthony Perez 

Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans, France

Abstract

In the TRIVIALY PERFECT EDITING problem one is given an undirected graph $G = (V, E)$ and an integer k and seeks to add or delete at most k edges in G to obtain a trivially perfect graph. In a recent work, Dumas *et al.* [16] proved that this problem admits a kernel with $O(k^3)$ vertices. This result heavily relies on the fact that the size of trivially perfect modules can be bounded by $O(k^2)$ as shown by Drange and Pilipczuk [14]. To obtain their cubic vertex-kernel, Dumas *et al.* [16] then showed that a more intricate structure, so-called *comb*, can be reduced to $O(k^2)$ vertices. In this work we show that the bound can be improved to $O(k)$ for both aforementioned structures and thus obtain a kernel with $O(k^2)$ vertices. Our approach relies on the straightforward yet powerful observation that any large enough structure contains unaffected vertices whose neighborhood remains unchanged by an editing of size k , implying strong structural properties.

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

Keywords and phrases Parameterized complexity, kernelization algorithms, graph modification, trivially perfect graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.15

1 Introduction

In the TRIVIALY PERFECT EDITING problem one is given an undirected graph $G = (V, E)$ and an integer k and seeks to *edit* (add or delete) at most k edges in G so that the resulting graph is trivially perfect (*i.e.* does not contain any cycle on four vertices nor path on four vertices as an induced subgraph). More formally we consider the following problem:

TRIVIALY PERFECT EDITING

Input: A graph $G = (V, E)$, a *parameter* $k \in \mathbb{N}$

Question: Does there exist a set $F \subseteq [V]^2$ of size at most k , such that the graph $H = (V, E \Delta F)$ is trivially perfect?

Here $[V]^2$ denotes the set of all pairs of elements of V and $E \Delta F = (E \cup F) \setminus (E \cap F)$ is the symmetric difference between sets E and F . We define similarly the deletion (resp. completion) variant of the problem by only allowing to delete (resp. add) edges. Graph modification covers a broad range of well-studied problems that find applications in various areas. For instance, TRIVIALY PERFECT EDITING has been used to define the community structure of complex networks by Nastos and Gao [31] and is closely related to the well-studied graph parameter tree-depth [21, 33]. Theoretically, some of the earliest NP-Complete problems are graph modification problems [26, 20]. Regarding edge (graph) modification problems, one of the most notable one is the MINIMUM FILL-IN problem which aims at adding



© Maël Dumas and Anthony Perez;

licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 15; pp. 15:1–15:17

Leibniz International Proceedings in Informatics



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

edges to a given graph to obtain a chordal graph (*i.e.* a graph that does not contain any induced cycle of length at least 4). In a seminal result, Kaplan *et al.* [25] proved that MINIMUM FILL-IN admits a parameterized algorithm as well as a kernel containing $O(k^3)$ vertices. This result was later improved to $O(k^2)$ vertices by Natanzon *et al.* [32]. Parameterized complexity and kernelization algorithms provide a powerful theoretical framework to cope with decision problems.

Parameterized complexity

A parameterized problem Π is a language of $\Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. An instance of a parameterized problem is a pair (I, k) with $I \subseteq \Sigma^*$ and $k \in \mathbb{N}$, called the *parameter*. A parameterized problem is said to be *fixed-parameter tractable* if it can be decided in time $f(k) \cdot |I|^{O(1)}$. An equivalent definition of fixed-parameter tractability is the notion of *kernelization*. Given an instance (I, k) of a parameterized problem Π , a *kernelization algorithm* for Π (kernel for short) is a polynomial-time algorithm that outputs an equivalent instance (I', k') of Π such that $|I'| \leq h(k)$ for some function h depending on the parameter only and $k' \leq k$. It is well-known that a parameterized problem is fixed-parameter tractable if and only if it admits a kernelization algorithm (see *e.g.* [19]). Problem Π is said to admit a *polynomial kernel* whenever h is a polynomial.

Related work

Since the work of Kaplan *et al.* [25] many polynomial kernels for edge modification problems have been devised (see *e.g.* [3, 2, 12, 24, 1, 14, 27, 15, 11]). There is also evidence that under some reasonable theoretical complexity assumptions, some graph modification problems do not admit polynomial kernels [28, 23, 8, 30]. We refer the reader to a recent comprehensive survey on kernelization for edge modification problems by Crespelle *et al.* [9]. The TRIVIALY PERFECT EDITING problem has been well-studied in the literature [5, 22, 4, 1, 14, 16, 29, 31]. Recall that trivially perfect graphs are a subclass of chordal graphs that additionally do not contain any path on four vertices as an induced subgraph. These graphs are also known as *quasi-threshold* graphs. We note here that while the NP-Completeness of completion and deletion toward trivially perfect graphs has been known for some time [34, 6], the NP-Completeness of TRIVIALY PERFECT EDITING remained open until the work of Nastos and Gao [31]. Thanks to a result of Cai [7] stating that graph modification toward *any* graph class characterized by a finite set of forbidden induced subgraphs is fixed-parameter tractable, TRIVIALY PERFECT EDITING is fixed-parameter tractable. Regarding kernelization algorithms, Drange and Pilipczuk [14] provided a kernel containing $O(k^7)$ vertices, a result that was recently improved to $O(k^3)$ vertices by Dumas *et al.* [16]. These results also work for the deletion and completion variants. For the latter problem, a recent result by Bathie *et al.* [1] improves the bound to $O(k^2)$ vertices.

As part of the proof for the size of their cubic vertex-kernel, Dumas *et al.* [16] subsequently showed the following result. The structures used in Theorem 1 shall be defined later.

► **Theorem 1** ([16]). *Let (G, k) be an instance¹ of TRIVIALY PERFECT EDITING such that the sizes of its trivially perfect modules and combs are bounded by $p(k)$ and $c(k)$, respectively. If (G, k) is a YES-instance then G has $O(k^2 + k \cdot (p(k) + c(k)))$ vertices.*

¹ As we shall see Section 3.1 the instance also needs to be further reduced under standard reduction rules.

The proof of [16, Theorem 1] actually implies an $O(k^3 + k \cdot (p(k) + c(k)))$ bound and needs a small adjustment for Theorem 1 to hold. We give a detailed proof of Theorem 1 for the sake of completeness (Section 3.2).

The cubic vertex-kernel of Dumas *et al.* [16] relied on a result of Drange and Pilipczuk [14] that proved that $p \in O(k^2)$ and then used new reduction rules implying that $c \in O(k^2)$.

Our contribution

We provide reduction rules and structural properties on trivially perfect graphs that will imply an $O(k)$ bound for both functions p and c of Theorem 1. These new reduction rules allow us to prove the existence a quadratic vertex-kernel for TRIVIAALLY PERFECT EDITING. To bound the size of trivially perfect modules by $O(k)$, we first reduce the ones that contain a large matching of non-edges with the use of a simple reduction rule. To bound the ones that do not contain such structures, we will rely on so-called *combs*, introduced by Dumas *et al.* [16]. Combs correspond to parts of the graph that induce trivially perfect graphs (but not necessarily modules) with strong properties on their neighborhoods. They are composed of two main parts, called the *shaft* and the *teeth*, that will be independently reduced to a size linear in k . The reduction rule dealing with shafts will ultimately allow us to bound the size of trivially perfect modules with no large matching of non-edges. Our approach relies on the straightforward yet powerful observation that any large enough structure contains unaffected vertices whose neighborhood remains unchanged by an editing of size k . Finally, we note that our kernel works for both the deletion and completion variants of the problem.

Outline

Section 2 presents some preliminary notions and structural properties on (trivially perfect) graphs. Section 3 describes known as well as our additional reduction rules to obtain the claimed kernelization algorithm while Section 4 explain why our kernel is safe for the deletion variant of the problem. We conclude with some perspectives in Section 5.

2 Preliminaries

We consider simple, undirected graphs $G = (V, E)$ where V denotes the vertex set of G and $E \subseteq [V]^2$ its edge set. We will sometimes use $V(G)$ and $E(G)$ to clarify the context. The open (respectively closed) neighborhood of a vertex $u \in V$ is denoted by $N_G(u) = \{v \in V \mid \{u, v\} \in E\}$ (respectively $N_G[u] = N_G(u) \cup \{u\}$). Given a subset of vertices $S \subseteq V$ the neighborhood of S is defined as $N_G(S) = \cup_{v \in S} N_G(v) \setminus S$. Similarly, given a vertex $u \in V$ and $S \subseteq V$ we let $N_S(u) = N_G(u) \cap S$. In all aforementioned cases we forget the subscript mentioning graph G whenever the context is clear. Given a subset of vertices $S \subseteq V$ we denote by $G[S]$ the subgraph induced by S , that is $G[S] = (S, E_S)$ where $E_S = \{uv \in E : u \in S, v \in S\}$. In a slight abuse of notation, we use $G \setminus S$ to denote the induced subgraph $G[V \setminus S]$. A *connected component* is a maximal subset of vertices $S \subseteq V$ such that $G[S]$ is connected. A *module* of G is a set $M \subseteq V$ such that for all $u, v \in M$ it holds that $N(u) \setminus M = N(v) \setminus M$. Two vertices u and v are *true twins* whenever $N[u] = N[v]$, and a *critical clique* is a maximal set of true twins. A vertex $u \in V$ is *universal* if $N_G[u] = V$. The set of universal vertices forms a clique and is called the *universal clique* of G . A graph is *trivially perfect* if and only if it does not contain any C_4 (a cycle on 4 vertices) nor P_4 (a path on 4 vertices) as an induced subgraph. In the remainder of this section we describe

characterizations and structural properties of trivially perfect graphs. The first one relies on the well-known fact that any connected trivially perfect graph contains a universal vertex (see *e.g.* [36]).

► **Definition 2** (Universal clique decomposition, [13]). *A universal clique decomposition (UCD) of a connected graph $G = (V, E)$ is a pair $\mathcal{T} = (T = (V_T, E_T), \mathcal{B} = \{B_t\}_{t \in V_T})$ where T is a rooted tree and \mathcal{B} is a partition of the vertex set V into disjoint nonempty subsets such that:*

- *if $\{v, w\} \in E$ and $v \in B_t, w \in B_s$ then s and t are on a path from a leaf to the root, with possibly $s = t$,*
- *for every node $t \in V_T$, the set B_t of vertices is the universal clique of the induced subgraph $G[\bigcup_{s \in V(T_t)} B_s]$, where T_t denotes the subtree of T rooted at t .*

A simple way of understanding Definition 2 is to observe that such a decomposition can be obtained by removing the set U of universal vertices of G and then recursively repeating this process on every trivially perfect connected component of $G \setminus U$. Drange *et al.* [13] showed that a connected graph admits a UCD if and only if it is trivially perfect. Using the notion of UCD, Dumas *et al.* [16] proved the following characterization for trivially perfect graphs that will be heavily used in our reduction rules. A collection of subsets $\mathcal{F} \subseteq 2^U$ over some universe U is a *nested family* if $A \subseteq B$ or $B \subseteq A$ holds for any $A, B \in \mathcal{F}$.

► **Lemma 3** ([16]). *Let $G = (V, E)$ be a graph, $S \subseteq V$ a maximal clique of G and $\{K_1, \dots, K_r\}$ the set of connected components of $G \setminus S$. The graph G is trivially perfect if and only if the following conditions are verified:*

- (i) $G[S \cup K_i]$ is trivially perfect, $1 \leq i \leq r$
- (ii) $\bigcup_{1 \leq i \leq r} \{N_G(K_i)\}$ is a nested family
- (iii) $\forall u \in K_i, \forall v \in N_G(K_i), \{u, v\} \in E, 1 \leq i \leq r$. In other words, K_i is a module of G .

In the remainder of this paper, a *k-editing* of G into a trivially perfect graph is a set $F \subseteq [V]^2$ such that $|F| \leq k$ and the graph $H = (V, E \Delta F)$ is trivially perfect. Here $E \Delta F = (E \cup F) \setminus (E \cap F)$ denotes the symmetric difference between sets E and F . For the sake of readability, we simply speak of *k-editing* of G . We say that F is a *k-completion* (resp. *k-deletion*) when $H = (V, E \cup F)$ (resp. $H = (V, E \setminus F)$) is trivially perfect. A vertex is *affected* by a *k-editing* F if it is contained in some pair of F and *unaffected* otherwise.

Packing, anti-matching and blow-up

We now define some structures and operators that will be useful for our kernelization algorithm. We assume in the remainder of this section that we are given a graph $G = (V, E)$. The notion of *r-packing* will be used in reduction rules to ensure the existence of unaffected vertices in ordered sets of critical cliques or of trivially perfect modules.

► **Definition 4** (*r-packing*). *Let $\mathcal{S} = \{C_1, \dots, C_q\}$ be an ordered collection of pairwise disjoint subsets of V . We say that $\mathcal{C} \subseteq \mathcal{S}$ is a *r-packing* of \mathcal{S} if $\mathcal{C} = \{C_1, \dots, C_p\}$ for $1 \leq p \leq q$, $\sum_{i=1}^p |C_i| \geq r$ and the number of vertices contained in \mathcal{C} is minimum for this property.*

In a slight abuse of notation we use \mathcal{C} to denote both $\{C_1, \dots, C_p\}$ and the set $\bigcup_{i=1}^p C_i$.

► **Observation 5.** *Let $\mathcal{S} = \{C_1, \dots, C_q\}$ be an ordered collection of pairwise disjoint subsets of V such that $|C_j| \leq c$, for $1 \leq j \leq q$ and some integer $c > 0$. Let $\mathcal{C} = \{C_1, \dots, C_p\}$ be a *r-packing* of \mathcal{S} . Then $\sum_{i=1}^p |C_i| \leq r + (c - 1)$.*

Proof. Since $\sum_{i=1}^p |C_i| \geq r$ and the number of vertices in \mathcal{C} is minimum for this property we have that $\sum_{i=1}^{p-1} |C_i| \leq r - 1$. The result follows from the fact that $|C_p| \leq c$. ◀

► **Definition 6** (Anti-matching). *An anti-matching of G is a set of pairwise disjoint pairs $\{u, v\}$ of vertices of G such that $\{u, v\} \notin E$.*

In a slight abuse of notation we denote by $V(D)$ the set of vertices contained in pairs of an anti-matching D .

► **Observation 7.** *Let (G, k) be a YES-instance of TRIVIAALLY PERFECT EDITING and M be a module containing a $(k + 1)$ -sized anti-matching. Let F be a k -editing of G and $H = G \Delta F$. Then $N_G(M)$ is a clique in H .*

Proof. Let $D = \{\{u_i, v_i\} \mid 1 \leq i \leq k + 1\}$ be a $(k + 1)$ -sized anti-matching of M . Assume for a contradiction that $N_G(M)$ is not a clique in H and let $\{u, v\}$ be a non-edge of H with $u, v \in N_G(M)$. Since $|F| \leq k$ there exists $1 \leq j \leq k + 1$ such that $\{u_j, v_j\} \notin F$ and for every $x \in V(G) \setminus M$, $\{u_j, x\}, \{v_j, x\} \notin F$. Hence $\{u_j, u, v_j, v\}$ induces a C_4 in H , a contradiction. ◀

We conclude this section by introducing a gluing operation on trivially perfect graphs, namely *blow-up*, that will ease the design of some reduction rules.

► **Definition 8** (Blow-up). *Let u be a vertex of $G = (V, E)$ and $H = (V_H, E_H)$ be any graph. The blow-up of G by H at u , denoted $G(u \rightarrow H)$ is the graph obtained by replacing u by H in G . More formally:*

$$G(u \rightarrow H) = ((V \setminus \{u\}) \cup V_H, E(G \setminus \{u\}) \cup E_H \cup (V_H \times N_G(u)))$$

► **Proposition 9.** *Assume that G is trivially perfect and let u be a vertex of G such that $N_G[u]$ is a clique. For any trivially perfect graph H , the graph $G(u \rightarrow H)$ is trivially perfect.*

Proof. Let $S \subseteq V \setminus \{u\}$ be any maximal clique of G containing $N_G(u)$. We apply the forward direction of Lemma 3 on S to obtain components $\{K_1, \dots, K_r\}$ that are modules such that $G[S \cup K_i]$ is trivially perfect for every $1 \leq i \leq r$ and $\bigcup_{1 \leq i \leq r} \{N_G(K_i)\}$ is a nested family. Note that by construction and w.l.o.g., we may assume $K_1 = \{u\}$. The result then directly follows from the reverse direction of Lemma 3 by replacing K_1 by H . ◀

3 Reduction rules

In the remainder of this section we assume that we are given an instance $(G = (V, E), k)$ of TRIVIAALLY PERFECT EDITING.

3.1 Standard reduction rules

We first describe some well-known reduction rules [2, 3, 14, 16] that are essential to obtain a vertex-kernel using Theorem 1 [16]. We will assume in the remainder of this work that the instance at hand is reduced under Rules 1 and 2, meaning that none of them applies to the instance.

► **Rule 1.** *Let $C \subseteq V$ be a connected component of G such that $G[C]$ is trivially perfect. Remove C from G .*

► **Rule 2.** *Let $K \subseteq V$ be a critical clique of G such that $|K| > k + 1$. Remove $|K| - (k + 1)$ arbitrary vertices in K from G .*

► **Lemma 10** (Folklore, [2, 14]). *Rules 1 and 2 are safe and can be applied in polynomial time.*

3.2 An $O(k)$ bound on the size of trivially perfect modules

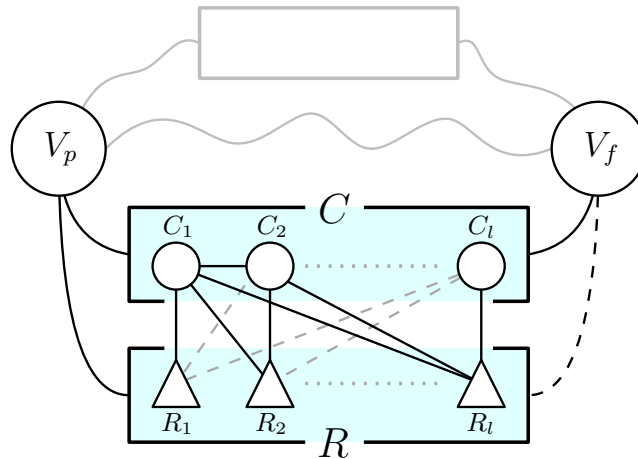
Using an additional reduction rule bounding the size of independent sets in any trivially perfect module by $O(k)$, Drange and Pilipczuk [14] proved that such modules can be reduced to $O(k^2)$ vertices. We strengthen this result by proving that trivially perfect modules can further be reduced to $O(k)$ vertices. We first deal with modules that contain a large anti-matching.

► **Rule 3.** Let $M \subseteq V$ be a trivially perfect module of G . If $G[M]$ contains a $(k + 1)$ -sized anti-matching D , then remove the vertices contained in $M \setminus V(D)$.

► **Lemma 11.** Rule 3 is safe.

Proof. Let $G' = (V', E')$ be the graph obtained after application of Rule 3. We need to prove that $(G = (V, E), k)$ is a YES-instance if and only if $(G' = (V', E'), k)$ is a YES-instance. The forward direction is straightforward since G' is an induced subgraph of G and trivially perfect graphs are hereditary. We now consider the reverse direction. Let $M' = V(D)$, the set of vertices kept by Rule 3. Moreover, let F' be a k -editing of G' and $H' = G' \Delta F'$. We will construct a k -editing F^* of G . Note that since the pairs contained in an anti-matching are disjoint (Definition 6), $|M'| = 2(k + 1)$. Moreover, since $|F'| \leq k$ there are at most $2k$ affected vertices. Hence let u be an unaffected vertex of M' . By Observation 7 and since M' contains a $(k + 1)$ -sized anti-matching we have that $N_{G'}(M')$ is a clique in H' . The graph $H_u = H' \setminus (M' \setminus \{u\})$ is trivially perfect by heredity and $N_{H_u}(u) = N_{G'}(M')$. It follows that $N_{H_u}(u)$ is a clique and Proposition 9 implies that the graph $H = H_u(u \rightarrow M)$ is trivially perfect. Let F^* be the editing such that $H = G \Delta F^*$. Since u is unaffected by F' and $u \in M$ we have $N_{H_u}(u) = N_G(M)$. Hence, since M is a module in G we have that $N_H(v) = N_G(v)$ for every vertex $v \in M$, implying that $F^* \subseteq F'$. This concludes the proof. ◀

In order to bound the size of any trivially perfect module by $O(k)$, we actually prove a more general reduction rule that will be useful for the rest of our kernelization algorithm. This rule operates on a more intricate structure, so-called *comb* [16], that induces a trivially perfect graph but not necessarily a module.



■ **Figure 1** A comb of a graph $G = (V, E)$ with shaft C and teeth R . Each set C_i is a critical clique while each set R_i induces a (possibly disconnected) trivially perfect module, $1 \leq i \leq l$. Notice that the sets V_p and V_f might be adjacent to some other vertices of the graph.

- **Definition 12** (Comb [16]). *A pair (C, R) of disjoint subsets of V is a comb of G if:*
- $G[C]$ is a clique that can be partitioned into l critical cliques $\{C_1, \dots, C_l\}$
 - R can be partitioned into l non-empty non-adjacent trivially perfect modules $\{R_1, \dots, R_l\}$
 - $N_G(C_i) \cap R = \bigcup_{j=i}^l R_j$ and $N_G(R_i) \cap C = \bigcup_{j=1}^i C_j$ for $1 \leq i \leq l$
 - there exist two (possibly empty) subsets of vertices $V_f, V_p \subseteq V(G) \setminus \{C \cup R\}$ such that:
 - $\forall x \in C, N_G(x) \setminus (C \cup R) = V_p \cup V_f$ and
 - $\forall y \in R, N_G(y) \setminus (C \cup R) = V_p$.

Given a comb (C, R) , C is called the *shaft* of the comb and R the *teeth* of the comb. See Figure 1 for an illustration of Definition 12. Recall that we assume that the graph is reduced under Rule 2, which means that $|C_i| \leq k + 1$ for $1 \leq i \leq l$. Dumas *et al.* [16] showed the following proposition on the structure of combs.

- **Proposition 13** ([16]). *Given a comb (C, R) of G , the subgraph $G[C \cup R]$ is trivially perfect. Moreover the sets V_p and V_f , and the ordered partitions (C_1, \dots, C_l) of C and (R_1, \dots, R_l) of R are uniquely determined.*

In the following we assume that any comb (C, R) is given with the ordered partitions (C_1, \dots, C_l) of C and (R_1, \dots, R_l) of R . We note here that Definition 12 slightly differs from the one given in [16] where the set V_f was required to be non-empty for technical reasons. Dropping this constraint will ease the presentation of our reduction rules.

We now give several observations that will help understand Definition 12, in particular its relation to trivially perfect modules. Given a trivially perfect graph $G = (V, E)$ and its UCD $\mathcal{T}_G = (T, \mathcal{B})$, one can construct a comb (C, R) of G by simply taking a path P from a node v_1 of T to one of its descendent v_l . The shaft C are the vertices in bags of this path, the teeth R are the bags of subtrees rooted in the children (not on P) of any node on the path P . We can observe that in this case, V_p corresponds to vertices in the bags on the path from the parent of v_1 to the root of T and that V_f is empty.

In particular, the vertex set of any connected trivially perfect graph can be partitioned into a comb (C, R) by taking a path from the root of its UCD to one of its leaves. This means that when $V_p = V_f = \emptyset$, Definition 12 corresponds to a connected trivially perfect graph. Similarly, if only the set V_f is empty then Definition 12 corresponds to a *connected trivially perfect module* since for every $u \in C \cup R$ it holds that $N_G(u) \setminus (C \cup R) = V_p$.

The following directly comes from the definition of a comb and is verified whether sets V_p and V_f are empty or not.

- **Observation 14.** *The set of vertices C (resp. R) is a module of $G \setminus R$ (resp. $G \setminus C$).*

We will show that combs can be safely reduced to $O(k)$ vertices. We first focus on combs having a large shaft, which will allow us to reduce trivially perfect modules with small anti-matching to $O(k)$ vertices (Lemma 20). Then we turn our attention to combs with many vertices in the teeth to bound the size of every comb to $O(k)$ vertices (Lemma 25).

Combs with large shafts

Dumas *et al.* [16] showed that the length of a comb (*i.e.* the number l of different critical cliques in the shaft) can be reduced linearly in k . However, as critical cliques contain $O(k)$ vertices by Rule 2, it only allowed the authors to bound the number of vertices in shafts of combs to $O(k^2)$. Rule 4 presented in this section keeps two sets \mathcal{C}_a and \mathcal{C}_b containing a linear number (in k) of vertices at the beginning and at the end of the shaft, allowing to bound

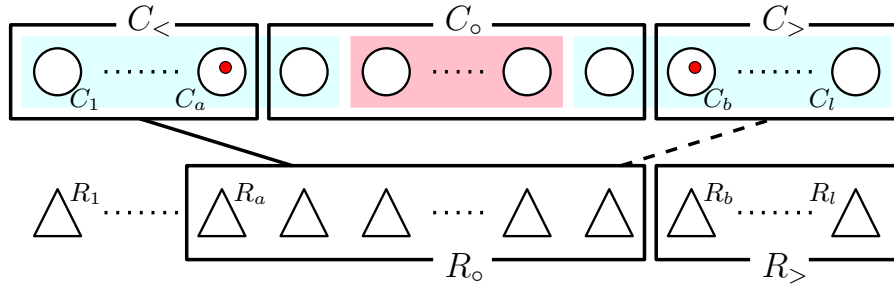
its size linearly in k . The two sets \mathcal{C}_a and \mathcal{C}_b will be large enough to ensure the existence of two vertices that will be unaffected by a given k -editing of the graph. We will use such vertices to prove that there exists a k -editing of the graph that does not affect any vertex in the shaft lying between \mathcal{C}_a and \mathcal{C}_b , implying the safeness of the rule.

► **Rule 4.** Let (C, R) be a comb of G such that there exist disjoint $(2k + 1)$ -packings \mathcal{C}_a of $\{C_1, \dots, C_l\}$ and \mathcal{C}_b of $\{C_l, C_{l-1}, \dots, C_1\}$. Remove $C' = C \setminus (\mathcal{C}_a \cup \mathcal{C}_b)$ from G .

► **Lemma 15.** Rule 4 is safe.

Proof. Let $G' = G \setminus C'$ be the graph obtained after application of Rule 4. Since G' is an induced subgraph of G and since trivially perfect graphs are hereditary, any k -editing of G is a k -editing of G' .

For the reverse direction, let F' be a k -editing of G' and $H' = G' \Delta F'$. We will construct a k -editing F^* of G . Let c_a and c_b be unaffected vertices in \mathcal{C}_a and \mathcal{C}_b , respectively. Note that both sets contain at least $2k + 1$ vertices and that F' affects at most $2k$ vertices, hence c_a and c_b are well-defined. Let C_a and C_b be the critical cliques of C containing c_a and c_b , $1 \leq a < b \leq l$. Moreover, let $C_o = C_{a+1} \cup \dots \cup C_{b-1}$ and $R_o = R_a \cup \dots \cup R_{b-1}$. Similarly, let $C_{<} = C_1 \cup \dots \cup C_a$, $C_{>} = C_b \cup \dots \cup C_l$ and $R_{>} = R_b \cup \dots \cup R_l$. These sets are depicted Figure 2. Finally, let $G_o = G \setminus C_o$ and $H_o = H' \setminus C_o$. Notice in particular that H_o is trivially perfect and that $C' \subseteq C_o$.



■ **Figure 2** Illustration of the comb and the sets used in the proof of Lemma 15. The circles are critical cliques of the shaft and the triangles are teeth. The red vertices correspond to c_a and c_b , the light blue rectangles correspond to sets \mathcal{C}_a and \mathcal{C}_b and the light red rectangle corresponds to C' , which is removed by Rule 4.

Let $F_o \subseteq F'$ be the k -editing such that $H_o = G_o \Delta F_o$ and S_o be a maximal clique of H_o containing $\{c_a, c_b\}$. Notice that since c_a and c_b are unaffected, S_o is included in $N_{G_o}(\{c_a, c_b\}) = C \cup V_p \cup V_f \cup R_{>}$. We use Lemma 3 on S_o to obtain a set of connected components $\{K_1, \dots, K_r\}$ of $H_o \setminus S_o$ such that $\{K_1, \dots, K_r\}$ are modules in H_o whose (possibly empty) neighborhoods in S_o form a nested family. We first modify F_o to obtain a k -editing of G_o where vertices of R_o are affected uniformly.

▷ **Claim 16.** There exists a k -editing F^* of G_o such that, in $H^* = G_o \Delta F^*$, R_o is a module and $H^*[R_o] = G[R_o]$.

Proof. We begin with several useful observations. First, R_o is a module in G_o since $R \supset R_o$ is a module in $G \setminus C$ (Observation 14) and vertices of R_o are adjacent to $C_{<}$ and non adjacent to $C_{>}$. Next, since any component K_i is a module in H_o , $1 \leq i \leq r$, and since c_a and c_b are unaffected by F_o , we have $N_{H_o}(K_i) \cap \{c_a, c_b\} = N_{G_o}(K_i) \cap \{c_a, c_b\}$. In other words, vertices in a same component K_i must have the same adjacency with $\{c_a, c_b\}$ in G_o and in

H_o . Similarly, no vertex $v \in R_o$ belongs to S_o since $N_{G_o}(v) \cap \{c_b\} = \emptyset$. Moreover, the only vertices of G_o that are adjacent to c_a but not c_b are exactly those of R_o . Hence for any vertex $v_o \in R_o$ it holds that $N_{H_o}(v_o) \subseteq S_o \cup R_o$.

Assume now that R_o is not a module in H_o and let $v_o \in R_o$ be a vertex contained in the least number of pairs of F_o with the other element in S_o . Consider the graph $\tilde{H} = H_o \setminus (R_o \setminus \{v_o\})$, which is trivially perfect by heredity. Since $N_{H_o}(v_o) \subseteq S_o \cup R_o$, it follows that $N_{\tilde{H}}(v_o) \subseteq S_o$ is a clique. Hence Proposition 9 implies that the graph $H^* = \tilde{H}(v_o \rightarrow G[R_o])$ is trivially perfect. Let F^* be the editing such that $H^* = G_o \triangle F^*$. By the choice of v_o we have $|F^*| \leq |F_o|$. It follows that F^* is a desired k -editing, concluding the proof of Claim 16. \triangleleft

We henceforth consider $H^* = G_o \triangle F^*$ where F^* is the k -editing from Claim 16. Note that the components around S_o may be different in $H_o \setminus S_o$ and $H^* \setminus S_o$. In a slight abuse of notation, we still define these components by $\{K_1, \dots, K_r\}$. Recall that $\{K_1, \dots, K_r\}$ are modules in H^* whose (possibly empty) neighborhoods in S_o form a nested family.

\triangleright Claim 17. The graph $H = G \triangle F^*$ is trivially perfect.

Proof. The graph H corresponds to H^* where vertices of C_o have been added with the same neighborhood as in G . Let us first observe that $S = S_o \cup C_o$ is a maximal clique in H . Indeed, C_o is a clique by definition and $S_o \subseteq (C \cup V_p \cup V_f \cup R_{>}) \subseteq N_H(C_o) = N_G(C_o)$ (recall that C is adjacent to $V_p \cup V_f$ by Definition 12 and that vertices of C_o are adjacent to every vertex of $R_{>}$). Hence components $\{K_1, \dots, K_r\}$ defined in $H^* \setminus S_o$ are the same in $H \setminus S$ and their neighborhoods are nested in S_o . We split $\{K_1, \dots, K_r\}$ into three types components w.r.t their adjacencies with $\{c_a, c_b\}$, namely:

1. α -components that are non-adjacent to both c_a and c_b
2. β -components that are adjacent to c_a but not c_b
3. δ -components that are adjacent to both c_a and c_b

In what follows we let K_α , K_β and K_δ denote any α -, β - and δ -component, respectively. Note that $N_{H^*}(K_\alpha) \subseteq N_{H^*}(K_\beta) \subseteq N_{H^*}(K_\delta) \subseteq S_o$ holds by construction. Recall that since c_a and c_b are unaffected by F^* , $N_G(K_i) \cap \{c_a, c_b\} = N_H(K_i) \cap \{c_a, c_b\}$ for any K_i . We claim that $\{N_H(K_i) \mid 1 \leq i \leq r\}$ is a nested family. Note that Lemma 3 will imply the result since S is a maximal clique in H . To sustain this claim, recall that the neighborhoods of vertices of C_o are identical in G and H . Moreover, $N_H[c_b] \subseteq N_H[C_o] \subseteq N_H[c_a]$ holds as these vertices are unaffected by F^* . It follows that α -components (resp. δ -components) are non-adjacent (resp. adjacent) to every vertex of C_o in H . This means in particular that the neighborhoods of both α - and δ -components are nested in S . Moreover we can observe that vertices of β -components are exactly the ones of R_o since they are the only ones that are adjacent to c_a but not c_b in G . Hence, in H , we still have:

$$N_H(K_\alpha) \subseteq N_H(K_\beta) \subseteq N_H(K_\delta)$$

It remains to prove that the neighborhoods of β -components are nested in S . Let w.l.o.g. $\{K_1, \dots, K_p\}$, $1 \leq p \leq r$ be the β -components. By definition of a comb, the β -components (which are also R_o) can be ordered w.r.t. the inclusion of their neighborhood in $G[C_o]$. We can assume w.l.o.g. that the ordering is $N_{G[C_o]}(K_1) \subseteq \dots \subseteq N_{G[C_o]}(K_p)$. Moreover we can observe that for any β -component K_i we have $N_{G[C_o]}(K_i) = N_{H[C_o]}(K_i)$, $1 \leq i \leq p$. Since R_o is a module in H^* by Claim 16 and since vertices of β -components are exactly those of R_o , it follows that the neighborhoods of β -components are nested. Hence $\{N_H(K_i) \mid 1 \leq i \leq r\}$ is a nested family and H is a trivially perfect graph by Lemma 3. \triangleleft

15:10 An Improved Kernelization Algorithm for Trivially Perfect Editing

By Claim 17 the graph $H = G \triangle F^*$ is trivially perfect and as $|F^*| \leq k$, it follows that F^* is a k -editing of G , concluding the proof of Lemma 15. \blacktriangleleft

► **Observation 18.** *Assume that the instance (G, k) is reduced under Rules 2 and 4. For any comb (C, R) of G it holds that $|C| \leq 6k + 2$.*

Proof. Since G is reduced under Rule 2 every critical clique C_i of the shaft contains at most $k + 1$ vertices, $1 \leq i \leq l$. By Observation 5, any $(2k + 1)$ -packing of $\{C_1, \dots, C_l\}$ (resp. $\{C_l, \dots, C_1\}$) contains at most $3k + 1$ vertices. It follows that $|C| \leq 6k + 2$ since otherwise one could find two *disjoint* $(2k + 1)$ -packings of $\{C_1, \dots, C_l\}$ and of $\{C_l, \dots, C_1\}$ and Rule 4 would apply. \blacktriangleleft

We are now ready to show how to reduce the size of any trivially perfect module. We need a combinatorial result that will be useful to obtain the claimed bound.

► **Lemma 19.** *Let $G = (V, E)$ be a connected trivially perfect graph and α be the size of a maximum anti-matching of G . There exists a comb (C, R) of G such that $V = C \cup R$ and $|R| \leq 4\alpha$. Moreover, such a comb can be computed in polynomial time.*

Proof. We provide a constructive proof that will directly imply the last part of the result. Recall that any trivially perfect graph contains a universal vertex and let $U_1 \subseteq V(G)$ be the universal clique of G . Let $R_1^1, \dots, R_{p_1}^1$ denote the connected components of $G \setminus U_1$. Since G does not contain any $(\alpha + 1)$ -sized anti-matching, there is at most one set R_i^1 , $1 \leq i \leq p_1$ such that $|R_i^1| > \alpha$ (as there is no edge between R_i^1 and R_j^1 , $1 \leq i < j \leq p_1$).

Assume without loss of generality that $|R_1^1| > \alpha$. We add all vertices of $\cup_{i=2}^{p_1} R_i^1$ to some set $R_{<}$ and we will repeat this process on $G[R_1^1]$ until every connected component is smaller than α . More formally, at step $j > 1$, for the trivially perfect graph $G_j = G[R_1^{j-1}]$, let U_j be its universal clique and $R_1^j, \dots, R_{p_j}^j$ be the connected components of $G_j \setminus U_j$. Let R_1^j be the one of size greater than α if it exists, if it does not, stop the process and let l be the last step. In particular, $|R_i^l| \leq \alpha$, $1 \leq i \leq p_l$. Let $R_{<} = \cup_{j=1}^{l-1} \cup_{i=2}^{p_j} R_i^j$ and $R_{>} = R_1^l \cup \dots \cup R_{p_l}^l$.

Recall that $|R_1^{l-1}| > \alpha$ by construction. This implies that $|R_{<}| \leq \alpha$ since otherwise $G[R_{<} \cup R_1^{l-1}]$ would contain a $(\alpha + 1)$ -sized anti-matching. We claim that $|R_{>}| \leq 3\alpha$. To support this claim, let us consider the $(\alpha + 1)$ -packing $\{R_1^l, \dots, R_{p_l}^l\}$ of $\{R_1^l, \dots, R_{p_l}^l\}$ and let $R' = \cup_{i=1}^{p_l} R_i^l$ be its vertices. Let $R'' = R_{>} \setminus R'$. Recall that l is the last step of the process and $|R_i^l| \leq \alpha$ for $1 \leq i \leq p_l$. Hence by Observation 5 it holds that $|R'| \leq 2\alpha$. Thus, we have that $|R''| \leq \alpha$ since otherwise $G[R' \cup R'']$ would contain a $(\alpha + 1)$ -sized anti-matching, a contradiction. Hence $|R_{>}| = |R'| + |R''| \leq 3\alpha$.

To obtain a comb for G we consider the set $C = \{U_1, \dots, U_l\}$ as the shaft (recall that U_1 is the universal clique of G and that U_j denotes the universal clique of $G[R_1^{j-1}]$ at every step $1 < j \leq l$). Moreover, for every $1 \leq j < l$, the tooth R_j is equal to $R_2^j \cup \dots \cup R_{p_j}^j$, the last tooth R_l being $R_{>}$. By construction $(C, R = \cup_{j=1}^l R_j)$ is a comb of G such that $|R| = |R_{<}| + |R_{>}| \leq 4\alpha$. This concludes the proof. \blacktriangleleft

► **Lemma 20.** *Assume that the instance (G, k) is reduced under Rules 1–4 and let M be a trivially perfect module of G . Then M contains at most $11k + 2$ vertices.*

Proof. Observe that if M contains an anti-matching of size more than k , then it is reduced under Rule 3 and contains $2k + 2$ vertices. Hence, suppose that M does not contain a $(k + 1)$ -sized anti-matching. Assume first that $G[M]$ is connected. Let (C, R) be a comb obtained through Lemma 19, such that $C \cup R = M$ and $|R| \leq 4k$. By Observation 18 we have that $|C| \leq 6k + 2$. It follows that $|M| \leq |C| + |R| \leq 10k + 2$.

To conclude it remains to deal with the case where $G[M]$ is disconnected. Let $\{M_1, \dots, M_p\}$ denote the connected components of $G[M]$. As M does not contain a $(k+1)$ -sized anti-matching, at most one of its connected component has size greater than k , we may assume w.l.o.g. that it is M_1 , if existent. Let \mathcal{C} be the $(k+1)$ -packing of $\{M_1, \dots, M_p\}$. As $|M_1| \leq 10k+2$ and $|M_i| \leq k$ for $2 \leq i \leq p$, we have that $|\mathcal{C}| \leq 10k+2$. Moreover, since M does not contain any $(k+1)$ -sized anti-matching, $|M \setminus \mathcal{C}| \leq k$ and thus $|M| \leq 11k+2$. This concludes the proof. \blacktriangleleft

3.3 Combs with large teeth

We now turn our attention to the case where a given comb contains many vertices in its teeth. The arguments are somewhat symmetric to the ones used in the proof of Lemma 15. The main difference lies in the fact that the information provided by unaffected vertices differ when they are contained in the teeth rather than in the shaft.

► **Rule 5.** Let (C, R) be a comb of G such that there exist three disjoint sets $\mathcal{R}_a, \mathcal{R}_b$ and \mathcal{R}_c where:

- \mathcal{R}_a is a $(2k+1)$ -packing of $\{R_1, \dots, R_l\}$,
- $\mathcal{R}_c = \{R_l, \dots, R_q\}$ is a $(2k+1)$ -packing of $\{R_l, \dots, R_1\}$,
- \mathcal{R}_b is a $(2k+1)$ -packing of $\{R_{q-1}, \dots, R_1\}$,

Remove $R' = R \setminus (\mathcal{R}_a \cup \mathcal{R}_b \cup \mathcal{R}_c)$ from G .

► **Lemma 21.** Rule 5 is safe.

Proof. Let $G' = G \setminus R'$ be the graph obtained after application of Rule 5. Since G' is an induced subgraph of G and since trivially perfect graphs are hereditary, any k -editing of G is a k -editing of G' .

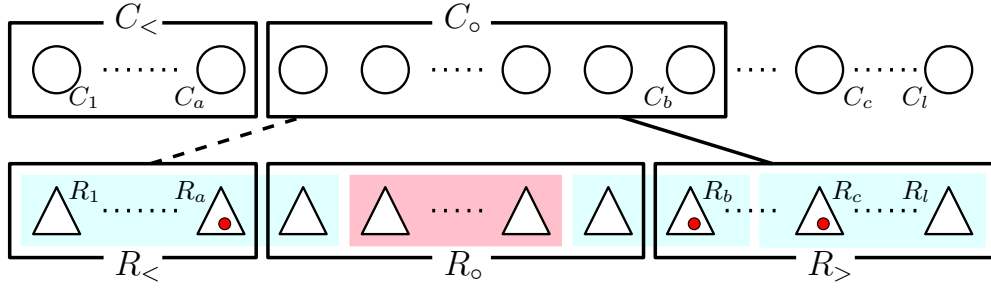
For the reverse direction, let F' be a k -editing of G' and $H' = G' \triangle F'$. We will construct a k -editing F^* of G . Let r_a, r_b and r_c be unaffected vertices in $\mathcal{R}_a, \mathcal{R}_b$ and \mathcal{R}_c , respectively. Note that these vertices exist as these sets contain at least $2k+1$ vertices and F' affects at most $2k$ vertices. Let R_a, R_b and R_c , $1 \leq a < b < c \leq l$, be the teeth of R containing r_a, r_b and r_b , respectively (these sets are well-defined since the packings $\mathcal{R}_a, \mathcal{R}_b$ and \mathcal{R}_c are disjoint). Moreover, since r_a, r_b and r_c are unaffected by F' their neighborhoods are equal in G' and H' and hence $(N_{H'}(r_a) \setminus R_a) \subseteq (N_{H'}(r_b) \setminus R_b) \subseteq (N_{H'}(r_c) \setminus R_c)$.

► **Claim 22.** The set $N_{H'}(r_b) \setminus R_b$ is a clique in H' .

Proof. Assume for a contradiction that $N_{H'}(r_b) \setminus R_b$ contains a non-edge $\{u, v\}$. Recall that there is no edge between R_b and R_c . Hence, since $(N_{H'}(r_b) \setminus R_b) \subseteq (N_{H'}(r_c) \setminus R_c)$ we have that the set $\{r_b, u, v, r_c\}$ induces a C_4 in H' , a contradiction. \triangleleft

Let $R_\circ = R_{a+1} \cup \dots \cup R_{b-1}$ and $C_\circ = C_{a+1} \cup \dots \cup C_b$. Similarly, let $C_< = C_1 \cup \dots \cup C_a$, $R_< = R_1 \cup \dots \cup R_a$ and $R_> = R_b \cup \dots \cup R_l$. Finally, let $G_\circ = G \setminus R_\circ$ and $H_\circ = H' \setminus R_\circ$. These sets are depicted Figure 3. Notice in particular that H_\circ is trivially perfect and that $R' \subseteq R_\circ$. Let $F_\circ \subseteq F'$ be the k -editing such that $H_\circ = G_\circ \triangle F_\circ$. We first modify F_\circ to obtain a k -editing of G_\circ where every vertex of C_\circ is affected uniformly.

► **Claim 23.** There exists a k -editing F^* of G_\circ such that, in $H^* = G_\circ \triangle F^*$, C_\circ is a clique module.



■ **Figure 3** Illustration of the comb and the sets used in the proof of Lemma 21. The circles are critical cliques of the shaft and the triangles are teeth. The red vertices correspond to r_a , r_b and r_c , the light blue rectangles correspond to sets \mathcal{R}_a , \mathcal{R}_b and \mathcal{R}_c and the light red rectangle corresponds to \mathcal{R}' , which is removed by Rule 5.

Proof. Note that C_o is a critical clique in G_o since $C \supset C_o$ is a module in $G \setminus R$ (Observation 14) and vertices of C_o are non-adjacent to vertices of $R_<$ and adjacent to vertices of $R_>$. Assume now that C_o is not a clique module in H_o and let $v_o \in C_o$ be a vertex contained in the least number of pairs of F_o . Consider the graph $H'_o = H_o \setminus (C_o \setminus \{v_o\})$, which is trivially perfect by heredity, and let H^* be the graph obtained from H'_o by adding vertices of $C_o \setminus \{v_o\}$ as true twins of v_o . Let F^* be the editing such that $H^* = G_o \triangle F^*$. The graph H^* is trivially perfect as the class of trivially perfect graphs is closed under true twin addition. It follows from construction that C_o is a clique module in H^* and by the choice of v_o , $|F^*| \leq |F_o|$. ◀

We henceforth consider $H^* = G_o \triangle F^*$ where F^* is the editing from Claim 23. We now show that vertices of R_o can be added into H^* while ensuring it remains trivially perfect.

▷ **Claim 24.** The graph $H = G \triangle F^*$ is trivially perfect.

Proof. We start by removing the vertices of $R_b \setminus \{r_b\}$ from H^* , which will give us more control on the neighborhood of r_b and ease some arguments. Let $\tilde{H} = H^* \setminus (R_b \setminus \{r_b\})$, this graph is trivially perfect by heredity. Let S be a maximal clique of \tilde{H} containing r_b . By Claim 22, $N_{\tilde{H}}(r_b)$ is a clique and since r_b is unaffected by F^* we have that $S = N_{\tilde{H}}[r_b] = C_< \cup C_o \cup V_p \cup \{r_b\}$. We use Lemma 3 on S to obtain a set of connected components $\{K_1, \dots, K_r\}$ of $\tilde{H} \setminus S$ such that $\{K_1, \dots, K_r\}$ are modules in \tilde{H} whose (possibly empty) neighborhoods in S form a nested family. We further split components $\{K_1, \dots, K_r\}$ into two types: K_i is an α -component if $N_{\tilde{H}}(K_i) \subseteq (N_{\tilde{H}}(r_a) \cap S)$ and a β -component otherwise, $1 \leq i \leq r$. Since $N_{\tilde{H}}(r_a) \cap S = V_p \cup C_<$ we have that, for any α -component K_α , $N_{\tilde{H}}(K_\alpha) \subseteq V_p \cup C_<$. Moreover, since $S = N_{\tilde{H}}[r_b]$ and since C_o is a clique module in \tilde{H} by Claim 23, every β -component K_β satisfies $N_{\tilde{H}}(K_\beta) = V_p \cup C_< \cup C_o = S \setminus \{r_b\}$.

Observe now that $(V_p \cup C_<) \subseteq N_G(R_o) \subseteq S \setminus \{r_b\}$. In other words, the neighborhood of any tooth of R_o contains the neighborhood of any α -component and is contained in the neighborhood of any β -component. Moreover the neighborhoods of the teeth of R_o are nested in G by definition of a comb. It follows that the vertices of R_o can be safely added to \tilde{H} with the same neighborhood as they have in G , ensuring that the resulting graph H_b is trivially perfect. It remains to add the vertices of R_b back into the graph. By Claim 22 and Proposition 9, the graph $H = H_b(r_b \rightarrow G[R_b])$ is trivially perfect. ◀

By Claim 24 the graph $H = G \triangle F^*$ is trivially perfect and as $|F^*| \leq k$, it follows that F^* is a k -editing of G , concluding the proof of Lemma 21. ◀

► **Lemma 25.** *Assume that the instance (G, k) is reduced under Rules 1–5. Let (C, R) be a comb of G . Then $|C \cup R| = O(k)$.*

Proof. First, note that $|C| \leq 6k + 2$ thanks to Observation 18. We proceed in the same fashion to bound the size of R . As the teeth of a comb are trivially perfect modules, Lemma 20 implies that $|R_i| \leq 11k + 2$, $1 \leq i \leq l$. Hence by Observation 5 any $(2k + 1)$ -packing of $\{R_1, \dots, R_l\}$ requires at most $13k + 2$ vertices. It follows that $|R| \leq 39k + 6$ since otherwise one could find three disjoint $(2k + 1)$ -packings of R that meet the requirements of Rule 5. Altogether we obtain that $|C \cup R| \leq 45k + 8$ which concludes the proof. ◀

3.4 Reducing the graph exhaustively

We conclude this section by showing that the graph can be reduced in polynomial time.

► **Lemma 26.** *There is a polynomial time algorithm that outputs an instance $G' = (V', E')$ such that none of Rules 1 to 5 applies.*

Proof. First, Rules 1 and 2 can be applied in polynomial time thanks to Lemma 10. We now need to apply the other rules on trivially perfect modules and combs. For the modules, it is sufficient to reduce *strong modules*, which are modules that do not overlap with other modules. We can enumerate strong modules in linear time [35]. For each strong module M we can check in polynomial time if it is trivially perfect. We can moreover check if M contains a $(k + 1)$ -sized anti-matching by finding a maximum matching in the complement graph $G[M]$, for instance using Edmonds' algorithm [17]. If M has a large anti-matching, then we can apply Rule 3. Otherwise, if $|M| \geq 11k + 2$ then it can be reduced using Rule 4. Indeed, $G[M]$ contains in this case at most one connected component M' with more than k vertices, such that $|M \setminus M'| \leq k$ (since otherwise M would contain a $(k + 1)$ -sized anti-matching). We compute a comb (C, R) through Lemma 19 in $G[M']$, with $|R| \leq 4k$. It follows that $|C| > 6k + 2$ and Observation 18 implies that Rule 3 applies.

It remains to show that the combs not included in a trivially perfect module can be reduced in polynomial time. In order to do this Dumas *et al.* [16] showed that so-called *critical combs* can be enumerated in polynomial time, a critical comb being an inclusion-wise maximal comb where $V_f \neq \emptyset$ and $R \cup C \cup V_f$ does not induce a trivially perfect module. In particular, critical combs contain every comb not included in a trivially perfect module. Hence it is sufficient to only reduce these combs. Given a critical comb, Rules 4 and 5 can be applied in polynomial time. This concludes the proof. ◀

Combining Theorem 1 and Lemmata 20, 25, and 26 we obtain the main result of this work.

► **Theorem 27.** *TRIVIALY PERFECT EDITING admits a kernel with $O(k^2)$ vertices.*

Proof. We give a formal proof of Theorem 1 for the sake of completeness. Note that most arguments and notations are extracted from the proof of [16, Theorem 1]. Recall that $c(k)$ and $p(k)$ are functions defined as, respectively, the maximum size of a trivially perfect module in and a comb of G in Theorem 1. Let $(G = (V, E), k)$ be a reduced yes-instance of TRIVIALY PERFECT EDITING and F a k -editing set of G . Let $H = G \Delta F$ and $\mathcal{T} = (T, \mathcal{B})$ the universal clique decomposition of H . The graph G is not necessarily connected, thus T is a forest. Let A be the set of nodes $t \in V(T)$ such that the bag B_t contains a vertex affected by F . Since $|F| \leq k$, we have $|A| \leq 2k$. Let $A' \subseteq V(T)$ be the least common ancestor closure of A plus the root of each connected component of T . The least common ancestor closure

is obtained as follows: start with $A' = A$ and while there is $u, v \in A'$ whose least common ancestor w (in T) is not in A' , add w to A' . According to [18, Lemma 1] the least common ancestor closure of A is of size at most $2|A|$. Moreover, Rule 1 implies that there are at most $2k$ connected components in H and thus $2k$ roots, hence $|A'| \leq 6k$.

Let D be a connected component of $T \setminus A'$. We can observe that, by construction of A' , only three cases are possible:

- $N_T(D) = \emptyset$ (D is a connected component of T).
 - $N_T(D) = \{a\}$ (D is a subtree of T whose parent is $a \in A'$).
 - $N_T(D) = \{a_1, a_2\}$ with one of the nodes $a_1, a_2 \in A'$ being an ancestor of the other in T .
- Dumas *et al.* [16] denote these connected components as respectively of type 0, 1 or 2. For $D \subseteq V(T)$, let $W(D) = \bigcup_{t \in D} B_t$ denote the set of vertices of G corresponding to bags of D .

There is no connected component of type 0 or else $W(D)$ would be a connected component of G inducing a trivially perfect graph. Rule 1 would have been applied to this component, contradicting the fact that G is a reduced instance.

Now consider the set of type 1 components D_1, D_2, \dots, D_r of $T \setminus A'$ attached in T to the same node $a \in A'$. Dumas *et al.* [16] showed that $W_a = W(D_1) \cup W(D_2) \cup \dots \cup W(D_r)$ is a trivially perfect module of G . By Lemma 20, we have $|W_a| = c(k)$. There are at most $|A'| \leq 6k$ such sets W_a , thus the set of vertices of G in bags of type 1 components is of size $O(k \cdot c(k))$.

Now consider the type 2 connected components D of $T \setminus A'$ which have two neighbors in T . Let a_1 and a_2 be these neighbors, one being the ancestor of the other, say a_1 is the ancestor of a_2 . Let $\{a_1, t_1, \dots, t_l, a_2\}$ be the path from a_1 to a_2 in the tree. The component D can be seen as a comb of shaft $(B_{t_1}, \dots, B_{t_l})$. More precisely, by construction of the universal clique decomposition, $W(D)$ can be partitioned into a comb (C, R) of H : the critical clique decomposition of C is $(C_1 = B_{t_1}, \dots, C_l = B_{t_l})$, and each R_i corresponds to the union of bags of the subtrees rooted at t_i which do not contain $B_{t_{i+1}}$, for $1 \leq i < l$, and to the union of bags of the subtrees rooted at t_l which do not contain a_2 , for $i = l$. Since (C, R) was not affected by F , it is also a comb of G . Thus for each type 2 component D , $W(D)$ contains $p(k)$ vertices. Since T is a forest, it can contain at most $|A'| - 1 \leq 6k - 1$ such components in $T \setminus A'$. Therefore the set of bags containing type 2 connected components of $T \setminus A'$ contains $O(k \cdot p(k))$ vertices.

It remains to bound the set of vertices of G which are in bags of A' . The vertices corresponding to nodes of $A' \setminus A$ are critical cliques of G , and are hence of size at most $k + 1$ by Rule 2. Thus the set of vertices in bags of $A' \setminus A$ is of size $O(k^2)$. We conclude by showing a similar bound for vertices in bags of A . Such vertices induce critical cliques in H but not necessarily in G . However, note that in each such critical clique the set of vertices not affected by F correspond to clique modules in G (not necessarily maximal). Such vertices are contained in exactly one critical clique of G and have thus been reduced by Rule 2. It follows that the set of affected critical cliques of H contains at most $2k + 2k \cdot (k + 1)$ vertices. Altogether we obtain that $|V(G)| = O(k^2 + k \cdot (p(k) + c(k)))$ which concludes the proof of Theorem 1. To obtain Theorem 27 we simply recall that Lemmata 20 and 25 imply that $c(k) = O(k)$ and $p(k) = O(k)$, respectively. ◀

4 The deletion variant

As mentioned in the introduction, a quadratic vertex-kernel is known to exist for TRIVIALY PERFECT COMPLETION [1]. The results presented Section 3 can be adapted to prove that TRIVIALY PERFECT DELETION also admits a quadratic vertex-kernel by simply replacing any mention of “editing” by “deletion”.

More precisely, one can see that in order to prove the safeness of Rules 3–5, the k -editing F^* for the original graph that is derived from a k -editing F' for the reduced instance only uses operations that were done by F' . In particular, if F' only contains non-edges then so does F^* , meaning that it is a valid solution. Together with the fact that Rules 1 and 2 are safe for the deletion variant, we obtain the following.

► **Theorem 28.** *TRIVIALY PERFECT DELETION admits a kernel with $O(k^2)$ vertices.*

We conclude by mentioning that Theorem 27 also holds for TRIVIALY PERFECT COMPLETION for the same reasons as the deletion variant. However, a vertex-kernel with $O(k^2)$ is already known for this problem [1]. Moreover, the constant factor on the number of vertices is smaller than the one of our kernel.

5 Conclusion

In this work we improved known kernelization algorithms for the TRIVIALY PERFECT EDITING and TRIVIALY PERFECT DELETION problems, providing a quadratic vertex-kernel for both of them. This matches the best known bound for the completion variant [1]. Improving upon these bounds is an appealing challenge that may require a novel approach. On the other hand, it would be interesting to develop lower bounds for kernelization on such problems. Finally, even if the use of unaffected vertices in the design of reduction rules is common, its combination with the structural properties of trivially perfect graphs in terms of their maximal cliques allowed us to design stronger reduction rules. We hope that the approach presented in this work may lead to finding or improving kernelization algorithms for some related problems. Let us for instance mention the cubic vertex-kernel for PROPER INTERVAL COMPLETION [3] and the quartic one for PTOLEMAIC COMPLETION [10] that might be appropriate candidates.

References

- 1 Gabriel Bathie, Nicolas Bousquet, Yixin Cao, Yuping Ke, and Théo Pierron. (Sub) linear kernels for edge modification problems toward structured graph classes. *Algorithmica*, 84(11):3338–3364, 2022. doi:10.1007/s00453-022-00969-1.
- 2 Stéphane Bessy, Christophe Paul, and Anthony Perez. Polynomial kernels for 3-leaf power graph modification problems. *Discrete Applied Mathematics*, 158(16):1732–1744, 2010. doi:10.1016/j.dam.2010.07.002.
- 3 Stéphane Bessy and Anthony Perez. Polynomial kernels for proper interval completion and related problems. *Information and Computation*, 231:89–108, 2013. doi:10.1016/j.ic.2013.08.006.
- 4 Ulrik Brandes, Michael Hamann, Luise Häuser, and Dorothea Wagner. Skeleton-based clustering by quasi-threshold editing. In *Algorithms for Big Data: DFG Priority Program 1736*, pages 134–151. Springer, 2023. doi:10.1007/978-3-031-21534-6_7.
- 5 Ulrik Brandes, Michael Hamann, Ben Strasser, and Dorothea Wagner. Fast quasi-threshold editing. In *Proceedings of the 23rd European Symposium on Algorithms, ESA*, pages 251–262, 2015. doi:10.1007/978-3-662-48350-3_22.
- 6 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.
- 7 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.

15:16 An Improved Kernelization Algorithm for Trivially Perfect Editing

- 8 Leizhen Cai and Yufei Cai. Incompressibility of H -free edge modification problems. *Algorithmica*, 71(3):731–757, 2015. doi:10.1007/s00453-014-9937-x.
- 9 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr 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.
- 10 Christophe Crespelle, Benjamin Gras, and Anthony Perez. Completion to chordal distance-hereditary graphs: a quartic vertex-kernel. In *Proceedings of the 47th International Workshop on Graph-Theoretic Concepts in Computer Science, WG*, pages 156–168, 2021. doi:10.1007/978-3-030-86838-3_12.
- 11 Christophe Crespelle, Rémi Pellerin, and Stéphan Thomassé. A quasi-quadratic vertex kernel for cograph edge editing, 2022. arXiv:2212.14814.
- 12 Pål Grønås Drange, Markus Fanebust Dregi, Daniel Lokshantov, and Blair D. Sullivan. On the threshold of intractability. *Journal of Computer and System Sciences*, 124:1–25, 2022. doi:10.1016/j.jcss.2021.09.003.
- 13 Pål Grønås Drange, Fedor V. Fomin, Michał Pilipczuk, and Yngve Villanger. Exploring the subexponential complexity of completion problems. *ACM Transactions on Computation Theory*, 7(4):1–38, 2015. doi:10.1145/2799640.
- 14 Pål Grønås Drange and Michał Pilipczuk. A polynomial kernel for trivially perfect editing. *Algorithmica*, 80(12):3481–3524, 2018. doi:10.1007/s00453-017-0401-6.
- 15 Maël Dumas, Anthony Perez, and Ioan Todinca. Polynomial kernels for strictly chordal edge modification problems. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation, IPEC*, pages 17:1–17:16, 2021. doi:10.4230/LIPIcs.IPEC.2021.17.
- 16 Maël Dumas, Anthony Perez, and Ioan Todinca. A cubic vertex-kernel for trivially perfect editing. *Algorithmica*, 85(4):1091–1110, 2023. doi:10.1007/s00453-022-01070-3.
- 17 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.
- 18 Fedor V. Fomin, Daniel Lokshantov, Neeldhara Misra, and Saket Saurabh. Planar F -deletion: Approximation, kernelization and optimal FPT algorithms. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 470–479, 2012. doi:10.1109/FOCS.2012.62.
- 19 Fedor V. Fomin, Daniel Lokshantov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.
- 20 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 21 Martin Charles Golumbic. Trivially perfect graphs. *Discrete Mathematics*, 24(1):105–107, 1978. doi:10.1016/0012-365X(78)90178-4.
- 22 Lars Gottesbüren, Michael Hamann, Philipp Schoch, Ben Strasser, Dorothea Wagner, and Sven Zühlsdorf. Engineering exact quasi-threshold editing. In *Proceedings of the 18th International Symposium on Experimental Algorithms, SEA*, pages 10:1–10:14, 2020. doi:10.4230/LIPIcs.SEA.2020.10.
- 23 Sylvain Guillemot, Frédéric Havet, Christophe Paul, and Anthony Perez. On the (non-)existence of polynomial kernels for P_t -free edge modification problems. *Algorithmica*, 65(4):900–926, 2013. doi:10.1007/s00453-012-9619-5.
- 24 Jiong Guo. Problem kernels for NP-complete edge deletion problems: Split and related graphs. In *Proceedings of the 18th International Symposium on Algorithms and Computation, ISAAC*, pages 915–926, 2007. doi:10.1007/978-3-540-77120-3_79.
- 25 Haim Kaplan, Ron Shamir, and Robert E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM Journal on Computing*, 28(5):1906–1922, 1999. doi:10.1137/S0097539796303044.
- 26 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972. doi:10.1007/978-1-4684-2001-2_9.

- 27 Christian Komusiewicz and Johannes Uhlmann. A cubic-vertex kernel for flip consensus tree. *Algorithmica*, 68(1):81–108, 2014. doi:10.1007/s00453-012-9663-1.
- 28 Stefan Kratsch and Magnus Wahlström. Two edge modification problems without polynomial kernels. *Discrete Optimization*, 10(3):193–199, 2013. doi:10.1016/j.disopt.2013.02.001.
- 29 Yunlong Liu, Jianxin Wang, Jie You, Jianer Chen, and Yixin Cao. Edge deletion problems: Branching facilitated by modular decomposition. *Theoretical Computer Science*, 573:63–70, 2015. doi:10.1016/j.tcs.2015.01.049.
- 30 Dániel Marx and R.B. Sandeep. Incompressibility of H-free edge modification problems: Towards a dichotomy. *Journal of Computer and System Sciences*, 125:25–58, 2022. doi:10.1016/j.jcss.2021.11.001.
- 31 James Nastos and Yong Gao. Familial groups in social networks. *Social Networks*, 35(3):439–450, 2013. doi:10.1016/j.socnet.2013.05.001.
- 32 Assaf Natanzon, Ron Shamir, and Roded Sharan. A polynomial approximation algorithm for the minimum fill-in problem. *SIAM Journal on Computing*, 30(4):1067–1079, 2000. doi:10.1137/S0097539798336073.
- 33 Jaroslav Nešetřil and Patrice Ossona De Mendez. On low tree-depth decompositions. *Graphs and combinatorics*, 31(6):1941–1963, 2015. doi:10.1007/s00373-015-1569-7.
- 34 Roded Sharan. *Graph modification problems and their applications to genomic research*. PhD thesis, Tel-Aviv University, 2002.
- 35 Marc Tedder, Derek Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In : *Proceedings of the 35th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 634–645, 2008. doi:10.1007/978-3-540-70575-8_52.
- 36 Jing-Ho Yan, Jer-Jeong Chen, and Gerard J. Chang. Quasi-threshold graphs. *Discrete applied mathematics*, 69(3):247–255, 1996. doi:10.1016/0166-218X(96)00094-7.

From Data Completion to Problems on Hypercubes: A Parameterized Analysis of the Independent Set Problem

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

Sebastian Ordyniak  

School of Computing, University of Leeds, UK

Stefan Szeider  

Algorithms and Complexity Group, TU Wien, Austria

Abstract

Several works have recently investigated the parameterized complexity of data completion problems, motivated by their applications in machine learning, and clustering in particular. Interestingly, these problems can be equivalently formulated as classical graph problems on induced subgraphs of powers of partially-defined hypercubes.

In this paper, we follow up on this recent direction by investigating the Independent Set problem on this graph class, which has been studied in the data science setting under the name Diversity. We obtain a comprehensive picture of the problem's parameterized complexity and establish its fixed-parameter tractability w.r.t. the solution size plus the power of the hypercube.

Given that several such FO-definable problems have been shown to be fixed-parameter tractable on the considered graph class, one may ask whether fixed-parameter tractability could be extended to capture all FO-definable problems. We answer this question in the negative by showing that FO model checking on induced subgraphs of hypercubes is as difficult as FO model checking on general graphs.

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

Keywords and phrases Independent Set, Powers of Hypercubes, Diversity, Parameterized Complexity, Incomplete Data

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.16

Funding *Robert Ganian*: Robert Ganian acknowledges support from Project No. Y1329 of the Austrian Science Fund (FWF).

Iyad Kanj: Iyad Kanj acknowledges support from DePaul University through URC grant 602061.

Sebastian Ordyniak: Project EP/V00252X/1 of the Engineering and Physical Sciences Research Council (EPSRC).

Stefan Szeider: Stefan Szeider acknowledges support from Project No. P36420 of the Austrian Science Fund (FWF).



© Eduard Eiben, Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider; licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 16; pp. 16:1–16:14

Leibniz International Proceedings in Informatics



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

1 Introduction

Recently, there has been an increasing interest in studying the parameterized complexity of clustering problems motivated by their applications in machine learning [2, 3, 4, 5, 6, 17, 19, 20, 25, 26, 29, 39, 40], particularly their applications to fundamental clustering problems [1, 28, 41, 44]. In many of these clustering problems, we are given a set of d -dimensional vectors over the Boolean/binary domain, where the vectors are regarded as rows of a matrix. It is worth noting that due to the applications of such problems in incomplete-data settings, a number of past works on the topic also studied settings where some of the entries in these vectors are unknown [30, 19, 20, 29, 17, 8, 9, 10, 21, 37]. The objective is to determine if these vectors (or, in the incomplete-data setting, their completions) satisfy some desirable clustering properties. Examples of such properties include admitting a partitioning into k clusters each of diameter (or radius) at most r (for some given $k, r \in \mathbb{N}$), or admitting a k -cluster of diameter (or radius) at most r , where the distance under consideration is typically the Hamming distance [12, 16, 19, 20, 23, 32, 33, 34, 35]; here, a k -cluster of diameter r is a set of k points which have pairwise distance of at most r .

As it turns out, many of these well-studied clustering problems can be formulated as classical graph problems on induced subgraphs of powers of the hypercube graph. For instance, finding a cluster of diameter at most $r \in \mathbb{N}$, for a given r , is equivalent to the CLIQUE problem defined on the subgraph of the r -th power of the hypercube that is induced by the subset of hypercube vertices corresponding to the given input vectors. Similarly, partitioning the set of vectors into k clusters each of diameter at most r , for some given $r, k \in \mathbb{N}$, is equivalent to the partitioning into k cliques problem on the same graph class, whereas partitioning the set of vectors into clusters, each of radius at most r with respect to some vector in the set, is equivalent to the k -dominating set problem on the same graph class described above. We remark that, to the best of our knowledge, this graph class is not a subclass of commonly studied graph classes and has not been considered in previous works pertaining to algorithmic upper or lower bounds for graph-theoretic problems.

Contribution. In this paper, we study the parameterized complexity of another classical graph problem defined on induced subgraphs of powers of the hypercube: the INDEPENDENT SET problem. In the context of data analytics, the problem arises when studying the “diversity” of a given set of vectors, a notion that can be viewed as the opposite of minimising the number of clusters in a cluster partitioning of the set of vectors (in fact, in the area of data analytics this is studied directly under the nomenclature *diversity* or dispersion [11, 31, 45]). More precisely, motivated by the aforementioned extensive interest in the analysis of incomplete data, we focus on the more general incomplete data setting. We refer to this problem as POW-HYP-IS-COMPLETION: given a set of Boolean vectors with some missing entries and integers k and r , the goal is to complete the missing entries so that the resulting set of vectors contains a subset S of k vectors such that the Hamming distance between each pair is at least $r + 1$ (or to correctly determine that such a set does not exist).

The main contribution of this paper is a complete characterisation of the parameterized complexity of POW-HYP-IS-COMPLETION w.r.t. the two parameters k and r : we provide a fixed-parameter algorithm for POW-HYP-IS-COMPLETION when parameterized by $k + r$, and complement this positive result with intractability results for the cases where any of these two parameters is dropped. In particular, we show that the problem is NP-complete already for $r = 2$ – that is, the problem is paraNP-hard parameterized by r , and W[1]-hard parameterized by k alone. Interestingly, the FPT result shows that the parameterized complexity of the

problem is independent of any restrictions on the number or the structure of the missing entries in the input vectors – contrasting many of the previous results on clustering incomplete data [30, 19, 20, 29]. We remark that even the fixed-parameter tractability of the problem in the complete data setting (i.e., where all entries are known) is non-obvious, but follows as an immediate corollary of our result.

For our final contribution, we revisit the observation that several of the complete-data clustering problems recently considered in the literature (e.g., see [19, 20]) reduce to well-known graph problems on the class of induced subgraphs of powers of the hypercube. Since it was shown that all of these graph problems are fixed-parameter tractable when restricted to this graph class and the graph problems are expressible in First Order Logic (FO), a natural question to ask is whether these FPT results can be generalised to any graph problem expressible in FO logic. We resolve this question in the negative.

Related Work. The problem of computing the diversity of a data set, which forms the underpinning of our study of POW-HYP-IS-COMPLETION, has been studied in a variety of different contexts and settings. For instance, Ceccarelo, Pietracaprina, Pucci and Upfal studied approximation algorithms for the problem [11]. Gawrychowski, Krasnopolosky, Mozes, and Weimann obtained a linear-time algorithm for the problem when the data set is represented as a tree [31], improving upon the previous polynomial-time algorithm of Bhattacharya and Houle [7]. Sacharidis, Mehta, Skoutas, Patroumpas and Voisard provided heuristics for dynamic versions of the problem [45].

More broadly, there is extensive work on problems arising in the context of incomplete data. Hermelin and Rozenberg [38] studied the CLOSEST STRING WITH WILDCARDS problem, which can be seen as the problem of finding a data completion and a center to a minimum-radius cluster containing all the data points. Koana, Froese and Niedermeier [39] recently revisited the earlier work of Hermelin and Rozenberg [38] and obtained, among other results, a fixed-parameter algorithm for that problem parameterized by the radius plus the maximum number of missing entries per row; see also the related work of the same authors [40]. Eiben et al. considered a number of different clustering problems in the presence of incomplete data [18, 19], and a subset of these authors previously investigated the fundamental MATRIX COMPLETION problem in the same setting [30]. The parameterized complexity of k -means clustering on incomplete data was investigated by Eiben et al. [17] and Ganian et al. [29].

2 Preliminaries

Problem Terminology and Definition

Let \vec{a} and \vec{b} be two vectors in $\{0, 1, \square\}^d$, where \square is used to represent coordinates whose value is unknown (i.e., missing entries). We denote by $\Delta(\vec{a}, \vec{b})$ the set of coordinates in which \vec{a} and \vec{b} are guaranteed to differ, i.e., $\Delta(\vec{a}, \vec{b}) = \{i \mid (\vec{a}[i] = 1 \wedge \vec{b}[i] = 0) \vee (\vec{a}[i] = 0 \wedge \vec{b}[i] = 1)\}$, and we denote by $\delta(\vec{a}, \vec{b})$ the *Hamming distance* between \vec{a} and \vec{b} measured only between known entries, i.e., $|\Delta(\vec{a}, \vec{b})|$. Moreover, for a subset $D' \subseteq [d]$ of coordinates, we denote by $\vec{a}[D']$ the vector \vec{a} restricted to the coordinates in D' .

Let $M \subseteq \{0, 1\}^d$ and let $[d] = \{1, \dots, d\}$. For a vector $\vec{a} \in M$ and $t \in \mathbb{N}$, we denote by $N_t(\vec{a})$ the t -*Hamming neighbourhood* of \vec{a} , i.e., the set $\{\vec{b} \in M \mid \delta(\vec{a}, \vec{b}) \leq t\}$ and by $N_t(M)$ the set $\bigcup_{\vec{a} \in M} N_t(\vec{a})$. We say that $M^* \subseteq \{0, 1, \square\}^d$ is a *completion* of $M \subseteq \{0, 1, \square\}^d$ if there is a bijection $\alpha : M \rightarrow M^*$ such that for all $\vec{a} \in M$ and all $i \in [d]$ it holds that either $\vec{a}[i] = \square$ or $\alpha(\vec{a})[i] = \vec{a}[i]$.

We now proceed to give the formal definition of the problem under consideration:

POW-HYP-IS-COMPLETION

Input: A set M with elements from $\{0, 1, \square\}^d$ and $k, r \in \mathbb{N}$.
 Question: Is there a completion M^* of M and a subset S of M^* with $|S| = k$ such that, for any two vectors $a, b \in S$, we have $\delta(a, b) \geq r + 1$?

Observe that in a matrix representation of the above problem, we can represent the input matrix as a *set* of vectors where each row of the matrix corresponds to one element in our set.

We remark that even though the statements are given in the form of decision problems, all tractability results presented in this paper are constructive and the associated algorithms can also output a solution (when it exists) as a witness, along with the decision. In the case where we restrict the input to vectors over $\{0, 1\}^d$ (*i.e.*, where all entries are known), we omit “-COMPLETION” from the problem name.

Parameterized Complexity

The basic motivation behind parameterized complexity is to find a parameter that describes the structure of the problem instance such that the combinatorial explosion can be confined to this parameter. More formally, a *parameterized problem* Q is a subset of $\Omega^* \times \mathbb{N}$, where Ω is a fixed finite alphabet. Each instance of Q is a pair (I, κ) , where $\kappa \in \mathbb{N}$ is called the *parameter*. A parameterized problem Q is *fixed-parameter tractable* (FPT) [24, 14, 13], if there is an algorithm, called an *FPT-algorithm*, that decides whether an input (I, κ) is a member of Q in time $f(\kappa) \cdot |I|^{\mathcal{O}(1)}$, where f is a computable function and $|I|$ is the input instance size. The class FPT denotes the class of all fixed-parameter tractable parameterized problems.

A parameterized problem Q is *FPT-reducible* to a parameterized problem Q' if there is an algorithm, called an *FPT-reduction*, that transforms each instance (I, κ) of Q into an instance (I', κ') of Q' in time $f(\kappa) \cdot |I|^{\mathcal{O}(1)}$, such that $\kappa' \leq g(\kappa)$ and $(I, \kappa) \in Q$ if and only if $(I', \kappa') \in Q'$, where f and g are computable functions. Based on the notion of FPT-reducibility, a hierarchy of parameterized complexity, *the W-hierarchy* $= \bigcup_{t \geq 0} W[t]$, where $W[t] \subseteq W[t+1]$ for all $t \geq 0$, has been introduced, in which the 0-th level $W[0]$ is the class FPT. The notions of hardness and completeness have been defined for each level $W[i]$ of the W-hierarchy for $i \geq 1$ [14, 13]. It is commonly believed that $W[1] \neq \text{FPT}$ (see [14, 13]). The $W[1]$ -hardness has served as the main working hypothesis of fixed-parameter intractability. A problem is *paraNP-hard* if it is NP-hard for a constant value of the parameter [24].

Sunflowers

A *sunflower* in a set family \mathcal{F} is a subset $\mathcal{F}' \subseteq \mathcal{F}$ such that all pairs of elements in \mathcal{F}' have the same intersection.

► **Lemma 1** ([22, 24]). *Let \mathcal{F} be a family of subsets of a universe U , each of cardinality exactly b , and let $a \in \mathbb{N}$. If $|\mathcal{F}| \geq b!(a-1)^b$, then \mathcal{F} contains a sunflower \mathcal{F}' of cardinality at least a . Moreover, \mathcal{F}' can be computed in time polynomial in $|\mathcal{F}|$.*

3 The Parameterized Complexity of POW-HYP-IS-COMPLETION

Our aim for POW-HYP-IS-COMPLETION is to establish fixed-parameter tractability parameterized by $k + r$ (*i.e.*, regardless of the structure or number of missing entries). As our first step, we show that all rows in an arbitrary instance (M, k, r) can be, w.l.o.g., assumed to contain at most $\mathcal{O}(k \cdot r)$ many \square 's.

Next, we observe that if M is sufficiently large and the r -Hamming neighbourhood of each vector is upper-bounded by a function of $k + r$, then – since the number of \square 's is bounded – (M, k, r) is a YES-instance. The argument here is analogous to the classical argument showing that INDEPENDENT SET is trivial on large bounded-degree graphs.

On a high level, we would now like to find and remove an “irrelevant vector” from M – since here the number of \square 's on *every* row is bounded, any instance reduced in this way to only contain a bounded number of vectors can be solved via a brute-force fixed-parameter procedure. However, finding an irrelevant vector is rather challenging, primarily because the occurrence of \square 's is not restricted. Instead, we develop a more powerful set representation \mathcal{F}' for vectors in the instance which also uses elements to keep track of the presence of \square 's in the neighbours of \vec{v} . We can then apply the Sunflower Lemma to find a sufficiently-large sunflower in \mathcal{F}' , and in the core of the proof we argue that (1) such a sunflower consists of at most a bounded number of “important petals” (which can be identified in polynomial time), and (2) any petal that is not important represents an irrelevant vector.

3.1 Dealing with Unstructured Missing Data

In this subsection, we design an algorithm for POW-HYP-IS-COMPLETION which remains efficient even when the number and placement of unknown entries is not explicitly restricted on the input.

We begin with a simple lemma that allows us to deal with vectors (*i.e.*, rows) with a large number of missing entries. For brevity, let a k -diversity set be a set containing k vectors which have pairwise Hamming distance at least $r + 1$.

► **Lemma 2.** *Let $\mathcal{I} = (M, k, r)$ be an instance of POW-HYP-IS-COMPLETION where $k \geq 1$ and let $\vec{v} \in M$ be a vector containing more than $(k - 1) \cdot (r + 1)$ -many \square 's. Then \mathcal{I} is a YES-instance if and only if $\mathcal{I}' = (M \setminus \{\vec{v}\}, k - 1, r)$ is a YES-instance. Moreover, a completion and k -diversity set for \mathcal{I} can be computed from a completion and $(k - 1)$ -diversity set for \mathcal{I}' in linear time.*

Proof. The forward direction is trivial: for any completion M^* of M and k -diversity set S in M^* , we can obtain a $(k - 1)$ -diversity set and completion for \mathcal{I}' by simply removing \vec{v} from M^* and S .

For the backward direction, consider a completion M'^* of $M' = M \setminus \vec{v}$ and a $(k - 1)$ -diversity set $S = \{\vec{s}_1, \dots, \vec{s}_{k-1}\}$ in M'^* . Let us choose an arbitrary set C of $(k - 1) \cdot (r + 1)$ coordinates in \vec{v} that all contain \square , and let us then partition C into k -many subsets $\alpha_1, \dots, \alpha_k$ each containing precisely $r + 1$ coordinates. Now consider the vector \vec{v}^* obtained from \vec{v} as follows:

- for each $i \in [k - 1]$ and every coordinate $j \in \alpha_i$, set $\vec{v}^*[j]$ to the opposite value of $\vec{s}_i[j]$ (*i.e.*, $\vec{v}^*[j] = 1$ if and only if $\vec{s}_i[j] = 0$);
- for every other coordinate j of \vec{v}^* , we set $\vec{v}^*[j] = \vec{v}[j]$ if $\vec{v}[j] \neq \square$ and $\vec{v}^*[j] = 0$ otherwise.

Clearly, $M^* = M'^* \cup \{\vec{v}^*\}$ is a completion of M . Moreover, since \vec{v}^* differs from each vector in S in at least $r + 1$ coordinates, $S \cup \{\vec{v}^*\}$ is a k -diversity set in M^* . ◀

Next, we show that instances which are sufficiently large and where each vector only “interferes with” a bounded number of other vectors are easy to solve. For ease of presentation, let $\zeta(k, r, t) = 3^{(k-1) \cdot (r+1)} \cdot t! \cdot \left((k - 1) \cdot \left(3(k - 1) \cdot (r + 1) + r + t \right) \right)^t$ be the exact meaning of “sufficiently large” in this case.

16:6 From Data Completion to Problems on Hypercubes

► **Lemma 3.** *Let $\mathcal{I} = (M, k, r)$ be an instance of POW-HYP-IS-COMPLETION. If $|M| \geq k \cdot r \cdot \zeta(k, r, r)$ and $|N_t(\vec{v})| < \zeta(k, r, t)$ for every $\vec{v} \in M$ and $t \leq r$, then a k -diversity set in \mathcal{I} can be found in polynomial time.*

Proof. One can find a solution to \mathcal{I} by iterating the following greedy procedure k times: choose an arbitrary vector \vec{v} , add it into a solution, and delete all other vectors with Hamming distance at most r from \vec{v} . By the bound on $|N_t(\vec{v})|$, each choice of \vec{v} will only lead to the deletion of at most $r \cdot \zeta(k, r, r)$ vectors from M . Moreover, since δ measures the Hamming distance only between known entries, *any* completion of the missing entries can only increase (and never decrease) the Hamming distance between vectors. Hence, the size of M together with the bounded size of the Hamming neighbourhood of \vec{v} guarantee that this procedure will find a solution of cardinality k in \mathcal{I} which will remain valid for every completion of M . ◀

We can now move on to the main part of the proof: a procedure which either outputs a solution outright or finds an irrelevant vector.

► **Lemma 4.** *Let $\mathcal{I} = (M, k, r)$ be an instance of POW-HYP-IS-COMPLETION such that $|N_t(\vec{v})| \geq \zeta(k, r, t)$ for some vector $\vec{v} \in M$ and $t \leq r$ and such that each vector in M contains at most $(k-1) \cdot (r+1)$ \square 's. There is a polynomial-time procedure that finds a vector $\vec{f} \in M$ satisfying the following properties:*

- (M, k, r) is a YES-instance if and only if $\mathcal{I}' = (M \setminus \{\vec{f}\}, k, r)$ is a YES-instance, and
- A completion and diversity set for \mathcal{I} can be computed from a solution and diversity set for \mathcal{I}' in linear time.

Proof. We will begin by constructing a set system over the neighbourhood of \vec{v} . Let $Z = \{z \in [d] \mid \vec{v}[z] = \square\}$ be the set of coordinates where \vec{v} is incomplete. Clearly, since $|N_t(\vec{v})| \geq 3^{(k-1) \cdot (r+1)} \cdot t! \cdot \left((k-1) \cdot (3(k-1) \cdot (r+1) + r + t) \right)^t$ and $|Z| \leq (k-1) \cdot (r+1)$, we can find a subset $N \subseteq N_t(\vec{v})$ of vectors whose cardinality is at least $t! \cdot \left((k-1) \cdot (3(k-1) \cdot (r+1) + r + t) \right)^t$ such that all vectors in N are the same on the coordinates in Z , i.e., $\forall \vec{x}, \vec{y} \in N : \forall z \in Z : \vec{x}[z] = \vec{y}[z]$.

Now, let F be a set containing 2 elements for each coordinate $j \in [d] \setminus Z$ of vectors in M : the element \square_j and the element D_j . We construct a set system \mathcal{F} over F as follows: for each vector $\vec{x} \in N$, we add a set \hat{x} to \mathcal{F} that contains:

- \square_j if and only if $\vec{x}[j] = \square$, and
- D_j if and only if $\vec{x}[j] \neq \vec{v}[j]$.

Observe that, since \vec{x} contains at most $(k-1) \cdot (r+1)$ \square 's by assumption and since \vec{x} differs from \vec{v} in at most t -many completed coordinates, every set in \mathcal{F} has cardinality at most $(k-1) \cdot (r+1) + t$. This means we can apply Lemma 1 to find a sunflower \mathcal{F}' in \mathcal{F} of cardinality at least $(k-1) \cdot (3(k-1) \cdot (r+1) + r + t) + 1$; for ease of presentation, we will identify the elements of \mathcal{F}' with the vectors they represent. Let \vec{f} be an arbitrarily chosen vector from \mathcal{F}' ; we claim that \vec{f} satisfies the properties claimed in the lemma, and to complete the proof it suffices to establish this claim.

The backward direction is trivial: if \mathcal{I}' is a YES-instance then clearly \mathcal{I} is a YES-instance as well. It is also easy to observe that a completion and diversity set for \mathcal{I} can be computed from a solution and diversity set for \mathcal{I}' in linear time (adding a vector does not change the validity of a solution). What we need to show is that if \mathcal{I} is a YES-instance, then so is \mathcal{I}' (i.e., $(M \setminus \{\vec{f}\}, k, r)$); moreover, this final claim clearly holds if \mathcal{I} admits a solution that does not contain \vec{f} .

So, assume that M admits a completion M^* which contains a k -diversity set $S = \{\vec{f}, \vec{s}_1, \dots, \vec{s}_{k-1}\}$. Let C be the core of the sunflower \mathcal{F}' , and note that all vectors in \mathcal{F}' have precisely the same content in the coordinates in C .

Finding a replacement for \vec{f} . We would now like to argue that, for some completion which we will define later, \mathcal{F}' contains a vector that can be used to replace \vec{f} in the solution.

Let $\vec{s}_i \in S$ be an arbitrary vector. First, let us consider the case that, in M , \vec{s}_i differs from \vec{v} in more than $3(k-1) \cdot (r+1) + r + t$ coordinates (*i.e.*, $\vec{v}[j] \neq \vec{s}_i[j]$ in M for at least $3(k-1) \cdot (r+1) + r + t$ choices of j). Then *every* vector in \mathcal{F}' will have Hamming distance greater than r from \vec{s}_i *regardless of the completion*.

Indeed, for every vector $f' \in \mathcal{F}'$ there are at most $3(k-1) \cdot (r+1)$ coordinates j such that at least one of $\vec{v}[j]$, $\vec{s}_i[j]$, \vec{f}' , meaning that there are at least $r + t$ *other* coordinates where \vec{v} differs from \vec{s}_i and which are guaranteed to be complete – and since $\delta(\vec{f}', \vec{v}) = t$, \vec{f}' it must hold that $\delta(\vec{f}', \vec{s}_i) > r$ (by the triangle inequality). Hence indeed every vector in \mathcal{F}' must have distance at least $r + 1$ from \vec{s}_i , and in this case we will create a set $S_i = \emptyset$ (the meaning of this will become clear later).

Now, consider the converse case, *i.e.*, that \vec{s}_i differs from \vec{v} in at most $3(k-1) \cdot (r+1) + r + t$ coordinates. We may now extend the sunflower \mathcal{F}' by adding a set representation of \vec{s}_i , *i.e.*, a set Q_i which contains \square_j if and only if $\vec{s}_i[j] = \square$ and D_j if and only if $\vec{s}_i[j] \neq \vec{v}[j]$ (for all $j \in [d] \setminus Z$). Observe that $|Q_i| \leq 3(k-1) \cdot (r+1) + r + t$, and in particular $Q_i \setminus C$ intersects with at most $3(k-1) \cdot (r+1) + r + t$ elements of \mathcal{F}' . Let S_i be the set of all such elements, *i.e.*, elements of \mathcal{F}' which have a non-empty intersection with Q_i outside of the core (formally, with $Q_i \setminus C$).

To conclude the proof, we will show that there is a completion M^* of M' such that any arbitrarily chosen vector \vec{f}' in the non-empty set $\mathcal{F}' \setminus (\{\vec{f}\} \cup \bigcup_{i \in [k-1]} S_i)$ can replace \vec{f} in the k -diversity set S .

Arguing Replaceability. Consider a new completion M^* of $M \setminus \vec{f}$ obtained as follows:

- For each vector $\vec{w} \in \mathcal{F}' \setminus S$, we complete
 1. the \square 's in $C \cup Z$ precisely in the same way as \vec{f} , and
 2. for every other \square at coordinate j , we set $\vec{w}[j] = -(\vec{v}[j] - 1)$ (*i.e.*, to the opposite of \vec{v} ; recall that $\vec{v}[j] \neq \square$ since $j \notin Z$);
- all other \square 's in all other vectors in $M \setminus \vec{f}$ are completed in precisely the same way as in M^* .

Since M^* precisely matches M^* on all vectors in $S \setminus \vec{f}$, it follows that $S \setminus \vec{f}$ is a $(k-1)$ -diversity set in M^* . Moreover, consider for a contradiction that $\delta(\vec{f}', \vec{s}_i) \leq r$ for some $\vec{s}_i \in S$ *after completion*, *i.e.*, in M^* . Then clearly \vec{s}_i could not differ from \vec{v} in more than $3(k-1) \cdot (r+1) + r + t$ coordinates in M' , since – as we already argued – in this case every vector in \mathcal{F}' will have Hamming distance greater than r from \vec{s}_i regardless of the completion.

Hence, we must be in the case where \vec{s}_i differed from \vec{v} in at most $3(k-1) \cdot (r+1) + r + t$ coordinates in M' . Now consider how $\delta(\vec{f}', \vec{s}_i)$ differs from $\delta(\vec{f}, \vec{s}_i)$. First of all, there is no difference between these two distances on the coordinates in $Z \cup C$ due to our construction of M^* and choice of N . For the remaining coordinates, we will consider separately the set X of coordinates in the petals of \vec{f} and \vec{f}' (*i.e.*, the set $\{j \in [d] \setminus (Z \cup C) \mid \vec{f}[j] \neq \vec{v}[j] \vee \vec{f}'[j] \neq \vec{v}[j]\}$), and the set $Y = [d] \setminus (C \cup Z \cup X)$ of all remaining coordinates. It follows that $\vec{v}[j] = \vec{f}[j] = \vec{f}'[j]$ for all coordinates $j \in Y$, and hence there is no difference between the two distances on these coordinates either.

So, all that is left is to consider the difference between $\delta(\vec{f}', \vec{s}_i)$ and $\delta(\vec{f}, \vec{s}_i)$ on the coordinates in X . Among these coordinates, \vec{f} can only differ from \vec{s}_i in *at most* $t - |C|$ many coordinates – notably in the coordinates of its own petal – because the coordinates in the petal of \vec{f}' do not intersect with Q_i . On the other hand, our construction guarantees that \vec{f}' differs from \vec{s}_i in *at least* $t - |C|$ coordinates in X ; more precisely, on all coordinates in the petal of \vec{f}' , since on these coordinates (1) \vec{s}_i is equal to \vec{v} and (2) \vec{f}' differs from \vec{v} .

In summary, we conclude that $\delta(\vec{f}', \vec{s}_i) \geq \delta(\vec{f}, \vec{s}_i)$ and hence $(S \setminus \{\vec{f}\}) \cup \{\vec{f}'\}$ is a k -diversity set in M'^* , as claimed. ◀

We can now establish our main result for POW-HYP-IS-COMPLETION.

► **Theorem 5.** POW-HYP-IS-COMPLETION *is fixed-parameter tractable parameterized by $k + r$.*

Proof. The algorithm proceeds as follows. Given an instance $\mathcal{I} = (M, k, r)$ of POW-HYP-IS-COMPLETION, it first checks whether M contains a vector with more than $(k - 1) \cdot (r + 1)$ \square 's; if yes, it applies Lemma 2 and restarts on the reduced instance. Second, it checks whether $|M| \geq k \cdot r \cdot \zeta(k, r, r)$; if not, it uses the fact that the number of \square 's and the number of rows is bounded by a function of the parameter to find a completion and a k -diversity set in \mathcal{I} (or determine that one does not exist) by brute force.

Third, it checks whether each vector \vec{v} satisfies $|N_t(\vec{v})| < \zeta(k, r, t)$ for every $t \in [r]$; if yes, then it solves \mathcal{I} by invoking Lemma 3. Otherwise, it invokes Lemma 4 to reduce the cardinality of M by 1 and restarts. If the algorithm eventually terminates with a “NO”, then we know that the initial input was a NO-instance; otherwise, it will output a solution which can be transformed into a solution for the original input by the used lemmas. ◀

3.2 Lower Bounds

► **Theorem 6.** POW-HYP-IS *is NP-complete and W[1]-hard parameterized by k .*

Proof. We prove both NP-hardness and W[1]-hardness results by giving a polynomial-time FPT reduction from INDEPENDENT SET (IS), which is W[1]-hard [14].

Let (G, k) be an instance of IS, where $V(G) = \{v_1, \dots, v_n\}$, and let $m = E(G)$. Fix an arbitrary ordering $\mathcal{O} = (e_1, \dots, e_m)$ of the edges in $E(G)$.

For each vertex $v_i \in V(G)$, define a vector $\vec{a}_i \in \{0, 1\}^m$ by setting $\vec{a}_i[j] = 1$ if v_i is incident to e_j and $\vec{a}_i[j] = 0$ otherwise. Now expand the set of coordinates of these vectors by adding to each of them $n(n - 1)$ new coordinates, $n - 1$ coordinates for each v_i , $i \in [n]$; we refer to the $n - 1$ (extra) coordinates of v_i as the “private” coordinates of v_i . For each v_i , $i \in [n]$, set $n - 1 - \deg(v_i)$ many coordinates among the private coordinates of v_i to 1, and all other new coordinates of v_i to 0. Let $M = \{\vec{a}_i \mid i \in [n]\}$ be the set of expanded vectors, where $\vec{a}_i \in \{0, 1\}^{m+n(n-1)}$, for $i \in [n]$. The reduction from IS to POW-HYP-IS produces the instance $\mathcal{I} = (M, k, 2n - 4)$ of POW-HYP-IS; clearly, this reduction is a polynomial-time FPT-reduction.

Observe that, for any two distinct vertices $v_i, v_j \in V(G)$, $\delta(\vec{a}_i, \vec{a}_j) = 2n - 2$ if v_i and v_j are nonadjacent and $\delta(\vec{a}_i, \vec{a}_j) = 2n - 4$ if v_i and v_j are adjacent.

The proof that (G, k) is a YES-instance of IS iff $(M, k, 2n - 4)$ is a YES-instance of POW-HYP-IS is now straightforward. ◀

► **Theorem 7.** POW-HYP-IS *is NP-complete even when $r = 2$.*

Proof. We reduce from the INDEPENDENT SET problem (which is NP-complete). Let (G, k) be an instance of INDEPENDENT SET and let G' be the graph obtained from G after subdividing every edge exactly twice. We first observe that G has an independent set of size at least k if and only if G' has an independent set of size at least $|E(G)| + k$. This is because if $I \subseteq V(G)$ is an independent set of G , then we can add one of the subdivision vertices for every edge of G because I does not contain both endpoints of an edge. On the other hand, if $I \subseteq V(G')$ is an independent set of G' , then we can assume without loss of generality that I does not contain both endpoints of an edge in G because we could easily transform I into an independent set of the same size by replacing one of the endpoints of such an edge with a subdivided vertex.

Next we construct an instance $\mathcal{I} = (M, |E(G)| + k, 2)$ of POW-HYP-IS in polynomial-time such that G' has an independent set of size at least $|E(G)| - k$ if and only if \mathcal{I} is a YES-instance. We set $d = 2|V(G)|$ and obtain M as follows. Let $V(G) = \{v_1, \dots, v_n\}$. For every $v_i \in V(G)$, we add the vector \vec{v}_i that is 1 at the two coordinates i and $i + 1$ and otherwise 0. Moreover, for every $e = v_i v_j \in E(G)$, we add the vector e^1 that is 1 at the coordinates i , $i + 1$, and j and the vector e^2 that is 1 at the coordinates j , $j + 1$, and i . This completes the construction of \mathcal{I} . The equivalence now follows because two vectors in M have distance at most $r = 2$ if and only if their corresponding vertices in G' are adjacent; here e^1 and e^2 correspond to the two subdivision vertices on the edge e . ◀

4 On Graph Problems on Induced Subgraphs of the Hypercubes

In this section, we discuss the implications of the results in the previous section for fundamental problems defined on induced subgraphs of powers of the hypercube graph.

In particular, the d -dimensional hypercube graph is the graph Q_d whose vertex set is the set of all Boolean d -dimensional vectors, and two vertices are adjacent if and only if their two vectors differ in precisely 1 coordinate. We can then define the class \mathcal{Q}_d^r as the class of all graphs that are induced subgraphs of the r -th power of Q_d . We note that, in line with the commonly used definition of hypercube graphs [15, 27], we consider the vertices in \mathcal{Q}_d^r to be vectors and hence every graph $G \in \mathcal{Q}_d^r$ contains an explicit characterisation of its vertices as vectors.

In this setting, it is straightforward to observe that POW-HYP-IS is precisely the INDEPENDENT SET problem on \mathcal{Q}_d^r . Moreover, the clustering problems IN-CLUSTERING, DIAM-CLUSTERING, and LARGE DIAM-CLUSTER considered in [19, 20] are precisely the DOMINATING SET, PARTITION INTO CLIQUES, and CLIQUE problems, respectively, on \mathcal{Q}_d^r . Therefore, all the upper and lower bound results derived in this paper and in [19, 20] pertaining to these clustering problems hold true for their corresponding graph problems on \mathcal{Q}_d^r .

▶ **Corollary 8.** *Given $r, d, k \in \mathbb{N}$ and a graph $G \in \mathcal{Q}_d^r$, determining whether G has a:*

- *dominating set of size k is FPT parameterized by $k + r$;*
- *partition into k cliques is FPT parameterized by $k + r$;*
- *independent set of size k is FPT parameterized by $k + r$;*
- *clique of size k is FPT parameterized by r .*

We note that all the tractability results outlined in Corollary 8 are tight, which follows from the lower-bound results obtained in Section 3.2 and in [19, 20], in the sense that dropping any parameter from our parameterizations leads to an intractable problem.

16:10 From Data Completion to Problems on Hypercubes

Observing that three of the graph properties in the problems discussed above are expressible in First Order Logic (FO) and result in FO formulas whose length is a function of the parameter k , an interesting question that ensues from the above discussion is whether these positive results can be extended to the generic problem of First-Order Model Checking [43, 36], formalised below. We will show next that the answer to this question is negative – and, in fact, remains negative even when we restrict ourselves to induced subgraphs of hypercubes (*i.e.*, for $r = 1$).

Q-FO-MODEL-CHECKING

Input:	A first-order (FO) formula ϕ , integers d, r , and a graph $G \in \mathcal{Q}_d^r$.
Parameter:	$ \Phi $
Question:	Does $G \models \Phi$?

We denote by FO-MODEL-CHECKING the general FO Model Checking problem on graphs, *i.e.*, \mathcal{C} -FO-MODEL-CHECKING with \mathcal{C} being the class of all graphs.

► **Lemma 9.** *Let H be an arbitrary graph. There is a graph $G \in \mathcal{Q}_{|V(H)|+|E(H)|}^1$ such that G is isomorphic to the graph H' obtained from H after subdividing every edge of H exactly once and attaching a leaf to every vertex resulting from a subdivision. Moreover, G can be computed from H in polynomial time.*

Proof. Let $n = |V(H)|$ and $m = |E(H)|$. To prove the lemma, we construct a matrix representation $M \in \{0, 1\}^{n+m}$ of H' which has one row (vector) for every vertex in H and where two vertices in H' are adjacent if and only if their corresponding rows in M have Hamming distance at most 1. Let v_1, \dots, v_n be an arbitrary ordering of the vertices of H , and e_1, \dots, e_m be an arbitrary ordering of its edges. Then, M contains one row r_i for every $i \in [n]$ that is 1 at its i -th entry and 0 at all other entries. Moreover, for every edge $e_\ell = \{v_i, v_j\} \in E(H)$, M contains the following two rows:

- the row r_e (corresponding to the degree-3 vertex in H' obtained from e) that is 1 at the i -th and j -th entries, and 0 at all other entries; and
- the row r'_e (corresponding to the leaf in H' obtained from e) that is 1 at the i -th, j -th, and $(n + \ell)$ -th entries, and 0 at all other entries.

This completes the construction of M . Clearly, two rows in M have Hamming distance at most one if and only if their corresponding vertices in H' are adjacent, as required. ◀

► **Theorem 10.** *Q-FO-MODEL-CHECKING is $W[t]$ -hard for every $t \in \mathbb{N}^*$.*

Proof. We give a parameterized reduction from FO MODEL CHECKING, which is $W[t]$ -hard for every $t \in \mathbb{N}^*$. Let $\mathcal{I} := (\Phi, H)$ be an instance of FO MODEL CHECKING. We will show the theorem by constructing the equivalent instance $\mathcal{I}' := (\Phi', G)$ such that $G \in \mathcal{Q}_d^1$ and $|\Phi| \leq f(|\Phi'|)$ for some computable function f and value d that is polynomially bounded in the input size. G is obtained from H in the same manner as in Lemma 9. Moreover, Φ' is obtained from Φ as follows:

- Let $\phi_V(x)$ be the formula that holds for a variable x if and only if x corresponds to one of the original vertices in G , *i.e.*, $\phi_V(x) := \forall y E(x, y) \exists z \neq x \wedge E(y, z)$;
- replace every subformula of the form $\exists x \phi$ (for some variable x and some subformula ϕ of Φ) with the formula $\exists x \phi_V(x) \wedge \phi$;
- replace every subformula of the form $\forall x \phi$ (for some variable x and some subformula ϕ of Φ) with the formula $\forall x \phi_V(x) \rightarrow \phi$; and
- replace every atom $E(x, y)$, where E is the adjacency predicate and x and y are variables, with the formula $\exists s E(x, s) \wedge E(s, y) \wedge x \neq y$.

It is straightforward now to show that $H \models \Phi$ if and only if $G \models \Phi'$, and that $|\Phi'| \leq 20|\Phi|$. Moreover, because of Lemma 9, $G' \in \mathcal{Q}_d^1$, as required. ◀

5 Conclusion

In this paper, we studied the parameterized complexity of the classical INDEPENDENT SET problem on induced subgraphs of powers of hypercubes, but with the additional complication that the “positions” of the vertices in the hypercube representation may be partially unknown. We considered the two most natural parameters for the problem: the size k of the independent set and the power r of the hypercube, and provided a complete characterisation of the problem’s complexity w.r.t. k and r . We also performed a meta-investigation of the parameterized complexity of graph problems on this graph class that are expressible in FO logic and showed the existence of such problems that are parameterized intractable.

A natural future direction of our work is to study the parameterized complexity of other graph problems on this class, in particular those that have applications in clustering. One famous open problem that comes to mind is the p -center problem [16, 32]. The problem can be formulated similarly to the above setting, with the exception of allowing the selection of vertices to be from the whole hypercube, as opposed to restricting them to the input subgraph. In particular, the well-known p -centers problem reduces to the k -dominating set problem in the r -th power of the hypercube graph, but where the k vertices in the dominating set are not restricted to the input subgraph, but can be chosen from \mathcal{Q}_d . This problem was shown to be FPT parameterized by $k + r$ [20]. An intriguing NP-hard restriction of the problem is the problem slice corresponding to $p = 1$, or what is known as the 1-center problem, or equivalently, the CLOSEST STRING problem [32, 42]. The parameterized complexity of the problem parameterized by each of k and r alone remain important open questions.

References

- 1 Charu C. Aggarwal and Chandan K. Reddy. *Data Clustering: Algorithms and Applications*. Chapman & Hall/CRC, 1st edition, 2013.
- 2 Sayan Bandyapadhyay, Fedor V. Fomin, Petr A. Golovach, William Lochet, Nidhi Purohit, and Kirill Simonov. How to find a good explanation for clustering? In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 3904–3912. AAAI Press, 2022.
- 3 Sayan Bandyapadhyay, Fedor V. Fomin, Petr A. Golovach, Nidhi Purohit, and Kirill Simonov. FPT approximation for fair minimum-load clustering. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPICs*, pages 4:1–4:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- 4 Sayan Bandyapadhyay, Fedor V. Fomin, Petr A. Golovach, Nidhi Purohit, and Kirill Simonov. Lossy kernelization of same-size clustering. In Alexander S. Kulikov and Sofya Raskhodnikova, editors, *Computer Science - Theory and Applications - 17th International Computer Science Symposium in Russia, CSR 2022, Virtual Event, June 29 - July 1, 2022, Proceedings*, volume 13296 of *Lecture Notes in Computer Science*, pages 96–114. Springer, 2022.
- 5 Sayan Bandyapadhyay, Fedor V. Fomin, Petr A. Golovach, and Kirill Simonov. Parameterized complexity of feature selection for categorical data clustering. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPICs*, pages 14:1–14:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

- 6 Sayan Bandyopadhyay, Fedor V. Fomin, and Kirill Simonov. On coresets for fair clustering in metric and Euclidean spaces and their applications. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPICs*, pages 23:1–23:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 7 Binay K. Bhattacharya and Michael E. Houle. Generalized maximum independent sets for trees in subquadratic time. In Alok Aggarwal and C. Pandu Rangan, editors, *Algorithms and Computation, 10th International Symposium, ISAAC '99, Chennai, India, December 16-18, 1999, Proceedings*, volume 1741 of *Lecture Notes in Computer Science*, pages 435–445. Springer, 1999.
- 8 Emmanuel J. Candès and Yaniv Plan. Matrix completion with noise. *Proceedings of the IEEE*, 98(6):925–936, 2010.
- 9 Emmanuel J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717–772, 2009.
- 10 Emmanuel J. Candès and Terence Tao. The power of convex relaxation: near-optimal matrix completion. *IEEE Trans. Information Theory*, 56(5):2053–2080, 2010.
- 11 Matteo Ceccarello, Andrea Pietracaprina, Geppino Pucci, and Eli Upfal. MapReduce and streaming algorithms for diversity maximization in metric spaces of bounded doubling dimension. *PVLDB*, 10(5):469–480, 2017.
- 12 Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. *Journal of Computer and System Sciences*, 68(2):417–441, 2004.
- 13 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 14 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.
- 15 Tomáš Dvořák and Petr Gregor. Hamiltonian paths with prescribed edges in hypercubes. *Discrete Mathematics*, 307(16):1982–1998, 2007.
- 16 M.E Dyer and A.M Frieze. A simple heuristic for the p -centre problem. *Oper. Res. Lett.*, 3(6):285–288, 1985.
- 17 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 2021, Virtual Conference, January 10 - 13, 2021*, pages 2649–2659. SIAM, 2021.
- 18 Eduard Eiben, Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. The parameterized complexity of clustering incomplete data. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, pages 7296–7304. AAAI Press, 2021. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/16896>, doi:10.1609/aaai.v35i8.16896.
- 19 Eduard Eiben, Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. Finding a cluster in incomplete data. 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 47:1–47:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- 20 Eduard Eiben, Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. On the parameterized complexity of clustering problems for incomplete data. *Journal of Computer and System Sciences*, 134:1–19, 2023.
- 21 Ehsan Elhamifar and René Vidal. Sparse subspace clustering: Algorithm, theory, and applications. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(11):2765–2781, 2013.
- 22 Paul Erdős and Richard Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 1(1):85–90, 1960.
- 23 Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC '88*, pages 434–444. ACM, 1988.

- 24 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer, Berlin, 2006.
- 25 Fedor V. Fomin, Petr A. Golovach, Tanmay Inamdar, Nidhi Purohit, and Saket Saurabh. Exact exponential algorithms for clustering problems. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPICs*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- 26 Fedor V. Fomin, Petr A. Golovach, and Kirill Simonov. Parameterized k -clustering: Tractability island. *J. Comput. Syst. Sci.*, 117:50–74, 2021.
- 27 John P. Hayes Frank Harary and Horng-Jyh Wu. A survey of the theory of hypercube graphs. *Comput. Math. Appl.*, 15(4):277–289, 1988.
- 28 Guojun Gan, Chaoqun Ma, and Jianhong Wu. *Data clustering - theory, algorithms, and applications*. SIAM, 2007.
- 29 Robert Ganian, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. The complexity of k -means clustering when little is known. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 6960–6987, 2022.
- 30 Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. Parameterized algorithms for the matrix completion problem. In *ICML*, volume 80 of *JMLR Workshop and Conference Proceedings*, pages 1642–1651, 2018.
- 31 Pawel Gawrychowski, Nadav Krasnopolosky, Shay Mozes, and Oren Weimann. Dispersion on Trees. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.
- 32 Leszek Gąsieniec, 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*, pages 905–906, 1999.
- 33 Leszek Gąsieniec, Jesper Jansson, and Andrzej Lingas. Approximation algorithms for Hamming clustering problems. *Journal of Discrete Algorithms*, 2(2):289–301, 2004.
- 34 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
- 35 Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for CLOSEST STRING and related problems. *Algorithmica*, 37(1):25–42, 2003.
- 36 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017.
- 37 Moritz Hardt, Raghu Meka, Prasad Raghavendra, and Benjamin Weitz. Computational limits for matrix completion. In *Proceedings of The 27th Conference on Learning Theory*, volume 35 of *JMLR Workshop and Conference Proceedings*, pages 703–725. JMLR.org, 2014.
- 38 Danny Hermelin and Liat Rozenberg. Parameterized complexity analysis for the closest string with wildcards problem. *Theoretical Computer Science*, 600:11–18, 2015.
- 39 Tomohiro Koana, Vincent Froese, and Rolf Niedermeier. Parameterized algorithms for matrix completion with radius constraints. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, June 17-19, 2020, Copenhagen, Denmark*, volume 161 of *LIPICs*, pages 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 40 Tomohiro Koana, Vincent Froese, and Rolf Niedermeier. The complexity of binary matrix completion under diameter constraints. *J. Comput. Syst. Sci.*, 132:45–67, 2023.
- 41 Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2nd edition, 2014.
- 42 Ming Li, Bin Ma, and Lusheng Wang. On the closest string and substring problems. *J. ACM*, 49(2):157–171, 2002.

16:14 From Data Completion to Problems on Hypercubes

- 43 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.
- 44 Boris Mirkin. *Clustering For Data Mining: A Data Recovery Approach*. Chapman & Hall/CRC, 2005.
- 45 Dimitris Sacharidis, Paras Mehta, Dimitrios Skoutas, Kostas Patroumpas, and Agnès Voisard. Selecting representative and diverse spatio-textual posts over sliding windows. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management, SSDBM 2018, Bozen-Bolzano, Italy, July 09-11, 2018*, pages 17:1–17:12, 2018.

Approximate Monotone Local Search for Weighted Problems

Bariş Can Esmer ✉ 

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
Saarbrücken Graduate School of Computer Science, Saarland Informatics Campus, Germany

Ariel Kulik ✉ 

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Dániel Marx ✉ 

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Daniel Neuen ✉ 

University of Bremen, Germany

Roohani Sharma ✉ 

Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

In a recent work, Esmer et al. describe a simple method – Approximate Monotone Local Search – to obtain exponential approximation algorithms from existing parameterized exact algorithms, polynomial-time approximation algorithms and, more generally, parameterized approximation algorithms. In this work, we generalize those results to the weighted setting.

More formally, we consider monotone subset minimization problems over a weighted universe of size n (e.g., VERTEX COVER, d -HITTING SET and FEEDBACK VERTEX SET). We consider a model where the algorithm is only given access to a subroutine that finds a solution of weight at most $\alpha \cdot W$ (and of arbitrary cardinality) in time $c^k \cdot n^{\mathcal{O}(1)}$ where W is the minimum weight of a solution of cardinality at most k . In the unweighted setting, Esmer et al. determine the smallest value d for which a β -approximation algorithm running in time $d^n \cdot n^{\mathcal{O}(1)}$ can be obtained in this model. We show that the same dependencies also hold in a weighted setting in this model: for every fixed $\varepsilon > 0$ we obtain a β -approximation algorithm running in time $\mathcal{O}((d + \varepsilon)^n)$, for the same d as in the unweighted setting.

Similarly, we also extend a β -approximate brute-force search (in a model which only provides access to a membership oracle) to the weighted setting. Using existing approximation algorithms and exact parameterized algorithms for weighted problems, we obtain the first exponential-time β -approximation algorithms that are better than brute force for a variety of problems including WEIGHTED VERTEX COVER, WEIGHTED d -HITTING SET, WEIGHTED FEEDBACK VERTEX SET and WEIGHTED MULTICUT.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Mathematics of computing → Approximation algorithms

Keywords and phrases parameterized approximations, exponential approximations, monotone local search

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.17

Related Version *Full Version:* <https://arxiv.org/abs/2308.15306>

Funding *Ariel Kulik:* This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 852780-ERC (SUBMODULAR).

Dániel Marx: Research supported by the European Research Council (ERC) consolidator grant No. 725978 SYSTEMATICGRAPH.



© Barış Can Esmer, Ariel Kulik, Dániel Marx, Daniel Neuen, and Roohani Sharma; licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 17; pp. 17:1–17:23

Leibniz International Proceedings in Informatics



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

1 Introduction

In this work, we are interested in *subset problems*, where the goal is to find a subset of a given n -sized universe U that satisfies some property Π (e.g., VERTEX COVER, HITTING SET, FEEDBACK VERTEX SET, MULTICUT). Such problems can trivially be solved in time $\mathcal{O}^*(2^n)$ ¹, and in the past decades there has been great interest in designing algorithms that beat this exhaustive search and run in time $\mathcal{O}^*(d^n)$ for as small $1 < d < 2$ as possible (see, e.g., [16]). On the other hand, many of the considered problems admit polynomial-time α -approximation algorithms for some constant $\alpha > 1$ (e.g., VERTEX COVER admits a polynomial-time 2-approximation [5]). To bridge the gap between exact exponential-time algorithms and polynomial-time α -approximation algorithms for some possibly large constant α , there has been recent interest in *exponential-time approximation algorithms* [2, 4, 6, 9–12, 19] to obtain approximation ratios that are better than what is considered possible in polynomial time.

In a recent work, Esmer et al. [12] describe a simple method – Approximate Monotone Local Search – to obtain exponential-time approximation algorithms for certain subset problems from existing parameterized exact algorithms, polynomial-time approximation algorithms and, more generally, parameterized approximation algorithms. More precisely, the focus in [12] lies on *subset minimization problems* where, given a universe U with n elements, we are aiming to find a set $S \subseteq U$ of minimum cardinality satisfying some property Π . To allow for approximation algorithms, we restrict to *monotone* problems, i.e., the family \mathcal{F} of solution sets is closed under taking supersets. In this setting, a β -approximation algorithm is asked to return a solution set $S \in \mathcal{F}$ such that $|S| \leq \beta \cdot |\text{OPT}|$ where OPT denotes a solution of minimum size. Given access to a parameterized α -approximation algorithm running in time $\mathcal{O}^*(c^k)$ (where the parameter k denotes the size of the desired optimal solution), Esmer et al. [12] determine the best possible value $d = \text{amls}(\alpha, c, \beta)$ such that a β -approximation algorithm running in time $\mathcal{O}^*(d^n)$ can be obtained. Using existing parameterized approximation algorithms, which in particular include polynomial-time approximation and exact parameterized algorithms, this leads to the fastest exponential-time approximation algorithms for a variety of problems including VERTEX COVER, d -HITTING SET, FEEDBACK VERTEX SET and ODD CYCLE TRANSVERSAL.

In this work, we are interested in *weighted* monotone subset minimization problems. Here, the universe U is additionally equipped with a weight function $\mathbf{w}: U \rightarrow \mathbb{N}$ and we are asking for a solution set $S \in \mathcal{F}$ of minimum weight $\mathbf{w}(S) := \sum_{u \in S} \mathbf{w}(u)$. Accordingly, in a β -approximation algorithm, we are seeking a solution set $S \in \mathcal{F}$ such that $\mathbf{w}(S) \leq \beta \cdot \mathbf{w}(\text{OPT})$ where OPT denotes a solution of minimum weight. Looking at [12], it can be observed that the obtained algorithms only extend to the weighted setting for the special case $\alpha = \beta = 1$. Indeed, in this particular case Fomin, Gaspers, Lokshantov and Saurabh [15] already show in an earlier work that an exact parameterized algorithm running in time $\mathcal{O}^*(c^k)$ can be turned into an exact exponential-time algorithm running time $\mathcal{O}^*((2 - \frac{1}{c})^n)$. As already pointed out in [15], the result also holds in a weighted setting and implies exact exponential-time algorithms for, e.g., d -HITTING SET and FEEDBACK VERTEX SET. On the other hand, for $\beta > 1$, the algorithm presented in [12] does not work in a weighted setting. In a nutshell, the main idea in [12] is to randomly sample a set of vertices X and to bound the probability that it contains a sufficiently large portion of an optimum solution OPT. However, in a weighted setting, such an approach is destined to fail since even adding a single element of large weight to an optimum solution may lead to an unbounded approximation factor.

¹ The \mathcal{O}^* notation hides polynomial factors in the expression.

The main contribution of this work is to adapt the tools from [12] to the weighted setting. Given a parameterized α -approximation algorithm running in time $\mathcal{O}^*(c^k)$ for a weighted subset minimization problem (where the parameter k still denotes the *size* of the desired optimal solution, we give additional details in the next paragraph), for every fixed $\varepsilon > 0$ we obtain a β -approximation algorithm running in time $\mathcal{O}((\mathbf{a}\mathbf{m}\mathbf{l}\mathbf{s}(\alpha, c, \beta) + \varepsilon)^n)$. Note that this matches the corresponding bound in the unweighted setting up to the additive ε in the base of the exponent in the running time.

To state our main result more precisely, let us formalize the requirements for the parameterized α -approximation algorithm. Similar to [12], we require an α -approximate extension algorithm. Such an algorithm receives as input a set $X \subseteq U$ and a number $k \geq 0$. If there is an extension $S \subseteq U$ to a solution set (i.e., $X \cup S \in \mathcal{F}$) of size at most k , then the algorithm outputs a set $T \subseteq U$ such that $X \cup T \in \mathcal{F}$ and $\mathbf{w}(T) \leq \alpha \cdot \mathbf{w}(S^*)$ where S^* is a minimum-weight extension of X of size at most k . Note that the size of T does not need to be bounded in k ; the parameter k only restricts the size of an “optimum solution” which we compare against. For example, the polynomial-time 2-approximation algorithm for WEIGHTED VERTEX COVER [5] immediately results in a 2-approximate extension algorithm: given a graph G , $X \subseteq V(G)$ and $k \geq 0$, we apply the 2-approximation algorithm to $G - X$ and return the output $T \subseteq V(G)$ (this algorithm behaves independently of k). With this, our main result can informally be stated as follows.

Suppose a weighted monotone subset minimization problem admits an α -approximate extension algorithm running in time $\mathcal{O}^*(c^k)$. Then there is a β -approximation algorithm running in time $\mathcal{O}^*((\mathbf{a}\mathbf{m}\mathbf{l}\mathbf{s}(\alpha, c, \beta) + \varepsilon)^n)$ for every $\varepsilon > 0$.

The basic idea to achieve this result is to partition the universe U into subsets U_i of elements of roughly the same weight. We apply the results from [12] to each of the sets U_i separately which results in a “query set” for U_i , i.e., a set of queries made by the algorithm from [12] to the α -approximate extension algorithm. The crucial observation is that these queries are made in a non-adaptive way, i.e., the “query set” only depends on the set U_i . We then combine the “query sets” for the blocks U_i into a query set for the whole weighted set U . In particular, the results from [12] are only used in a black-box manner.

For many problems such as VERTEX COVER [5], d -HITTING SET [5] and FEEDBACK VERTEX SET [3], polynomial-time approximation algorithms directly extend to the weighted setting, and provide α -approximate extension algorithms as discussed above. On the other hand, while many parameterized exact algorithms do not directly extend to the weighted setting, several problems have been studied in the weighted setting. For example, for WEIGHTED VERTEX COVER [21] provides a $\mathcal{O}^*(1.363^k)$ algorithm that, given a vertex-weighted graph and a number $k \geq 0$, returns a vertex cover of weight at most W where W is the minimum weight of a vertex cover of size at most k (if there is no vertex cover of size at most k , the algorithm reports failure). As before, to obtain a 1-approximate extension subroutine, we apply the algorithm to the graph $G - X$ (where X is the input set for the extension subroutine). Similar results are also available for WEIGHTED d -HITTING SET [14, 21] and WEIGHTED FEEDBACK VERTEX SET [1]. Also, some simple branching algorithms immediately extend to the weighted setting (e.g., deletion to \mathcal{H} -free graphs where \mathcal{H} is a finite set of forbidden induced subgraphs). Finally, we can also rely on parameterized approximation algorithms. For example, [18] provides such algorithms for several problems including WEIGHTED DIRECTED FVS and WEIGHTED MULTICUT.

We remark that there are also FPT algorithms for other parameterizations in the weighted setting. For example, Niedermeier and Rossmanith [20] show that WEIGHTED VERTEX COVER is fixed-parameter tractable parameterized by the weight W of a minimum-weight vertex cover. However, such subroutines are not useful in our setting since we aim to bound the running time of our approximation algorithms with respect to the number of vertices.

Similar to [12], we compare our algorithms to a brute-force search. Here, we consider a setting where the weighted monotone subset minimization problem can only be accessed via a membership oracle, i.e., given a set $X \subseteq U$, we can test (in polynomial time) whether X is a solution set. For every $\beta \geq 1$ we define $\mathbf{brute}(\beta) := 1 + \exp\left(-\beta \cdot \mathcal{H}\left(\frac{1}{\beta}\right)\right)$ where $\mathcal{H}(\beta) := -\beta \ln \beta - (1 - \beta) \ln(1 - \beta)$ denotes the entropy function. In [11] it has been shown that, in the unweighted setting, there is a β -approximation algorithm running in time $\mathcal{O}^*((\mathbf{brute}(\beta))^n)$ that only exploits a membership test. We also extend this result to the weighted setting, i.e., we show that any weighted monotone subset minimization problem can be solved in time $(\mathbf{brute}(\beta))^{n+o(n)}$ given only a membership oracle.

Esmer et al. [12] show that $\mathbf{aml}(\alpha, c, \beta) < \mathbf{brute}(\beta)$ for all $\alpha, c \geq 1$ and $\beta > 1$. Since the same bounds are achieved in the weighted setting, all algorithms obtained above are strictly faster than the brute-force β -approximation algorithm. In particular, we obtain exponential-time approximation algorithms that are faster than the approximate brute-force search for the weighted versions of VERTEX COVER, d -HITTING SET, FEEDBACK VERTEX SET, TOURNAMENT FVS, SUBSET FVS, CLUSTER GRAPH VERTEX DELETION, COGRAPH VERTEX DELETION, SPLIT VERTEX DELETION, PARTIAL VERTEX COVER, DIRECTED FEEDBACK VERTEX SET, DIRECTED SUBSET FVS, DIRECTED ODD CYCLE TRANSVERSAL and MULTICUT (all problems are defined in Appendix B).

Organization

The paper is organized as follows. In Section 2 we state the problems we want to address, provide the necessary definitions and notation, and formally state our main results. In Section 3, we demonstrate how our methods can be applied to specific problems to obtain exponential-time approximation algorithms. Section 4 contains the proof of Theorem 3, our result on exponential-time approximation algorithms for the WEIGHTEDSM problem in the membership model. Similarly, Section 5 contains the proof of Theorem 8, our result on exponential-time approximation algorithms for the WEIGHTEDSM problem in the extension model. Finally, in Section 6 we conclude the paper by summarizing our main contributions and key findings.

2 Our Results

To formally state our results we use an abstract notion of a problem and oracle-based computational models. Let U be a finite set of elements. We use n to denote the cardinality of U . A set system \mathcal{F} of U is a family $\mathcal{F} \subseteq 2^U$ of subsets of U . We say the set system \mathcal{F} is *monotone* if (i) $U \in \mathcal{F}$ and (ii) for all $T \subseteq S \subseteq U$, if $T \in \mathcal{F}$ then $S \in \mathcal{F}$ as well.

An instance of the *Weighted Monotone Subset Minimization problem* (WEIGHTEDSM) is a triplet $(U, \mathbf{w}, \mathcal{F})$ where U is a finite set, $\mathbf{w}: U \rightarrow \mathbb{N}$ is a weight function over the elements of U , and \mathcal{F} is a monotone set system of U . The set of solutions is \mathcal{F} and the objective is to find $S \in \mathcal{F}$ with minimum total weight $\mathbf{w}(S) := \sum_{u \in S} \mathbf{w}(u)$. We use $\mathbf{opt}(U, \mathbf{w}, \mathcal{F}) := \min\{\mathbf{w}(S) \mid S \in \mathcal{F}\}$ to denote the optimum value of a solution to the WEIGHTEDSM instance $(U, \mathbf{w}, \mathcal{F})$. We refer to the special case in which $\mathbf{w}(u) = 1$ for all $u \in U$ as the *Unweighted Monotone Subset Minimization problem* (UNWEIGHTEDSM).

The Weighted Monotone Subset Minimization problem is a meta-problem which captures multiple well studied problems as special cases, e.g., WEIGHTED VERTEX COVER, WEIGHTED FEEDBACK VERTEX SET and WEIGHTED MULTICUT. We study the problem using two computational models. In both models the set U and the weight function \mathbf{w} are given as part of the input. The set \mathcal{F} can only be accessed using an oracle, and the models differ in the type of supported oracle queries.

2.1 Membership Oracles and Weighted Approximate Brute Force

In the *membership model* the input to the algorithm is a universe U and a weight function $\mathbf{w}: U \rightarrow \mathbb{N}$. Additionally, the algorithm has access to a membership oracle for a monotone set system \mathcal{F} of U , that is, the algorithm can check if a subset $S \subseteq U$ satisfies $S \in \mathcal{F}$ in a single step. For every $\alpha \geq 1$, we say an algorithm is an α -approximation for WEIGHTEDSM (UNWEIGHTEDSM) in the membership model if for every WEIGHTEDSM (UNWEIGHTEDSM) instance $(U, \mathbf{w}, \mathcal{F})$ the algorithm returns a set $S \in \mathcal{F}$ such that $\mathbf{w}(S) \leq \alpha \cdot \text{opt}(U, \mathbf{w}, \mathcal{F})$.

One can easily attain a 1-approximation algorithm for WEIGHTEDSM in the membership model by iterating over all subsets of U and querying the oracle for each. This leads to an algorithm with running time $\mathcal{O}^*(2^n)$. Moreover, it can be easily shown there is no 1-approximation algorithm for WEIGHTEDSM (or for UNWEIGHTEDSM) in the membership model which runs in time $\mathcal{O}((2 - \varepsilon)^n)$. We refer to this algorithm as the (exact) brute force.

Intuitively, for every $\alpha > 1$, it should be possible to design an α -approximation algorithm for WEIGHTEDSM and UNWEIGHTEDSM in the membership model which runs in time $\mathcal{O}(c^n)$ for some $c < 2$. However, the value of the optimal c in this setting is not obvious. In [11] the authors studied UNWEIGHTEDSM in the membership model and pinpointed the right value of c . For every $\alpha \geq 1$ we define

$$\mathbf{brute}(\alpha) = 1 + \exp\left(-\alpha \cdot \mathcal{H}\left(\frac{1}{\alpha}\right)\right), \quad (1)$$

where $\mathcal{H}(x) = -x \ln(x) - (1 - x) \ln(1 - x)$ is the entropy function.

► **Lemma 1** ([11, Theorem 5.1]). *For every $\alpha \geq 1$ the following holds.*

1. *There is a deterministic α -approximation algorithm for UNWEIGHTEDSM in the membership model which runs in time $(\mathbf{brute}(\alpha))^n \cdot n^{\mathcal{O}(1)}$.*
2. *Let $\varepsilon > 0$. There is no α -approximation algorithm for UNWEIGHTEDSM in the membership model which runs in time $(\mathbf{brute}(\alpha) - \varepsilon)^n \cdot n^{\mathcal{O}(1)}$.*

As the algorithmic result in Lemma 1 can be viewed as an approximate analogue of the brute-force algorithm, it is commonly referred as α -approximate brute force. The lower bound given in [11] also holds if the algorithm is allowed to use randomization. As WEIGHTEDSM is a generalization of UNWEIGHTEDSM, the following corollary is an immediate consequence of Lemma 1.

► **Corollary 2.** *For every $\alpha \geq 1$ and $\varepsilon > 0$ there is no α -approximation algorithm for WEIGHTEDSM in the membership model which runs in time $(\mathbf{brute}(\alpha) - \varepsilon)^n \cdot n^{\mathcal{O}(1)}$.*

As the bound in Lemma 1 also holds if randomization is allowed, the same holds true for the bound in Corollary 2. Our first result is a generalization of the approximate brute-force algorithm of [11] for the weighted setting. That is, we provide an algorithm which matches the lower bound in Corollary 2 up to a sub-exponential factor.

► **Theorem 3** (Weighted Approximate Brute Force). *For every $\alpha > 1$ there is an α -approximation algorithm for WEIGHTEDSM in the membership model which runs in time $(\text{brute}(\alpha))^{n+o(n)}$.*

The proof of Theorem 3 is based on a rounding of the weight function \mathbf{w} and utilizes a construction from [11] which is applied to each of the rounded weight classes.

2.2 Extension Oracles and Weighted Approximate Monotone Local Search

Our second computational model deals with *extension oracles*. The input for these oracles is a set $S \subseteq U$ and a number $\ell \in \mathbb{N}_{\geq 0}$ and the output is an *extension* of S , that is, a set $X \subseteq U$ such that $S \cup X \in \mathcal{F}$. Furthermore, the returned set X is guaranteed to have a small weight in comparison to the minimum-weight extension of S which contains at most ℓ elements. For multiple problems, such as VERTEX COVER and FEEDBACK VERTEX SET, these oracles can be implemented using existing parameterized algorithms which have a running time of the form $c^\ell \cdot n^{O(1)}$. We therefore associate a running time of c^ℓ with the query (S, ℓ) . The formal definition of extension oracles is as follows.

► **Definition 4** (Extension Oracle). *Let $(U, \mathbf{w}, \mathcal{F})$ be a WEIGHTEDSM instance and let $\alpha \geq 1$. An α -extension oracle for $(U, \mathbf{w}, \mathcal{F})$ is a function $\text{Ext} : 2^U \times \mathbb{N}_{\geq 0} \rightarrow 2^U$ such that for every $S \subseteq U$ and $\ell \in \mathbb{N}_{\geq 0}$ the following holds:*

1. $\text{Ext}(S, \ell) \cup S \in \mathcal{F}$.
2. $\mathbf{w}(\text{Ext}(S, \ell)) \leq \alpha \cdot \min\{\mathbf{w}(X) \mid X \subseteq U, |X| \leq \ell, X \cup S \in \mathcal{F}\}$ (we set $\min \emptyset = \infty$).

In the (α, c) -extension model the input for the algorithm is a finite set U and a weight function $\mathbf{w} : U \rightarrow \mathbb{N}$. Furthermore, the algorithm is given oracle access to an α -extension oracle Ext of $(U, \mathbf{w}, \mathcal{F})$ for some monotone set system \mathcal{F} of U . For every $\beta \geq 1$, we say an algorithm is a β -approximation algorithm for WEIGHTEDSM (UNWEIGHTEDSM) in the (α, c) -extension model if for every WEIGHTEDSM (UNWEIGHTEDSM) instance $(U, \mathbf{w}, \mathcal{F})$ and α -extension oracle Ext of the instance, the algorithm returns $T \in \mathcal{F}$ such that $\mathbf{w}(T) \leq \beta \cdot \text{opt}(U, \mathbf{w}, \mathcal{F})$. The running time of an algorithm in this model is the number of computations steps plus c^ℓ for every query (S, ℓ) issued to the oracle during the execution. Following the standard worst-case analysis convention, we say an algorithm runs in time $f(n)$ if for every WEIGHTEDSM instance $(U, \mathbf{w}, \mathcal{F})$ and α -extension oracle Ext of the instance the algorithm runs in time at most $f(|U|)$.

The (α, c) -extension model is studied in [12] for the special case of UNWEIGHTEDSM. The authors of [12] provide a deterministic β -approximation algorithm in the (α, c) -extension model, known as *deterministic approximate monotone local search*, which runs in time $(\text{amls}(\alpha, c, \beta))^{n+o(n)}$, where $\text{amls}(\alpha, c, \beta)$ is defined as the optimal value of a continuous optimization problem. Throughout the paper we use amls to denote this function. We provide the formal definition of amls in the full version of the paper for completeness. We note that this formal definition is not required for the understanding of the results in this paper. It is also shown in [12] that the value of $\text{amls}(\alpha, c, \beta)$ can be computed up to precision of ε in time polynomial in the encoding length of α, c, β and ε .

This algorithmic result is complemented in [12] with a matching lower bound.

► **Lemma 5** ([12]). *For every $\alpha, \beta, c \geq 1$ and $\varepsilon > 0$ there is no deterministic β -approximation for UNWEIGHTEDSM in the (α, c) -extension model which runs in time $(\text{amls}(\alpha, c, \beta) - \varepsilon)^n \cdot n^{O(1)}$.*

Furthermore, it is shown that the running time of the deterministic approximate monotone local search is better than brute force for every $\beta > 1$.

► **Lemma 6** ([12]). *For all $\alpha, c \geq 1$ and $\beta > 1$ it holds that $\mathbf{amls}(\alpha, c, \beta) < \mathbf{brute}(\beta)$.*

The results of [12] also include a randomized algorithm which omits the subexponential factor in the running time and the lower bound also holds for randomized algorithms. In [12] the authors use the deterministic approximate monotone local search algorithm to obtain exponential-time approximation algorithms for multiple *unweighted* problems such as VERTEX COVER and FEEDBACK VERTEX SET. We generalize the algorithmic results of [12] to the weighted setting and similarly use it to obtain exponential-time approximation algorithms for *weighted* variants of the mentioned problems (see Section 3).

Since UNWEIGHTEDSM is a special case of WEIGHTEDSM, Lemma 5 immediately implies the following

► **Corollary 7.** *For every $\alpha, c, \beta \geq 1$ and $\varepsilon > 0$ there is no deterministic β -approximation for WEIGHTEDSM in the (α, c) -extension model which runs in time $(\mathbf{amls}(\alpha, c, \beta) - \varepsilon)^n \cdot n^{\mathcal{O}(1)}$.*

Our second result is an algorithm which matches the running time in Corollary 7 up to an additive term of ε in the base of the running time.

► **Theorem 8** (Weighted Approximate Monotone Local Search). *For every $\alpha, c \geq 1$, $\beta > 1$ and $\varepsilon > 0$ there is a deterministic β -approximation for WEIGHTEDSM in the (α, c) -extension model which runs in time $\mathcal{O}((\mathbf{amls}(\alpha, c, \beta) + \varepsilon)^n)$.*

The proof of Theorem 8 is similar to the one of Theorem 3. It applies rounding to the weights on the elements, and then utilizes a construction from [12] for each of the weight classes.

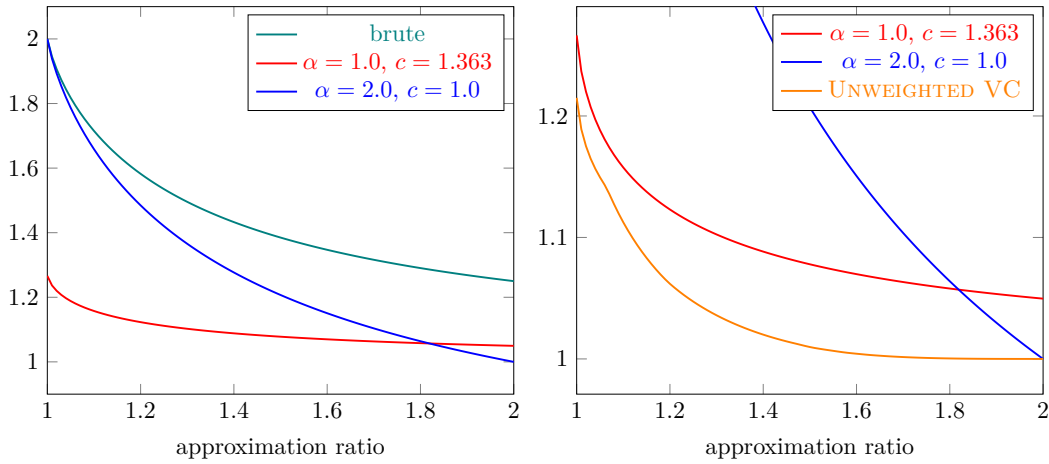
The result of Theorem 8 only applies for $\beta > 1$. If $\alpha = \beta = 1$ the result of [15] can be used to obtain an exact algorithm with running time $n^{\mathcal{O}(1)} \cdot (2 - \frac{1}{c})^n$. For $\alpha > \beta = 1$ Corollary 7 implies that the best possible running time is $\mathcal{O}^*(2^n)$ which can be attained by brute force.

3 Applications

Our results provide exponential-time approximation algorithms for a variety of weighted vertex-deletions problems. For illustration purposes, let us first focus on the WEIGHTED VERTEX COVER problem where we are given a graph G with vertex weights $\mathbf{w}: V(G) \rightarrow \mathbb{N}$, and we ask for a vertex cover $S \subseteq V(G)$ of minimum weight $\mathbf{w}(S) = \sum_{v \in S} \mathbf{w}(v)$. It is well-known that WEIGHTED VERTEX COVER admits a polynomial-time 2-approximation algorithm [5]. Also, the problem can be solved exactly in time $\mathcal{O}^*(1.238^n)$ [22].

For the unweighted version UNWEIGHTED VERTEX COVER, Bourgeois, Escoffier and Paschos [6] designed several exponential-time approximation algorithms for approximation ratios in the range $(1, 2)$. For example, they obtain a 1.1-approximation algorithm running in time $\mathcal{O}^*(1.127^n)$ where n denotes the number of vertices of the input graph. These running times are further improved in [11, 12] using the framework of Approximate Monotone Local Search. Indeed, the fastest known 1.1-approximation algorithm for UNWEIGHTED VERTEX COVER runs in time $\mathcal{O}^*(1.113^n)$ [12].

For the weighted version, no such results have been obtained so far. We use Theorem 8 to design the first exponential β -approximation algorithms for WEIGHTED VERTEX COVER for all $\beta \in (1, 2)$. For the extension oracle, we can rely on the well-known polynomial-time 2-approximation algorithm. Given a set $S \subseteq V(G)$, we delete all vertices in S and apply the



■ **Figure 1** The left figure shows running times for WEIGHTED VERTEX COVER and the right side provides a comparison to UNWEIGHTED VERTEX COVER. A dot at (β, d) means that the respective algorithm outputs an β -approximation in time $\mathcal{O}^*(d^n)$.

■ **Table 1** The table shows the running times for WEIGHTED VERTEX COVER (second and third row) and UNWEIGHTED VERTEX COVER (last row). An entry d in column β means that the respective algorithm outputs a β -approximation in time $\mathcal{O}^*(d^n)$.

	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9
brute	1.716	1.583	1.496	1.433	1.385	1.347	1.317	1.291	1.269
$(\alpha = 1, c = 1.363)$	1.158	1.123	1.103	1.089	1.078	1.07	1.064	1.058	1.054
$(\alpha = 2, c = 1)$	1.659	1.485	1.366	1.277	1.208	1.151	1.104	1.064	1.03
UNWEIGHTED VC	1.113	1.062	1.036	1.02	1.01	1.005	1.002	1.0004	1.00005

2-approximation algorithm to the graph $G - S$ which outputs a vertex cover X such that $\mathbf{w}(X) \leq 2 \cdot \mathbf{w}(\text{OPT})$ where OPT denotes a minimum vertex cover of $G - S$. As a result, we can implement a 2-extension oracle in polynomial time which corresponds (up to polynomial factors) to cost $c = 1$. So Theorem 8 results in a β -approximation algorithm for WEIGHTED VERTEX COVER which runs in time $\mathcal{O}^*((\text{amlS}(\alpha, c, \beta) + \varepsilon)^n)$ for every $\beta > 1$, where $\alpha = 2$ and $c = 1$. A visualization is given in Figure 1.

Instead of using a polynomial-time 2-approximation algorithm, we can also rely on existing FPT algorithms for WEIGHTED VERTEX COVER to simulate the extension oracle. Let us point out that different parameterizations have been considered for WEIGHTED VERTEX COVER in the literature. For example, Niedermeier and Rossmanith [20] give FPT algorithms for WEIGHTED VERTEX COVER parameterized by the weight of the optimal solution. However, in light of the computational model introduced above, we require FPT algorithms parameterized by the *size* of the solution. Given a graph G with vertex weights $\mathbf{w}: V(G) \rightarrow \mathbb{N}$ and an integer $k \geq 0$, we ask for a vertex cover of weight at most W where W is the minimum weight of a vertex cover of size at most k . The best known FPT algorithm (parameterized by k) for this problem runs in time $1.363^k \cdot n^{\mathcal{O}(1)}$ [21]. This algorithm provides an extension algorithm with parameters $\alpha = 1$ and $c = 1.363$. Using Theorem 8, we obtain a β -approximation algorithm for WEIGHTED VERTEX COVER running in time $\mathcal{O}^*((\text{amlS}(\alpha, c, \beta) + \varepsilon)^n)$. For $\beta = 1.1$, we obtain a running time of $\mathcal{O}^*(1.158^n)$.

■ **Table 2** List of weighted deletion problems admitting a single-exponential parameterized algorithm running in time $O^*(c_1^k)$ and/or a polynomial-time α_2 -approximation algorithm. The problems TOURNAMENT FVS, CLUSTER GRAPH VERTEX DELETION, COGRAPH VERTEX DELETION and SPLIT VERTEX DELETION can be easily reduced to d -HITTING SET for appropriate values of d by exploiting known characterizations in terms of forbidden induced subgraphs. Additionally, for TOURNAMENT FVS we can rely on iterative compression to obtain a $O^*(2^k)$ algorithm (see, e.g., [8]).

Problem	c_1	det.	α_2	det.
VERTEX COVER	1.363 [21]	✓	2 [5]	✓
FVS	3.618 [1]	✓	2 [3]	✓
TOURNAMENT FVS	2.0	✓	3	✓
SUBSET FVS	-	✓	8 [13]	✓
3-HITTING SET	2.168 [21]	✓	3 [5]	✓
d -HITTING SET ($d \geq 4$)	$d - 0.832$ [14]	✓	d [5]	✓
CLUSTER GRAPH VERTEX DELETION	2.168	✓	3	✓
COGRAPH VERTEX DELETION	3.168	✓	4	✓
SPLIT VERTEX DELETION	4.168	✓	5	✓
PARTIAL VERTEX COVER	-	✓	2 [7]	✓

We provide running times for selected approximation ratios for both algorithms in Table 1 and a graphical comparison in Figure 1. We also compare the running times to the approximate brute-force search and the best algorithms in the unweighted setting [12]. It can be observed that the second algorithm (using the FPT algorithm as an extension subroutine) is faster for $\beta \lesssim 1.82$. Also, there is still a noticeable gap to the unweighted setting. This can be mainly explained by the fact that the approximation algorithm for the unweighted setting [12] relies on parameterized approximation algorithms for UNWEIGHTED VERTEX COVER [17] which are currently unavailable in the weighted setting.

We stress that our results are not limited to WEIGHTED VERTEX COVER, but they are applicable to various vertex-deletion problems including WEIGHTED FEEDBACK VERTEX SET and WEIGHTED d -HITTING SET (see Figure 2). Table 2 gives an overview on problems for which we obtain exponential approximation algorithms by simulating the extension oracle by an FPT algorithm (parameterized by solution size) running in time $c_1^k \cdot n^{O(1)}$ or a polynomial-time α_2 -approximation algorithm. For all the problems listed in Table 2, we obtain the first exponential β -approximation algorithms for all $\beta \in (1, \alpha_2)$. Observe that these algorithms always outperform the approximate brute-force search by Lemma 6. We provide data on the running times of these algorithms in the full version of the paper.

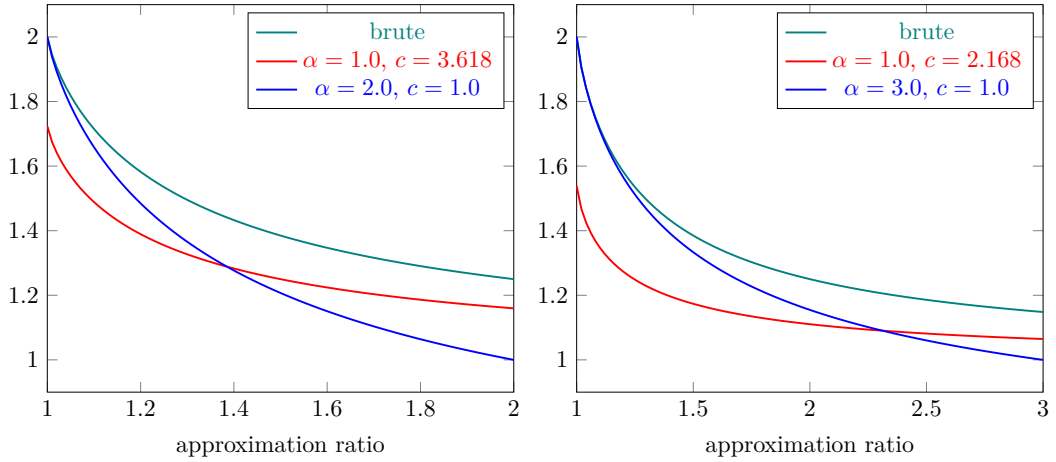
Finally, let us point out that our methods are also applicable if there is a parameterized approximation algorithm for a weighted vertex deletion problem, i.e., an α -approximation algorithm running time $c^k \cdot n^{O(1)}$. In [18], such algorithms have been obtained for WEIGHTED DIRECTED FEEDBACK VERTEX SET, WEIGHTED DIRECTED SUBSET FVS, WEIGHTED DIRECTED ODD CYCLE TRANSVERSAL and WEIGHTED MULTICUT. As a result, we also obtain exponential β -approximation algorithms for these problems that outperform the approximate brute-force search.

4 Weighted Approximate Brute Force

In this section we prove Theorem 3. The algorithm is based on the notion of *covering families*.

► **Definition 9** (Covering Family). *Let U be a finite set, $\mathbf{w}: U \rightarrow \mathbb{N}$ be a weight function and $\alpha \geq 1$. We say $\mathcal{C} \subseteq 2^U$ is an α -covering family of U and \mathbf{w} if for every $S \subseteq U$ there exists $T \in \mathcal{C}$ such that $S \subseteq T$ and $\mathbf{w}(T) \leq \alpha \cdot \mathbf{w}(S)$.*

17:10 Approximate Monotone Local Search for Weighted Problems



(a) FEEDBACK VERTEX SET.

(b) 3-HITTING SET.

■ **Figure 2** The figure shows running times for FEEDBACK VERTEX SET and 3-HITTING SET. A dot at (β, d) means that the respective algorithm outputs an β -approximation in time $O^*(d^n)$.

An α -covering family \mathcal{C} can be easily used to attain an α -approximation algorithm in the membership model as follows. The algorithm constructs the covering family \mathcal{C} and uses the membership oracle to compute $\mathcal{C} \cap \mathcal{F}$ using $|\mathcal{C}|$ queries. The algorithm then returns the set $Q \in \mathcal{C} \cap \mathcal{F}$ of minimum weight. To show correctness, consider a set $S \in \mathcal{F}$ such that $\mathbf{w}(S) = \text{opt}(U, \mathbf{w}, \mathcal{F})$. Since \mathcal{C} is an α -covering family there is $T \in \mathcal{C}$ such that $S \subseteq T$ and $\mathbf{w}(T) \leq \alpha \cdot \mathbf{w}(S) = \alpha \cdot \text{opt}(U, \mathbf{w}, \mathcal{F})$. Since \mathcal{F} is monotone it holds that $T \in \mathcal{F}$, thus $T \in \mathcal{C} \cap \mathcal{F}$, and we conclude that the algorithm returns a set of weight at most $\alpha \cdot \text{opt}(U, \mathbf{w}, \mathcal{F})$. The running time of the algorithm, up to polynomial factors, is the construction time of \mathcal{C} plus $|\mathcal{C}|$.

By the above argument, the proof of Theorem 3 boils down to the construction of α -covering families. To this end, we show the next result. Recall that n denotes the size of the universe U .

► **Lemma 10.** *There exists an algorithm which given a finite set U , a weight function $\mathbf{w} : U \rightarrow \mathbb{N}$ and $\alpha > 1$, constructs an α -covering family \mathcal{C} of U and \mathbf{w} such that $|\mathcal{C}| \leq (\text{brute}(\alpha))^{n+o(n)}$. Furthermore, the running time of the algorithm is $(\text{brute}(\alpha))^{n+o(n)}$.*

Using the argument above, Theorem 3 is an immediate consequence of Lemma 10. The construction of the covering family in Lemma 10 is based on a rounding of the weight function and a reduction to covering families in the unweighted case. The construction of such families is implicitly given in [11].

► **Lemma 11** ([11]). *There exists an algorithm which given a finite set U and $\alpha > 1$ returns an α -covering family \mathcal{C} of U and the uniform weight function $\mathbf{w} : U \rightarrow \{1\}$ such that $|\mathcal{C}| \leq (\text{brute}(\alpha))^n \cdot n^{O(1)}$. Furthermore, the algorithm runs in time $(\text{brute}(\alpha))^n \cdot n^{O(1)}$.*

The proof of Lemma 10 is based on the following seemingly weaker version of Lemma 10.

► **Lemma 12.** *There exists an algorithm \mathcal{A} which given a finite set U , a weight function $\mathbf{w} : U \rightarrow \mathbb{N}$, $\beta > 1$ and $0 < \delta < 1$, constructs a $(1 + \delta) \cdot \beta$ -covering family \mathcal{C} of U and \mathbf{w} such that $|\mathcal{C}| \leq (\text{brute}(\beta))^n \cdot n^{O(\frac{1}{\delta} \log(\frac{\beta}{\delta}))}$. Furthermore, the running time of the algorithm is $(\text{brute}(\beta))^n \cdot n^{O(\frac{1}{\delta} \log(\frac{\beta}{\delta}))}$.*

Proof. The algorithm \mathcal{A} works as follows:

- Define $\gamma := 1 + \frac{\delta}{2} > 1$. For $i \geq 0$ let

$$U_i := \{u \in U \mid \gamma^i \leq \mathbf{w}(u) < \gamma^{i+1}\} \quad (2)$$

and $n_i := |U_i|$. Let $I := \{i \in \mathbb{Z}_{\geq 0} \mid U_i \neq \emptyset\}$ denote the set of indices $i \geq 0$ for which U_i is non-empty. Note that $|I| \leq n$.

- For each $i \in I$ construct a β -covering family \mathcal{C}_i of the universe U_i and the uniform weight function using Lemma 11.
- Define $d := \lceil (2/\delta) \cdot \log(2n/\delta) \rceil$ and for each $k \in I$, let $I_k := \{i \in I \mid k-d \leq i \leq k\}$ denote the indices in I between $k-d$ and k .
- For every $k \in I$, let $r_k := |I_k|$ and define

$$W_k := \bigcup_{i \in I: 1 \leq i < k-d} U_i \quad \text{and}$$

$$\mathcal{Q}_k := \left\{ W_k \cup E_1 \cup \dots \cup E_{r_k} \mid (E_1, \dots, E_{r_k}) \in \prod_{i \in I_k} \mathcal{C}_i \right\}.$$

- Return the set $\mathcal{C} := \bigcup_{k \in I} \mathcal{Q}_k$.

▷ **Claim 13.** The algorithm \mathcal{A} returns a $(1 + \delta) \cdot \beta$ -covering family of U and \mathbf{w} .

Proof. Let us pick a set $S \subseteq U$. We show that there exists $T \in \mathcal{C}$ such that $S \subseteq T$ and $\mathbf{w}(T) \leq (1 + \delta) \cdot \beta \cdot \mathbf{w}(S)$.

By the definition of γ and d , it holds that

$$\gamma^d \geq \left(1 + \frac{\delta}{2}\right)^{(2/\delta) \cdot \log(2n/\delta)} \geq 2^{\log(2n/\delta)} = \frac{2n}{\delta} \quad (3)$$

using that $(1 + \frac{1}{x})^x \geq 2$ for all $x \geq 1$. Let $k \in I$ be the largest index such that $S \cap U_k \neq \emptyset$. It holds that

$$\mathbf{w}(W_k) < n \cdot \gamma^{k-d} \leq \frac{\delta}{2} \cdot \gamma^k \leq \frac{\delta}{2} \cdot \mathbf{w}(S) \quad (4)$$

where the first inequality follows from the fact that $|W_k| \leq n$ and each $u \in W_k$ belongs to a set U_i where $i \leq k-d-1$ and therefore $\mathbf{w}(u) < \gamma^{k-d-1+1} = \gamma^{k-d}$ by (2). The second inequality follows from (3) and finally the last inequality holds because by definition of k , there exists $u \in S \cap U_k$ such that $\mathbf{w}(u) \geq \gamma^k$ by (2).

For every $i \in I_k$ define $S_i := S \cap U_i$. Since \mathcal{C}_i is a β -covering family of U_i and the *uniform weight function*, for each $i \in I_k$ there exists $T_i \in \mathcal{C}_i$ such that $S_i \subseteq T_i$ and $|T_i| \leq \beta \cdot |S_i|$. Hence, for all $i \in I_k$ it holds that

$$\mathbf{w}(T_i) \leq \gamma^{i+1} \cdot |T_i| \leq \gamma^{i+1} \cdot \beta \cdot |S_i| \leq \gamma \cdot \beta \cdot \mathbf{w}(S_i) \quad (5)$$

where the first inequality follows from the fact that $T_i \subseteq U_i$ and (2), the second inequality follows from the definition of T_i and finally the last one again follows from the fact that $S_i \subseteq U_i$ and (2).

Let $\bar{T} := \bigcup_{i \in I_k} T_i$ and define

$$T := \left(W_k \cup \bar{T} \right) \in \mathcal{Q}_k \subseteq \mathcal{C}.$$

17:12 Approximate Monotone Local Search for Weighted Problems

Then we have

$$\begin{aligned}
 S &= S \cap U = S \cap \left(\bigcup_{i \in I} U_i \right) = \left(S \cap \left(\bigcup_{i \in I: i < k-d} U_i \right) \right) \cup \left(S \cap \left(\bigcup_{i \in I_k} U_i \right) \right) \\
 &= (S \cap W_k) \cup \left(\bigcup_{i \in I_k} (S \cap U_i) \right) \\
 &\subseteq W_k \cup \left(\bigcup_{i \in I_k} T_i \right) \\
 &= T.
 \end{aligned}$$

Finally, it also holds that

$$\begin{aligned}
 \mathbf{w}(T) &= \mathbf{w}(W_k) + \sum_{i \in I_k} \mathbf{w}(T_i) \\
 &\leq \frac{\delta}{2} \cdot \mathbf{w}(S) + \sum_{i \in I_k} \gamma \cdot \beta \cdot \mathbf{w}(S_i) && \text{by (4) and (5)} \\
 &\leq \frac{\delta}{2} \cdot \beta \cdot \mathbf{w}(S) + \gamma \cdot \beta \cdot \mathbf{w}(S) \\
 &\leq (1 + \delta) \cdot \beta \cdot \mathbf{w}(S).
 \end{aligned}$$

This shows that \mathcal{C} is a $(1 + \delta) \cdot \beta$ -covering family of U and \mathbf{w} . ◁

▷ Claim 14.

$$|\mathcal{C}| \leq (\mathbf{brute}(\beta))^n \cdot n^{\mathcal{O}(\frac{1}{\delta} \cdot \log(\frac{n}{\delta}))}.$$

Proof. By Lemma 11, for each $i \in I$ it holds that

$$|\mathcal{C}_i| \leq (\mathbf{brute}(\beta))^{n_i} \cdot n_i^c \tag{6}$$

for some $c > 0$. Thus, for every $k \in I$,

$$\begin{aligned}
 |\mathcal{Q}_k| &\leq \left| \prod_{i \in I_k} \mathcal{C}_i \right| = \prod_{i \in I_k} |\mathcal{C}_i| \leq \prod_{i \in I_k} (\mathbf{brute}(\beta))^{n_i} \cdot n_i^c \\
 &= (\mathbf{brute}(\beta))^{\sum_{i \in I_k} n_i} \cdot \prod_{i \in I_k} n_i^{c \cdot |I_k|} \\
 &\leq (\mathbf{brute}(\beta))^n \cdot n^{c \cdot (d+1)} \\
 &= (\mathbf{brute}(\beta))^n \cdot n^{\mathcal{O}(\frac{1}{\delta} \cdot \log(\frac{n}{\delta}))}.
 \end{aligned}$$

Finally, we have that

$$|\mathcal{C}| = \left| \bigcup_{k \in I} \mathcal{Q}_k \right| \leq \sum_{k \in I} |\mathcal{Q}_k| = (\mathbf{brute}(\beta))^n \cdot n^{\mathcal{O}(\frac{1}{\delta} \cdot \log(\frac{n}{\delta}))}$$

since $|I| \leq n$. ◁

▷ Claim 15. The running time of \mathcal{A} is $(\mathbf{brute}(\beta))^n \cdot n^{\mathcal{O}(\frac{1}{\delta} \cdot \log(\frac{n}{\delta}))}$.

Proof. The construction of $\{\mathcal{C}_i\}_{i \in I}$ takes

$$\sum_{i \in I} (\mathbf{brute}(\beta))^{n_i} \cdot n_i^{\mathcal{O}(1)} \leq (\mathbf{brute}(\beta))^n \cdot n^{\mathcal{O}(1)} \quad (7)$$

which follows from Lemma 11. Finally, the construction of \mathcal{C} takes time proportional to the size of \mathcal{C} where we have

$$|\mathcal{C}| = (\mathbf{brute}(\beta))^n \cdot n^{\mathcal{O}(\frac{1}{3} \cdot \log(\frac{n}{\beta}))}$$

by Claim 14. All in all, the running time of \mathcal{A} is upper bounded by $(\mathbf{brute}(\beta))^n \cdot n^{\mathcal{O}(\frac{1}{3} \cdot \log(\frac{n}{\beta}))}$. \triangleleft

The lemma follows from Claims 13–15. \blacktriangleleft

To prove Lemma 10, we combine Lemma 12 with the following technical lemma.

► **Lemma 16.** *Let $f: I \rightarrow \mathbb{R}$ be a continuous function on an open interval $I \subseteq \mathbb{R}$ and let $\alpha > 1$ such that $\alpha \in I$. Define $\beta(n) := \alpha - \frac{1}{\log(n)}$ and $\delta(n) := \frac{\alpha}{\beta(n)} - 1$ for all $n \in \mathbb{N}$. Then it holds that*

$$f(\beta(n))^n \cdot n^{\mathcal{O}(\frac{1}{\delta(n)} \cdot \log(\frac{n}{\delta(n)}))} = f(\alpha)^{n+o(n)}.$$

The proof of Lemma 16 is given in the full version of the paper.

Proof of Lemma 10. We claim that the algorithm \mathcal{A} from Lemma 12 with $\beta := \alpha - \frac{1}{\log(n)}$ and $\delta := \frac{\alpha}{\beta} - 1$ satisfies the properties listed in Lemma 10. Note that β and δ are functions of n , but we write β and δ instead of $\beta(n)$ and $\delta(n)$ for the sake of readability.

Observe that we have $(1 + \delta) \cdot \beta = \frac{\alpha}{\beta} \cdot \beta = \alpha$. Hence, by Lemma 12, the set returned by \mathcal{A} is an α -covering family \mathcal{C} of U and \mathbf{w} such that $|\mathcal{C}| \leq (\mathbf{brute}(\beta))^n \cdot n^{\mathcal{O}(\frac{1}{3} \cdot \log(\frac{n}{\beta}))}$. The running time of the algorithm is also bounded by $(\mathbf{brute}(\beta))^n \cdot n^{\mathcal{O}(\frac{1}{3} \cdot \log(\frac{n}{\beta}))}$.

By (1), $\mathbf{brute}(x)$ is a continuous function of x because the entropy function is continuous. Therefore by Lemma 16 it holds that $(\mathbf{brute}(\beta))^n \cdot n^{\mathcal{O}(\frac{1}{3} \cdot \log(\frac{n}{\beta}))} = (\mathbf{brute}(\alpha))^{n+o(n)}$ which proves the lemma. \blacktriangleleft

5 Weighted Monotone Local Search

In this section we prove Theorem 8. The proof presented here follows the outline of the proof of Theorem 3 in Section 4, using the concept of an (α, β) -extension family, an adaptation of the term α -covering family presented in Section 4 to the setting of extension oracles. While the items in a covering family are queries to a membership oracle, and hence are subsets of U , the items in an extension family represent queries to the extension oracle, and thus are pairs (T, ℓ) of a subset T of U and a non-negative integer ℓ . A reduction to a construction from [12] is used to build the extension family, and the extension family itself can be trivially used to attain a β -approximation algorithm for WEIGHTEDSM.

► **Definition 17 (Extension Family).** *Let U be a finite set and $\mathbf{w}: U \rightarrow \mathbb{N}$ be a weight function. Furthermore, let $\alpha, \beta \geq 1$. We say $\mathcal{E} \subseteq 2^U \times \mathbb{N}$ is an (α, β) -extension family of U and \mathbf{w} if for every $S \subseteq U$ there exists $(T, \ell) \in \mathcal{E}$ which satisfies the following:*

17:14 Approximate Monotone Local Search for Weighted Problems

$$\begin{aligned} |S \setminus T| &\leq \ell, \\ \mathbf{w}(T) + \alpha \cdot \mathbf{w}(S \setminus T) &\leq \beta \cdot \mathbf{w}(S). \end{aligned} \tag{8}$$

Let $c \geq 1$. The c -cost of an (α, β) -extension family \mathcal{E} of U and \mathbf{w} is defined as $\text{cost}_c(\mathcal{E}) := \sum_{(T, \ell) \in \mathcal{E}} c^\ell$. The proof of Theorem 8 relies on the following lemma.

► **Lemma 18.** *Let $\alpha, c \geq 1$, $\beta > 1$ and $\varepsilon > 0$. Then there is an algorithm which given a finite set U and a weight function $\mathbf{w}: U \rightarrow \mathbb{N}$ returns an (α, β) -extension family \mathcal{E} of U and \mathbf{w} such that $\text{cost}_c(\mathcal{E}) = \mathcal{O}\left(\left(\text{amls}(\alpha, c, \beta) + \varepsilon\right)^n\right)$. Furthermore, the running time of the algorithm is $\mathcal{O}\left(\left(\text{amls}(\alpha, c, \beta) + \varepsilon\right)^n\right)$.*

Before heading to the proof of Lemma 18, we show how the lemma can be used to prove Theorem 8.

Proof of Theorem 8. Let $\alpha, c \geq 1$, $\beta > 1$ and $\varepsilon > 0$. Consider the following algorithm \mathcal{A} :

- Given the input $(U, \mathbf{w}, \mathcal{F})$, use Lemma 18 with α, β, c and ε to construct an (α, β) -extension family \mathcal{E} of U and \mathbf{w} .
- Let Ext be the α -extension oracle. For each $(T_i, \ell_i) \in \mathcal{E}$, use Ext to compute $X_i := \text{Ext}(T_i, \ell_i)$.
- Define $\mathcal{T} := \{T_i \cup X_i \mid (T_i, \ell_i) \in \mathcal{E}\}$ and return a set in \mathcal{T} with the minimum weight, i.e., a set $T \in \mathcal{T}$ such that $\mathbf{w}(T) = \min\{\mathbf{w}(Y) \mid Y \in \mathcal{T}\}$.

The algorithm \mathcal{A} first creates an (α, β) -extension family \mathcal{E} . Then it goes over all elements $(T_i, \ell_i) \in \mathcal{E}$ and queries the oracle with (T_i, ℓ_i) . So the running time of this algorithm in the (α, c) -extension model is equal to the running time of the algorithm from Lemma 18 plus c^{ℓ_i} for every query (T_i, ℓ_i) , i.e., the cost of the extension family \mathcal{E} . By Lemma 18 this value is at most

$$\begin{aligned} (\text{amls}(\alpha, c, \beta) + \varepsilon)^n + \text{cost}_c(\mathcal{E}) &= \mathcal{O}\left(\left(\text{amls}(\alpha, c, \beta) + \varepsilon\right)^n\right) + \mathcal{O}\left(\left(\text{amls}(\alpha, c, \beta) + \varepsilon\right)^n\right) \\ &= \mathcal{O}\left(\left(\text{amls}(\alpha, c, \beta) + \varepsilon\right)^n\right) \end{aligned}$$

▷ **Claim 19.** The algorithm \mathcal{A} is a deterministic β -approximation for WEIGHTEDSM in the (α, c) -extension model.

Proof. Let $R \in \mathcal{T}$ be the set returned by the algorithm. Note that by definition of \mathcal{T} , $R = T_j \cup X_j$ for some $(T_j, \ell_j) \in \mathcal{E}$ and $X_j = \text{Ext}(T_j, \ell_j)$. In particular, $R = T_j \cup X_j = T_j \cup \text{Ext}(T_j, \ell_j) \in \mathcal{F}$ (see Definition 4).

Let $S \in \mathcal{F}$ be a set with minimum weight in \mathcal{F} , i.e., $\mathbf{w}(S) = \min\{\mathbf{w}(Y) \mid Y \in \mathcal{F}\} = \text{opt}(U, \mathbf{w}, \mathcal{F})$. Since \mathcal{E} is an (α, β) -extension family, there exists $(T_i, \ell_i) \in \mathcal{E}$ such that S, T_i and ℓ_i satisfy (8) for $T = T_i$ and $\ell = \ell_i$. Moreover, observe that

$$S \setminus T_i \in \{X \subseteq U \mid |X| \leq \ell_i, X \cup T_i \in \mathcal{F}\}$$

because $S \subseteq (S \setminus T_i) \cup T_i \in \mathcal{F}$. Hence, by the definition of Ext and (T_i, ℓ_i) , it follows that

$$\begin{aligned} \mathbf{w}(X_i) = \mathbf{w}(\text{Ext}(T_i, \ell_i)) &\leq \alpha \cdot \min\{\mathbf{w}(X) \mid X \subseteq U, |X| \leq \ell_i, X \cup T_i \in \mathcal{F}\} \\ &\leq \alpha \cdot \mathbf{w}(S \setminus T_i) \\ &\leq \beta \cdot \mathbf{w}(S) - \mathbf{w}(T_i). \end{aligned}$$

Finally, we have

$$\begin{aligned}
\mathbf{w}(R) &= \min\{\mathbf{w}(Y) \mid Y \in \mathcal{T}\} \\
&\leq \mathbf{w}(T_i \cup X_i) \\
&\leq \mathbf{w}(T_i) + \mathbf{w}(X_i) \\
&\leq \beta \cdot \mathbf{w}(S) \\
&= \beta \cdot \text{opt}(U, \mathbf{w}, \mathcal{F})
\end{aligned}$$

which proves the claim. \triangleleft

This shows that the algorithm \mathcal{A} is a deterministic β -approximation for WEIGHTEDSM in the (α, c) -extension model with the running time given in Theorem 8. \blacktriangleleft

The proof of Lemma 18 relies on the following construction for extension families in the unweighted case which is implicitly given in [12].

► **Lemma 20** ([12]). *Let $\alpha, c \geq 1$ and $\beta > 1$. Then there is a deterministic algorithm which given a finite set U , returns an (α, β) -extension family \mathcal{E} of U and the uniform weight function $\mathbf{w}: U \rightarrow \{1\}$ such that $\text{cost}_c(\mathcal{E}) \leq \left(\text{amls}(\alpha, c, \beta)\right)^{n+o(n)}$. Furthermore, the running time of the algorithm is $\left(\text{amls}(\alpha, c, \beta)\right)^{n+o(n)}$.*

In a manner analogous to Section 4, we begin by introducing Lemma 21, which presents a slightly weaker form of Lemma 18. Within Lemma 21, ζ takes the place of the previously mentioned β from Lemma 18. Subsequently, in the proof of Lemma 18, we set the value of ζ as a function of β .

► **Lemma 21.** *Let $\alpha, c \geq 1$, $\zeta > 1$ and $0 < \delta < 1$. Then there is an algorithm which given a finite set U and a weight function $\mathbf{w}: U \rightarrow \mathbb{N}$ returns an $(\alpha, (1 + \delta) \cdot \zeta)$ -extension family \mathcal{E} of U and \mathbf{w} such that $\text{cost}_c(\mathcal{E}) \leq \left(\text{amls}(\alpha, c, \zeta)\right)^{n+o(n)}$. Furthermore, the running time of the algorithm is $\left(\text{amls}(\alpha, c, \zeta)\right)^{n+o(n)}$.*

As already indicated above, the proof of Lemma 21 is similar to the proof of Lemma 12. The algorithm used to compute the desired extension family is given in Algorithm 1. The full proof of Lemma 21 can be found in Appendix A. Lastly, we use the following property of the function `amls`.

► **Lemma 22.** *For every fixed $\alpha > 1$ and $c > 1$, $\text{amls}(\alpha, c, x)$ is a continuous function of x on the interval $(1, \infty)$.*

The proof of Lemma 22 is given in the full version of the paper.

Proof of Lemma 18. Let $\alpha, c \geq 1$, $\beta > 1$ and $\varepsilon > 0$. Since `amls` (α, c, ζ) is continuous in ζ by Lemma 22, there exists a $\zeta' \in (1, \beta)$ such that $\text{amls}(\alpha, c, \zeta') < \text{amls}(\alpha, c, \beta) + \frac{\varepsilon}{2}$. To prove the lemma, we use the algorithm from Lemma 21 (i.e., Algorithm 1) with $\zeta := \zeta'$ and $\delta := \beta/\zeta - 1$.

Note that we have $(1 + \delta) \cdot \zeta = (\beta/\zeta) \cdot \zeta = \beta$. Hence, by Lemma 21, the set \mathcal{E} returned by Algorithm 1 is an (α, β) -extension family of U and \mathbf{w} such that

$$\text{cost}_c(\mathcal{E}) \leq \left(\text{amls}(\alpha, c, \zeta')\right)^{n+o(n)} \leq \left(\text{amls}(\alpha, c, \beta) + \frac{\varepsilon}{2}\right)^{n+o(n)} = \mathcal{O}\left(\left(\text{amls}(\alpha, c, \beta) + \varepsilon\right)^n\right)$$

17:16 Approximate Monotone Local Search for Weighted Problems

■ **Algorithm 1** Extension Family for Arbitrary Weight Functions.

Configuration: $\alpha \geq 1$, $c \geq 1$, $\zeta > 1$ and $0 < \delta < 1$

Input: A universe U , weight function $\mathbf{w}: U \rightarrow \mathbb{N}$

1: Define $\gamma := 1 + \frac{\delta}{2} > 1$. For $i \geq 0$ let

$$U_i := \{u \in U \mid \gamma^i \leq \mathbf{w}(u) < \gamma^{i+1}\} \quad (9)$$

and $n_i := |U_i|$. Let $I := \{i \in \mathbb{Z}_{\geq 0} \mid U_i \neq \emptyset\}$ denote the set of indices $i \geq 0$ for which U_i is non-empty. Note that $|I| \leq n$.

2: For each $i \in I$ construct an (α, ζ) -extension family \mathcal{E}_i of the universe U_i and the uniform weight function using Lemma 20 with respect to α , c and ζ .

3: Define $d := \lceil (2/\delta) \cdot \log(2n/\delta) \rceil$ and for each $k \in I$, let $I_k := \{i \in I \mid k-d \leq i \leq k\}$ denote the indices in I between $k-d$ and k .

4: For every $k \in I$, let $r_k := |I_k|$ and define

$$W_k := \bigcup_{i \in I: 1 \leq i < k-d} U_i \quad \text{and}$$

$$\mathcal{Q}_k := \left\{ \left(W_k \cup E_1 \cup \dots \cup E_{r_k}, \ell_1 + \dots + \ell_{r_k} \right) \mid \left((E_1, \ell_1), \dots, (E_{r_k}, \ell_{r_k}) \right) \in \prod_{i \in I_k} \mathcal{E}_i \right\}.$$

5: Return the set $\mathcal{E} := \bigcup_{k \in I} \mathcal{Q}_k$.

Finally, the running time of the algorithm is also bounded by

$$\left(\text{amls}(\alpha, c, \zeta') \right)^{n+o(n)} = \mathcal{O} \left(\left(\text{amls}(\alpha, c, \beta) + \varepsilon \right)^n \right)$$

which proves the lemma. ◀

6 Discussion

In this paper, we study weighted monotone subset minimization problems, where given a universe U with n elements and a weight function $\mathbf{w}: U \rightarrow \mathbb{N}$, the goal is to find a subset $S \subseteq U$ which satisfies a certain fixed property and has a minimum weight. For such problems, we show that the Approximate Monotone Local Search framework of Esmer et al. [12] can be extended to the weighted setting. In particular, given a parameterized α -approximate extension algorithm, that runs in time $c^k \cdot n^{\mathcal{O}(1)}$ and outputs a solution whose weight is at most $\beta \cdot \mathbf{w}(\text{OPT})$ where OPT is a solution of size at most k and minimum weight, one can design an exponential β -approximation algorithm that runs faster than the proposed (natural) brute-force algorithm.

Note that for most of our applications, the parameterized approximation algorithms that are available in the literature [1, 14, 18, 21] provide bi-objective guarantees which are stronger than the requirements from the α -approximate extension algorithm. In particular, these algorithms run in time $c^k \cdot n^{\mathcal{O}(1)}$ and output a solution of size at most $\gamma \cdot k$ and weight at most $\beta \cdot W$, if a solution of size at most k and weight at most W exists. That is, they (approximately) optimize the size and weight of the output solution *simultaneously*.

This leads to the following natural question. Consider more restrictive weighted monotone subset minimization problems where given a universe U on n vertices, a weight function \mathbf{w} on the elements of the universe, the goal is to find a subset of the universe of size at most k

that minimizes the weight and satisfies a certain fixed property. What is the analogue of brute-force in this setting? Can bi-objective parameterized approximation algorithms for weighted monotone subset minimization problems be used to design faster (than brute force) exponential approximation algorithms in this restrictive setting? What happens if we extend this setting to a bi-criteria optimization setting?

References

- 1 Akanksha Agrawal, Sudeshna Kolay, Daniel Lokshtanov, and Saket Saurabh. A faster FPT algorithm and a smaller kernel for block graph vertex deletion. In Evangelos Kranakis, Gonzalo Navarro, and Edgar Chávez, editors, *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, volume 9644 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2016. doi:10.1007/978-3-662-49529-2_1.
- 2 Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. *J. ACM*, 62(5):42:1–42:25, 2015. doi:10.1145/2775105.
- 3 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discret. Math.*, 12(3):289–297, 1999. doi:10.1137/S0895480196305124.
- 4 Nikhil Bansal, Parinya Chalermsook, Bundit Laekhanukit, Danupon Nanongkai, and Jesper Nederlof. New tools and connections for exponential-time approximation. *Algorithmica*, 81(10):3993–4009, 2019. doi:10.1007/s00453-018-0512-8.
- 5 Reuven Bar-Yehuda and Shimon Even. A linear-time approximation algorithm for the weighted vertex cover problem. *J. Algorithms*, 2(2):198–203, 1981. doi:10.1016/0196-6774(81)90020-1.
- 6 Nicolas Bourgeois, Bruno Escoffier, and Vangelis Th. Paschos. Approximation of max independent set, min vertex cover and related problems by moderately exponential algorithms. *Discret. Appl. Math.*, 159(17):1954–1970, 2011. doi:10.1016/j.dam.2011.07.009.
- 7 Nader H. Bshouty and Lynn Burroughs. Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem. In Michel Morvan, Christoph Meinel, and Daniel Krob, editors, *STACS 98, 15th Annual Symposium on Theoretical Aspects of Computer Science, Paris, France, February 25-27, 1998, Proceedings*, volume 1373 of *Lecture Notes in Computer Science*, pages 298–308. Springer, 1998. doi:10.1007/BFb0028569.
- 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 Marek Cygan, Lukasz Kowalik, and Mateusz Wykurz. Exponential-time approximation of weighted set cover. *Inf. Process. Lett.*, 109(16):957–961, 2009. doi:10.1016/j.ipl.2009.05.003.
- 10 Bruno Escoffier, Vangelis Th. Paschos, and Emeric Tourniaire. Super-polynomial approximation branching algorithms. *RAIRO Oper. Res.*, 50(4-5):979–994, 2016. doi:10.1051/ro/2015060.
- 11 Barış Can Esmer, Ariel Kulik, Dániel Marx, Daniel Neuen, and Roohani Sharma. Faster exponential-time approximation algorithms using approximate monotone local search. 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 50:1–50:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ESA.2022.50.
- 12 Barış Can Esmer, Ariel Kulik, Dániel Marx, Daniel Neuen, and Roohani Sharma. Optimally repurposing existing algorithms to obtain exponential-time approximations. *CoRR*, abs/2306.15331, 2023. To be published at SODA 2024. arXiv:2306.15331, doi:10.48550/arXiv.2306.15331.

- 13 Guy Even, Joseph Naor, and Leonid Zosin. An 8-approximation algorithm for the subset feedback vertex set problem. *SIAM J. Comput.*, 30(4):1231–1252, 2000. doi:10.1137/S0097539798340047.
- 14 Fedor V. Fomin, Serge Gaspers, Dieter Kratsch, Mathieu Liedloff, and Saket Saurabh. Iterative compression and exact algorithms. *Theor. Comput. Sci.*, 411(7-9):1045–1053, 2010. doi:10.1016/j.tcs.2009.11.012.
- 15 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. *J. ACM*, 66(2):8:1–8:23, 2019. doi:10.1145/3284176.
- 16 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010. doi:10.1007/978-3-642-16533-7.
- 17 Ariel Kulik and Hadas Shachnai. Analysis of two-variable recurrence relations with application to parameterized approximations. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 762–773. IEEE, 2020. doi:10.1109/FOCS46700.2020.00076.
- 18 Daniel Lokshtanov, Pranabendu Misra, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. FPT-approximation for FPT problems. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 199–218. SIAM, 2021. doi:10.1137/1.9781611976465.14.
- 19 Pasin Manurangsi and Luca Trevisan. Mildly exponential time approximation algorithms for vertex cover, balanced separator and uniform sparsest cut. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, volume 116 of *LIPICs*, pages 20:1–20:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.APPROX-RANDOM.2018.20.
- 20 Rolf Niedermeier and Peter Rossmanith. On efficient fixed-parameter algorithms for weighted vertex cover. *J. Algorithms*, 47(2):63–77, 2003. doi:10.1016/S0196-6774(03)00005-1.
- 21 Hadas Shachnai and Meirav Zehavi. A multivariate framework for weighted FPT algorithms. *J. Comput. Syst. Sci.*, 89:157–189, 2017. doi:10.1016/j.jcss.2017.05.003.
- 22 Magnus Wahlström. A tighter bound for counting max-weight solutions to 2sat instances. In Martin Grohe and Rolf Niedermeier, editors, *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings*, volume 5018 of *Lecture Notes in Computer Science*, pages 202–213. Springer, 2008. doi:10.1007/978-3-540-79723-4_19.

A Missing Proofs from Section 5

The proof of Lemma 21 contains repeated arguments from the proof of Lemma 12. To enhance the overall readability and reduce the notational burden, we keep the common arguments in both proofs. Finally, we need the following technical lemma whose proof is given in the full version of the paper.

► **Lemma 23.** *Let $g, d: \mathbb{N} \rightarrow \mathbb{N}$ be two functions such that $g \in n + o(n)$ and $d \in o(n)$.*

We define $f: \mathbb{N} \rightarrow \mathbb{N}$ via

$$f(n) = \max_{n=n_1+\dots+n_{d(n)}} \sum_{i=1}^{d(n)} g(n_i).$$

Then $f \in n + o(n)$.

Proof of Lemma 21. We claim that Algorithm 1 satisfies the conditions stated in the lemma.

▷ **Claim 24.** The set \mathcal{E} returned by Algorithm 1 is an $(\alpha, (1 + \delta) \cdot \zeta)$ -extension family of U and \mathbf{w} .

Proof. Let $S \subseteq U$ be a set. We argue that there exists $(T, \ell) \in \mathcal{E}$ such that S, T and ℓ satisfy (8). By the definition of γ and d , it holds that

$$\gamma^d \geq \left(1 + \frac{\delta}{2}\right)^{(2/\delta) \cdot \log(2n/\delta)} \geq 2^{\log(2n/\delta)} = \frac{2n}{\delta} \quad (10)$$

using that $(1 + \frac{1}{x})^x \geq 2$ for all $x \geq 1$. Let $k \in I$ be the largest index such that $S \cap U_k \neq \emptyset$. It holds that

$$\mathbf{w}(W_k) < n \cdot \gamma^{k-d} \leq \frac{\delta}{2} \cdot \gamma^k \leq \frac{\delta}{2} \cdot \mathbf{w}(S) \quad (11)$$

where the first inequality follows from the fact that $|W_k| \leq n$ and each $u \in W_k$ belongs to a set U_i where $i \leq k - d - 1$ and therefore $\mathbf{w}(u) < \gamma^{k-d-1+1} = \gamma^{k-d}$ by (9). The second inequality follows from (10) and finally the last inequality holds because by the definition of k , there exists $u \in S \cap U_k$ such that $\mathbf{w}(S) \geq \mathbf{w}(u) \geq \gamma^k$ by (9).

For $i \in I_k$ let $S_i := S \cap U_i$. Since \mathcal{E}_i is an (α, ζ) -extension family of U_i and the *uniform weight function*, for each $i \in I_k$ there exists $(T_i, \ell_i) \in \mathcal{E}_i$ such that

$$\begin{aligned} |S_i \setminus T_i| &\leq \ell_i \\ |T_i| + \alpha \cdot |S_i \setminus T_i| &\leq \zeta \cdot |S_i|. \end{aligned} \quad (12)$$

Let $\bar{T} := \bigcup_{i \in I_k} T_i$ and define

$$\begin{aligned} T &:= W_k \cup \bar{T} \\ \ell &:= \sum_{i \in I_k} \ell_i. \end{aligned}$$

By definition of \mathcal{Q}_k in Algorithm 1 it holds that $(T, \ell) \in \mathcal{Q}_k \subseteq \mathcal{E}$. Observe that

$$\begin{aligned} S \setminus T &= (S \setminus T) \cap U \\ &= (S \setminus T) \cap \left(W_k \cup \left(\bigcup_{i \in I_k} U_i \right) \right) \\ &= \left((S \setminus T) \cap W_k \right) \cup \left((S \setminus T) \cap \left(\bigcup_{i \in I_k} U_i \right) \right) \\ &= (S \setminus T) \cap \left(\bigcup_{i \in I_k} U_i \right) \\ &= \bigcup_{i \in I_k} (U_i \cap (S \setminus T)) \\ &= \bigcup_{i \in I_k} S_i \setminus T_i \end{aligned} \quad (13)$$

where the second equality follows from $S \subseteq \bigcup_{1 \leq i \leq k} U_i = W_k \cup \left(\bigcup_{k-d \leq i \leq k} U_i \right)$ and the fourth inequality holds because $W_k \subseteq T$ which further implies $(S \setminus T) \cap W_k = \emptyset$. Therefore it holds that

$$|S \setminus T| = \left| \bigcup_{i \in I_k} S_i \setminus T_i \right| \leq \sum_{i \in I_k} |S_i \setminus T_i| \leq \sum_{i \in I_k} \ell_i = \ell.$$

where the second inequality follows from (12).

17:20 Approximate Monotone Local Search for Weighted Problems

For all $i \in I_k$ we also have that

$$\begin{aligned}
 \mathbf{w}(T_i) + \alpha \cdot \mathbf{w}(S_i \setminus T_i) &\leq |T_i| \cdot \gamma^{i+1} + \alpha \cdot \gamma^{i+1} \cdot |S_i \setminus T_i| \\
 &\leq \gamma^{i+1} \cdot (|T_i| + \alpha \cdot |S_i \setminus T_i|) \\
 &\leq \gamma^{i+1} \cdot \zeta \cdot |S_i| \\
 &\leq \zeta \cdot \gamma \cdot \mathbf{w}(S_i)
 \end{aligned} \tag{14}$$

where the third inequality follows from (12). Therefore we have that

$$\begin{aligned}
 \mathbf{w}(T) + \alpha \cdot \mathbf{w}(S \setminus T) &= \mathbf{w}(W_k) + \mathbf{w}\left(\bigcup_{i \in I_k} T_i\right) + \alpha \cdot \mathbf{w}(S \setminus T) \\
 &\leq \mathbf{w}(W_k) + \sum_{i \in I_k} \left(\mathbf{w}(T_i) + \alpha \cdot \mathbf{w}(S_i \setminus T_i)\right) && \text{by (13)} \\
 &< \frac{\delta}{2} \cdot \mathbf{w}(S) + \sum_{i \in I_k} \left(\mathbf{w}(T_i) + \alpha \cdot \mathbf{w}(S_i \setminus T_i)\right) && \text{by (11)} \\
 &\leq \frac{\delta}{2} \cdot \mathbf{w}(S) + \sum_{i \in I_k} \zeta \cdot \gamma \cdot \mathbf{w}(S_i) && \text{by (14)} \\
 &< \zeta \cdot \frac{\delta}{2} \cdot \mathbf{w}(S) + \sum_{i \in I_k} \zeta \cdot \gamma \cdot \mathbf{w}(S_i) && \text{since } \zeta > 1 \\
 &\leq \zeta \cdot \frac{\delta}{2} \cdot \mathbf{w}(S) + \zeta \cdot \gamma \cdot \mathbf{w}(S) \\
 &\leq (1 + \delta) \cdot \zeta \cdot \mathbf{w}(S)
 \end{aligned}$$

which proves the claim. \triangleleft

▷ **Claim 25.**

$$\mathbf{cost}_c(\mathcal{E}) \leq \left(\mathbf{amls}(\alpha, c, \zeta)\right)^{n+o(n)}.$$

Proof. By Lemma 20, for each $i \in I$ it holds that

$$\mathbf{cost}_c(\mathcal{E}_i) \leq \left(\mathbf{amls}(\alpha, c, \zeta)\right)^{n_i+o(n_i)}. \tag{15}$$

For $i \in \mathbb{Z}_{\geq 0} \setminus I$ we define $\mathcal{E}_i = \{(\emptyset, 0)\}$. By doing so, we can make the assumption, without loss of generality, that \mathcal{E}_i is nonempty for all $i \in \mathbb{Z}_{\geq 0}$. Note that this assumption does not have any effect on the value of \mathbf{cost}_c and it simplifies the following analysis. Also let \mathbb{E} denote the Cartesian product of \mathcal{E}_{k-d} up to \mathcal{E}_k , i.e., $\mathbb{E} := \prod_{i \in \{k-d, \dots, k\}} \mathcal{E}_i$.

With this assumption, for all $k \in I$, we have

$$\begin{aligned}
 \mathbf{cost}_c(\mathcal{Q}_k) &= \sum_{((E_{k-d}, \ell_{k-d}), \dots, (E_{k-1}, \ell_{k-1}), (E_k, \ell_k)) \in \mathbb{E}} c^{\ell_{k-d} + \dots + \ell_{k-1} + \ell_k} \\
 &= \sum_{((E_{k-d}, \ell_{k-d}), \dots, (E_{k-1}, \ell_{k-1}), (E_k, \ell_k)) \in \mathbb{E}} c^{\ell_{k-d}} \cdot \dots \cdot c^{\ell_{k-1}} \cdot c^{\ell_k} \\
 &= \prod_{j=k-d}^k \sum_{(E, \ell) \in \mathcal{E}_j} c^\ell
 \end{aligned}$$

$$\begin{aligned}
&= \prod_{j=k-d}^k \text{cost}_c(\mathcal{E}_j) \\
&= \prod_{j \in I_k} \text{cost}_c(\mathcal{E}_j)
\end{aligned}$$

By (15) we obtain

$$\begin{aligned}
\text{cost}_c(\mathcal{Q}_k) &= \prod_{i \in I_k} \text{cost}_c(\mathcal{E}_i) \\
&\leq \prod_{i \in I_k} \left(\text{amls}(\alpha, c, \zeta) \right)^{n_i + o(n_i)} \\
&= \left(\text{amls}(\alpha, c, \zeta) \right)^{n + o(n)}
\end{aligned}$$

where the last step follows from Lemma 23.

Finally, it holds that

$$\text{cost}_c(\mathcal{E}) = \text{cost}_c\left(\bigcup_{k \in I} \mathcal{Q}_k\right) \leq \sum_{k \in I} \text{cost}_c(\mathcal{Q}_k) = \left(\text{amls}(\alpha, c, \zeta) \right)^{n + o(n)}$$

since $|I| \leq n$. ◁

▷ **Claim 26.** The running time of Algorithm 1 is $\left(\text{amls}(\alpha, c, \zeta) \right)^{n + o(n)}$.

Proof. The construction of $\{\mathcal{E}_i\}_{i \in I}$ takes time

$$\sum_{i \in I} \left(\text{amls}(\alpha, c, \zeta) \right)^{n_i + o(n_i)} \leq \left(\text{amls}(\alpha, c, \zeta) \right)^{n + o(n)}$$

by Lemma 20.

Finally, the construction of each \mathcal{Q}_k takes time proportional to its size, which is upper bounded by $\text{cost}_c(\mathcal{Q}_k)$. Then, the construction of \mathcal{E} takes at most time $\mathcal{O}(\text{cost}_c(\mathcal{E}))$, where we have

$$\text{cost}_c(\mathcal{E}) \leq \left(\text{amls}(\alpha, c, \zeta) \right)^{n + o(n)}$$

by Claim 25. All in all, the running time of Algorithm 1 is upper bounded by

$$\left(\text{amls}(\alpha, c, \zeta) \right)^{n + o(n)}. \quad \triangleleft$$

The lemma follows from Claims 24–26. ◀

B Problem Definitions

In this section, we give the problem definitions of all the problems discussed in the paper. For simplicity, we define the problems in their unweighted version. In the weighted version, the vertices are equipped with weights and we are looking for a solution S of minimum weight.

VERTEX COVER (VC)

Input: An undirected graph G .

Question: Find a minimum set S of vertices of G such that $G - S$ has no edges.

17:22 Approximate Monotone Local Search for Weighted Problems

PARTIAL VERTEX COVER (PVC)

Input: An undirected graph G and an integer $t \geq 0$.

Question: Find a minimum set S of vertices of G such that $G - S$ has at most $|E(G)| - t$ many edges.

d -HITTING SET (d -HS)

Input: A universe U and set family $\mathcal{F} \subseteq \binom{U}{\leq d}$.

Question: Find a minimum set $S \subseteq U$ such that for each $F \in \mathcal{F}$, $S \cap F \neq \emptyset$.

FEEDBACK VERTEX SET (FVS)

Input: An undirected graph G .

Question: Find a minimum set S of vertices of G such that $G - S$ is an acyclic graph.

SUBSET FEEDBACK VERTEX SET (SUBSET FVS)

Input: An undirected graph G and a set $T \subseteq V(G)$.

Question: Find a minimum set S of vertices of G such that $G - S$ has no cycle that contains at least one vertex of T .

TOURNAMENT FEEDBACK VERTEX SET (TFVS)

Input: A tournament graph G .

Question: Find a minimum set S of vertices of G such that $G - S$ is an acyclic tournament.

DIRECTED FEEDBACK VERTEX SET (DFVS)

Input: A directed graph G .

Question: Find a minimum set S of vertices of G such that $G - S$ is a directed acyclic graph.

DIRECTED SUBSET FEEDBACK VERTEX SET (SUBSET DFVS)

Input: A directed graph G and a set $T \subseteq V(G)$.

Question: Find a minimum set S of vertices of G such that $G - S$ has no directed cycle that contains at least one vertex of T .

DIRECTED ODD CYCLE TRANSVERSAL (DOCT)

Input: A directed graph G .

Question: Find a minimum set S of vertices of G such that $G - S$ has no directed cycle of odd length.

MULTICUT

Input: An undirected graph G and a set $\mathcal{P} \subseteq V(G) \times V(G)$.

Question: Find a minimum set S of vertices of G such that $G - S$ has no path from u to v for any $(u, v) \in \mathcal{P}$

For the next problems, we require some additional definitions. A graph G is a *cluster graph* if every connected component of G is a complete graph. A *cograph* is a graph G which does not contain P_4 (a path on 4 vertices) as an induced subgraph. Finally, a graph G is a *split graph* if the vertex set can be partitioned into two sets $V(G) = I \uplus C$ such that I is an independent set and C is a clique in G .

CLUSTER GRAPH VERTEX DELETION

Input: An undirected graph G .**Question:** Find a minimum set S of vertices of G such that $G - S$ is a cluster graph.


COGRAPH VERTEX DELETION

Input: An undirected graph G .**Question:** Find a minimum set S of vertices of G such that $G - S$ is a cograph.

SPLIT VERTEX DELETION

Input: An undirected graph G .**Question:** Find a minimum set S of vertices of G such that $G - S$ is a split graph.



Consistency Checking Problems: A Gateway to Parameterized Sample Complexity

Robert Ganian   

Technische Universität Wien, Austria

Liana Khazaliya   

Technische Universität Wien, Austria

Kirill Simonov   

Hasso Plattner Institute, Universität Potsdam, Germany

Abstract

Recently, Brand, Ganian and Simonov introduced a parameterized refinement of the classical PAC-learning sample complexity framework. A crucial outcome of their investigation is that for a very wide range of learning problems, there is a direct and provable correspondence between fixed-parameter PAC-learnability (in the sample complexity setting) and the fixed-parameter tractability of a corresponding “consistency checking” search problem (in the setting of computational complexity). The latter can be seen as generalizations of classical search problems where instead of receiving a single instance, one receives multiple yes- and no-examples and is tasked with finding a solution which is consistent with the provided examples.

Apart from a few initial results, consistency checking problems are almost entirely unexplored from a parameterized complexity perspective. In this article, we provide an overview of these problems and their connection to parameterized sample complexity, with the primary aim of facilitating further research in this direction. Afterwards, we establish the fixed-parameter (in)-tractability for some of the arguably most natural consistency checking problems on graphs, and show that their complexity-theoretic behavior is surprisingly very different from that of classical decision problems. Our new results cover consistency checking variants of problems as diverse as $(k-)$ PATH, MATCHING, 2-COLORING, INDEPENDENT SET and DOMINATING SET, among others.

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

Keywords and phrases consistency checking, sample complexity, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.18

Related Version *Full Version:* <https://arxiv.org/abs/2308.11416>

Funding *Robert Ganian:* Austrian Science Fund (FWF) [Y1329].

Liana Khazaliya: Vienna Science and Technology Fund (WWTF) [10.47379/ICT22029]; Austrian Science Fund (FWF) [Y1329]; European Union’s Horizon 2020 COFUND programme [LogiCS@TUWien, grant agreement No. 101034440].

Kirill Simonov: DFG Research Group ADYN via grant DFG 411362735.



1 Introduction

While the notion of time complexity is universally applicable and well studied across the whole spectrum of theoretical computer science, on its own it cannot capture the performance of the kinds of algorithms typically studied in the context of machine learning: **learning algorithms**. That is the domain of sample complexity, and here we will focus on the notion of (efficient) *PAC learning* [19, 13] – arguably the most classical, fundamental and widely known sample complexity framework. An important trait of PAC learning is that while it



© Robert Ganian, Liana Khazaliya, and Kirill Simonov;
licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 18; pp. 18:1–18:17

Leibniz International Proceedings in Informatics



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

is built on different principles than time complexity, the two frameworks are connected in a way which allows us to translate intractability and tractability results from one domain to another. It is precisely this connection that gave rise to famous lower bounds in the PAC learning setting, such as the inability to efficiently and properly learn 3-term DNF and 3-clause CNF formulas [17, 2] under the assumption that $P \neq NP$, and *consistency checking problems* form the pillar of this connection.

Given the success of parameterized complexity as a concept generalizing classical time complexity analysis, it would seem natural to ask whether its principles can also be used to obtain a deeper understanding of efficient PAC-learnability. Brand, Ganian and Simonov [6] very recently introduced the foundations for a parameterized theory of PAC learning, which crucially also includes a bridge to parameterized complexity theory in the usual time complexity setting. The primary goal of this article is to show how the parameterized complexity paradigm can be used to draw new boundaries of tractability in the PAC learning domain, and to provide the parameterized algorithms community with an understanding of the parameterized consistency checking problems which allow us to travel between the sample and time complexity settings in the parameterized regime. We showcase the tools that can be used to deal with parameterized consistency checking problems and the obstacles that await there in the domain of graph problems, where we obtain new algorithmic upper and lower bounds for consistency checking variants of multiple natural problems on graphs.

A Gentle Introduction to PAC Learning. It will be useful to set the stage with a high-level and informal example of the setting in which PAC learning operates¹. Let us imagine we would like to “learn” a way of labeling points in a plane as either “good” or “bad”, knowing that the good points are precisely those contained in some unknown axis-parallel rectangle R in the plane. A learning algorithm in the PAC regime would be allowed to ask for a set of correctly labeled sample points, each of which would be drawn from some unknown distribution D , and would attempt to use these to “learn” R (so that it can use it to label any point that it looks at, even those which were not given as samples). This mental experiment is useful since it immediately clarifies that

- there is some probability that a PAC learning algorithm completely fails, since the samples we receive could be non-representative (*for instance, there is a non-zero probability that even if D is uniform and R is small, the sample points could all be drawn from inside R*), and
- even if a PAC learning algorithm intuitively “works correctly”, it is essentially guaranteed that it will not classify some samples (i.e., the sample points) correctly (*for instance, there could be points that lie close to the exact boundary of R which are unlikely to be drawn as samples based on D , making it impossible to obtain the exact position of R*).

Given these natural limitations, we can informally explain what it means for a learning problem to be *efficiently PAC-learnable*: it admits an algorithm which

1. takes as input a sample size n , a confidence measure δ and an accuracy measure ε ,
2. runs in time $(n + \frac{1}{\delta} + \frac{1}{\varepsilon})^{\mathcal{O}(1)}$ and asks for $(n + \frac{1}{\delta} + \frac{1}{\varepsilon})^{\mathcal{O}(1)}$ samples, and then
3. outputs something which will, with probability at least $1 - \delta$, “work correctly” in almost all cases (measured by ε).

It needs to be clarified that beyond the study of efficient PAC learnability, a substantial amount of fundamental work in the PAC learning direction has also been carried out on whether a problem is PAC learnable at all [4, 12, 1], on the distinction between so-called

¹ Formal definitions are provided in Section 2.

proper and improper learning [13, 5], and on many other aspects and considerations that lie outside of the scope of this paper. Here, our focus lies on how the gap between efficient and “non-efficient” PAC-learnability of learning problems can be bridged by the parameterized PAC learning framework of Brand, Galian and Simonov [6], and the associated study of consistency checking problems.

To illustrate how parameterized complexity can be used here, let us turn to a different example of a simple learning problem that is based on the idea of representing cyber-attacks as graphs proposed, e.g., by Sheyner and Wing [18, 21]. Assume we have a network consisting of n nodes which is protected by k hidden defense nodes. A cyberattack on this network can be represented as a set of edges over the n nodes, and is evaluated as successful if and only if an edge in that attack is not incident to any defense node (i.e., an attack fails if and only if the defense nodes form a vertex cover of the attack edges). Individual samples represent attacks made on the network, and the learning task is to identify all the defense nodes. This problem corresponds to VERTEX COVER LEARNING [8], which Brand, Galian and Simonov showed to admit a PAC learning algorithm which requires polynomially many samples but time $2^k \cdot (n + \frac{1}{\delta} + \frac{1}{\epsilon})^{\mathcal{O}(1)}$ where k is the size of the sought-after vertex cover [6]. This is a prototypical representative of the class $\text{FPT-PAC}_{\text{time}}$. We remark that in the context of PAC learning, one explicitly distinguishes between the time required by the learning algorithm and the number of samples it uses, as the latter may in some contexts be much more difficult to obtain. A picture of the parameterized complexity landscape above efficient PAC learnability is provided later together with the formal definitions (see Figure 1).

Crucially, whenever we are dealing with a learning problem $\mathcal{P}_{\text{learn}}$ where the size of the hypothesis space (i.e., the number of “possible outputs”) is upper-bounded by a certain function (see Theorem 11), the parameterized sample complexity of $\mathcal{P}_{\text{learn}}$ can be directly and formally linked to the parameterized time complexity of the consistency checking variant $\mathcal{P}_{\text{cons}}$ of the same problem [6], where the task is to compute a “solution” (a hypothesis) which is consistent with a provided set of positive and negative examples. This motivates the systematic study of parameterized consistency checking problems, an area which has up to now remained almost entirely unexplored from the perspective of fixed-parameter (in-)tractability.

The Parameterized Complexity of Consistency Checking on Graphs. A few initial examples of parameterized consistency checking problems have been solved by the theory-building work of Brand, Galian and Simonov [6]; in particular, they showed that consistency checking for vertex-deletion problems where the base class \mathcal{H} can be characterized by a finite set of forbidden induced subgraphs is fixed-parameter tractable (which implies the aforementioned fact that VERTEX COVER LEARNING is in $\text{FPT-PAC}_{\text{time}}$), but no analogous result can be obtained for all classes \mathcal{H} characterized by a finite set of forbidden minors unless $\text{FPT} \neq \text{W}[1]$.

In this article, we expand on these results by establishing the fixed-parameter (in-)tractability of consistency checking for several other classical graph problems whose decision versions are well-known to the parameterized complexity community. The aim here is to showcase how parameterized upper- and lower-bound techniques fare when dealing with these new kinds of problems.

It is important to note that the tractability of consistency checking requires the tractability of the corresponding decision/search problem (as the latter can be seen as a special case of consistency checking), but the former can be much more algorithmically challenging than the latter: many trivial decision problems become computationally intractable in the consistency checking regime. We begin by illustrating this behavior on the classical 2-COLORING problem,

i.e., the task of partitioning the vertices of the graph into two independent sets. We show that while consistency checking for 2-COLORING is intractable (and hence a 2-coloring is not efficiently PAC-learnable), consistency checking for SPLIT GRAPH, i.e., the task of partitioning the vertices into an independent set and a clique, is polynomial-time tractable.

Moving on to parameterized problems, we begin by considering three classical edge search problems, notably MATCHING, (k -)PATH and EDGE CLIQUE COVER. In the classical decision or search settings, the first problem is polynomial-time solvable while the latter two admit well-known fixed-parameter algorithms. Interestingly, we show that consistency checking for the former two problems is $W[2]$ -hard², but is fixed-parameter tractable for the third, i.e., EDGE CLIQUE COVER.

Next, we turn our attention to the behavior of two classical vertex search problems, specifically INDEPENDENT SET and DOMINATING SET. While both problems are fixed-parameter intractable already in the classical search regime, here we examine their behavior on bounded-degree graphs (where they are well-known to be fixed-parameter tractable). Again, the consistency checking variants of these problems on bounded-degree graphs exhibit a surprising complexity-theoretic behavior: DOMINATING SET is FPT, but INDEPENDENT SET is $W[2]$ -hard even on bounded-degree graphs.

As the final contribution of the paper, we show that most of the aforementioned consistency checking lower bounds can be overcome if one additionally parameterizes by the number of negative samples. In particular, we obtain fixed-parameter consistency checking algorithms for 2-COLORING, MATCHING and (k -)PATH when we additionally assume that the number of negative samples is upper-bounded by the parameter. On the other hand, INDEPENDENT SET remains fixed-parameter intractable (at least $W[1]$ -hard) even under this additional restriction. As our final result, we show that INDEPENDENT SET becomes fixed-parameter tractable if we instead consider the total number of samples (i.e., both positive and negative) as an additional parameter. The proofs of these results are more involved than those mentioned in the previous paragraphs and rely on auxiliary graph constructions in combination with color coding. We remark that the parameterization by the number of negative samples in the consistency checking regime could be translated into a corresponding parameterization of the distribution in the PAC learning framework. A summary of our individual results for consistency checking problems is provided in Table 1.

Related Work. The connection between parameterized learning problems and parameterized consistency checking was also hinted at in previous works that studied the (parameterized) sample complexity of learning juntas [3] or learning first-order logic [20]. Moreover, the problem of computing optimal decision trees, which has received a significant amount of recent attention [16, 10], can also be seen as a consistency checking problem where the sought-after solution is a decision tree.

For space reasons, results marked with a “★” are proved in the Full Version³.

2 Preliminaries

We assume familiarity with basic graph terminology [9] and parameterized complexity theory [7]. We use $[t]$ to denote the set $\{1, \dots, t\}$. For brevity, we will denote sets of tuples of the form $\{(\alpha_1, \beta_1), \dots, (\alpha_t, \beta_t)\}$ as $(\alpha_i, \beta_i)_{i \in [t]}$, and the set of two-element subsets of a set Z

² More precisely, a fixed-parameter algorithm for either of these problems would imply $FPT=W[1]$ (see Section 4).

³ <https://arxiv.org/abs/2308.11416>

■ **Table 1** An overview of the concrete results obtained for consistency checking problems in this article, where the columns provide a comparison between the complexity of the decision/search variant, the consistency checking variant, and the consistency checking variant where the number of negative samples is taken as an additional parameter. Problems marked with “[degree]” are considered over bounded-degree input graphs/samples, and the “*” marks that the problem becomes fixed-parameter tractable when additionally parameterized by the total number of samples. The lower bounds stated in the table are simplified; the precise formal statements are provided in the appropriate theorems.

Problem	Decision/Search	Consistency Checking	Consistency Checking[samples]
2-COLORING	P	NP-hard (Thm. 12)	FPT (Thm. 16)
SPLIT GRAPH	P	P (Thm. 14)	—
MATCHING	P	W[2]-hard (Thm. 17)	FPT (Thm. 20)
(k)-PATH	FPT	W[2]-hard (Thm. 18)	FPT (Thm. 20)
EDGE CLIQUE COVER	FPT	FPT (Thm. 19)	—
INDEPENDENT SET[degree]	FPT	W[2]-hard (Thm. 22)	W[1]-hard* (Thm. 24, 25)
DOMINATING SET[degree]	FPT	FPT (Thm. 23)	—

as $\binom{Z}{2}$. As basic notation and terminology, we set $\{0, 1\}^* = \bigcup_{m \in \mathbb{N}} \{0, 1\}^m$. A *distribution* on $\{0, 1\}^n$ is a mapping $\mathcal{D}_n : \{0, 1\}^n \rightarrow [0, 1]$ such that $\sum_{x \in \{0, 1\}^n} \mathcal{D}_n(x) = 1$, and the *support* of \mathcal{D}_n is the set $\text{supp } \mathcal{D}_n = \{x \mid \mathcal{D}_n(x) > 0\}$.

2.1 Consistency Checking

While the original motivation for consistency checking problems originates from specific applications in PAC learning, one can define a consistency checking version of an arbitrary *search problem*.

In a search problem, we are given an instance $I \in \{0, 1\}^*$, and the task is to find a solution $S \in \{0, 1\}^*$, where the solution is verified by a predicate $\phi(\cdot, \cdot)$, so that $\phi(I, S)$ is true if and only if S is a solution to I . Since our focus here will lie on problems which are in NP, the predicate $\phi(\cdot, \cdot)$ will in all cases be polynomial-time computable. In the context of graph problems, I will typically be a graph (possibly with some auxiliary information such as edge weights or the bound on solution size), and S could be a set of vertices, a set of edges, a partitioning of the vertex set, etc. For example, in the search version of the VERTEX COVER problem the input is a graph G together with a bound k on the size of the target vertex cover, potential solutions are subsets of $V(G)$, and a subset S is a solution if and only if the size of S is k and S covers all edges of the graph G . One can then write the verifying predicate as

$$\phi((G, k), S) = (S \subset V(G)) \wedge (|S| = k) \wedge (\forall \{u, v\} \in E(G), \{u, v\} \cap S \neq \emptyset).$$

For a search problem \mathcal{P} , we define the corresponding consistency checking problem $\mathcal{P}_{\text{cons}}$ as follows. Instead of receiving a single instance $I \in \{0, 1\}^*$ as input, we receive a set of labeled samples $\mathcal{I} = \{(I_1, \lambda_1), (I_2, \lambda_2), \dots, (I_t, \lambda_t)\}$ where each $I_i, i \in [t]$, is an element of $\{0, 1\}^*$ and $\lambda_i \in \{0, 1\}$. The task is to compute a (*consistent*) *solution* $S \subset \{0, 1\}^*$ such that $\phi(I_i, S)$ holds if and only if $\lambda_i = 1$, for each $i \in [t]$, or to correctly determine that no such solution exists.

In the example of VERTEX COVER, for each $i \in [t]$, the instance is the pair (G_i, k_i) , so that the target solution has to be a vertex subset⁴ of $V(G_i)$, of size k_i , and it has to cover all edges of G_i , for each $i \in [t]$. Since vertices in all G_i 's and S are implicitly associated with their respective counterparts in the other graphs, we can instead treat the graphs G_i as defined over the same vertex set. Also, for instances $i \in [t]$ where $\lambda_i = 1$, if their values of k_i mismatch, then there is clearly no solution; and for those $i \in [t]$ with $\lambda_i = 0$, if the value k_i does not match the respective value of a positive sample, then the condition for λ_i is always satisfied. Therefore, we can equivalently reformulate the consistency checking version of VERTEX COVER as follows: Given the vertex set V , a number k , and a sequence of labeled edge sets $(E_1, \lambda_1), \dots, (E_t, \lambda_t)$, over V , is there a subset $S \subset V$ of size exactly k , so that S covers all edges of E_i if and only if $\lambda_i = 1$, for each $i \in [t]$?

One can immediately observe that the polynomial-time tractability of a search problem is a prerequisite for the polynomial-time tractability of the corresponding consistency checking problem. At the same time, the exact definition of the search problem (and in particular the solution S) can have a significant impact on the complexity of the consistency checking problem. We remark that there are two possible ways one can parameterize a consistency checking problem: one either uses the parameter to restrict the sought-after solution S , or the input \mathcal{I} . Each of these approaches can be tied to a parameterization of the corresponding PAC learning problem (see Subsection 2.3).

Formally, we say that $(\mathcal{P}_{\text{cons}}, \kappa, \lambda)$ is a parameterized consistency checking problem, where $\mathcal{P}_{\text{cons}}$ is a consistency checking problem, κ maps solutions $S \in \{0, 1\}^*$ to natural numbers, and λ maps lists of labeled instances $((I_1, \lambda_1), \dots, (I_t, \lambda_t))$, $I_i \in \{0, 1\}^*$, $\lambda_i \in \{0, 1\}$, to natural numbers. The input is then a list of labeled instances $\mathcal{L} = ((I_1, \lambda_1), \dots, (I_t, \lambda_t))$ together with parameters k, ℓ , such that $\ell = \lambda(\mathcal{L})$, and the task is to find a consistent solution S with $\kappa(S) = k$. For example, k could be a size bound on the targeted solution, and ℓ could be the maximum degree in any of the given graphs or the number of instances with $\lambda_i = 0$.

2.2 PAC-Learning

The remainder of this section is dedicated to a more formal introduction of the foundations of parameterized PAC learning theory and its connection to parameterized consistency checking problems. We note that while the content of the following subsections is important to establish the implications and corollaries of the results obtained in the article, readers who are interested solely in the obtained complexity-theoretic upper and lower bounds for consistency checking problems can safely skip them and proceed directly to Section 3.

To make the connection between consistency checking problems and parameterized sample complexity clear, we first recall the formalization of the classical theory of PAC learning [19, 14].

► **Definition 1.** *A concept is an arbitrary Boolean function $c : \{0, 1\}^n \rightarrow \{0, 1\}$. An assignment $x \in \{0, 1\}^n$ is called a positive sample for c if $c(x) = 1$, and a negative sample otherwise. A concept class \mathcal{C} is a set of concepts. For every $m \in \mathbb{N}$, we write $\mathcal{C}_m = \mathcal{C} \cap \mathcal{B}_m$, where \mathcal{B}_m is the set of all m -ary Boolean functions.*

► **Definition 2.** *Let \mathcal{C} be a concept class. A surjective mapping $\rho : \{0, 1\}^* \rightarrow \mathcal{C}$ is called a representation scheme of \mathcal{C} .*

We call each r with $\rho(r) = c$ a representation of concept c .

⁴ The property of being a subset is given by the implicit encoding in $\{0, 1\}^*$, e.g., vertices in all $V(G_i)$ and S are indexed by integers, and is defined in the same way across all instances. We thus say that S could be a subset of all $V(G_i)$ even though, formally speaking, these are disjoint sets.

► **Definition 3.** A learning problem is a pair (\mathcal{C}, ρ) , where \mathcal{C} is a concept class and ρ is a representation scheme for \mathcal{C} .

► **Definition 4.** A learning algorithm for a learning problem (\mathcal{C}, ρ) is a randomized algorithm such that:

1. It obtains the values n, ε, δ as inputs, where n is an integer and $0 < \varepsilon, \delta \leq 1$ are rational numbers.
2. It has access to a hidden representation r^* of some concept $c^* = \rho(r^*)$ and a hidden distribution \mathcal{D}_n on $\{0, 1\}^n$ through an oracle that returns labeled samples $(x, c^*(x))$, where $x \in \{0, 1\}^n$ is drawn at random from \mathcal{D}_n .
3. The output of the algorithm is a representation of some concept, called its hypothesis.

When dealing with individual instances of a learning problem, we will use $s = |r^*|$ to denote the size of the hidden representation.

► **Definition 5.** Let \mathcal{A} be a learning algorithm. Fix a hidden hypothesis c^* and a distribution on $\{0, 1\}^n$. Let h be a hypothesis output by \mathcal{A} and $c = \rho(h)$ be the concept h represents. We define

$$\text{err}_h = \mathbb{P}_{x \sim \mathcal{D}_n}(c(x) \neq c^*(x))$$

as the probability of the hypothesis and the hidden concept disagreeing on a sample drawn from \mathcal{D}_n , the so-called generalization error of h under \mathcal{D}_n .

The algorithm \mathcal{A} is called probably approximately correct (PAC) if it outputs a hypothesis h such that $\text{err}_h \leq \varepsilon$ with probability at least $1 - \delta$.

Usually, learning problems in this framework are regarded as tractable if they are PAC-learnable within polynomial time bounds. More precisely, we say that a learning problem L is *efficiently* PAC-learnable if there is a PAC algorithm for L that runs in time polynomial in $n, s, 1/\varepsilon$ and $1/\delta$.

Consider now a classical search problem \mathcal{P} and its consistency checking version $\mathcal{P}_{\text{cons}}$. One can naturally define the corresponding learning problem $\mathcal{P}_{\text{learn}}$: For a solution $S \in \{0, 1\}^*$, let $\phi(\cdot, S)$ be a concept and S its representation; this describes the concept class and its representation scheme. Going back to the VERTEX COVER example, for each graph size N , the concepts are represented by subsets of $[N]$ (encoded in binary). For a subset $S \subset [N]$, the respective concept c_S is a binary function that, given the encoding of an instance E , returns 1 if and only if S is a vertex cover of $G = ([N], E)$ of size k , where $[N]$ is treated as the respective “ground” vertex set of size N . A PAC-learning algorithm for this problem is thus given a vertex set $V = [N]$, an integer k , and an oracle that will produce a sequence of samples $(E_1, \lambda_1), \dots, (E_t, \lambda_t)$, where the instances E_i are drawn from a hidden distribution \mathcal{D} . With probability at least $(1 - \delta)$, the algorithm has to return a subset $S \subset [N]$ that is consistent with an instance sampled from \mathcal{D} with probability at least $(1 - \varepsilon)$. In fact, for VERTEX COVER and many other problems, it is sufficient to return a hypothesis that is consistent only with the seen samples (E_i, λ_i) , $i \in [t]$; this is formalized in the next subsection.

Naturally, we do not expect the learning version of VERTEX COVER to be efficiently PAC-learnable, as even finding a vertex cover of a certain size in a single instance is NP-hard. This motivates the introduction of parameters into the framework, which is presented next. We also recall the complexity reductions between (parameterized) consistency checking problem and its respective (parameterized) learning problem, which in particular allows to formally transfer the hardness results such as NP-hardness above.

Remark. A more general definition of learning problems is sometimes considered in the literature, where the output of a learning algorithm need not necessarily be from the same concept class \mathcal{C} (e.g., it can be a sub- or a super-class of \mathcal{C}). This is usually called *improper learning*, as opposed to the classical setting of *proper learning* defined above and considered in this article.

2.3 Parameterized PAC-Learning

We now define parameterized learning problems and recall the connection to the consistency checking problems, as given by the framework of Brand, Galian, and Simonov [6]. For brevity, we omit some of the less important technical details; interested readers can find the full technical exposition in the full description of the framework [6].

First we note that in parameterized PAC-learning, both the hidden concept and the hidden distribution can be parameterized, which is formally represented in the next definitions. We call a function κ from representations in $\{0, 1\}^*$ to natural numbers *parameterization of representations*, and a function λ assigning a natural number to every distribution on $\{0, 1\}^n$ for each n *parameterization of distributions*.

► **Definition 6** (Parameterized Learning Problems). *A parameterized learning problem is a learning problem (\mathcal{C}, ρ) together with a pair (κ, λ) , called its parameters, where κ is a parameterization of representations and λ is a parameterization of distributions.*

► **Definition 7** (Parameterized Learning Algorithm). *A parameterized learning algorithm for a parameterized learning problem $(\mathcal{C}, \rho, \kappa, \lambda)$ is a learning algorithm for (\mathcal{C}, ρ) in the sense of Definition 4. In addition to n, ε, δ , a parameterized learning algorithm obtains two inputs k and ℓ , which are promised to satisfy $k = \kappa(r^*)$ as well as $\ell = \lambda(\mathcal{D}_n)$, and the algorithm is required to always output a hypothesis h satisfying $\kappa(h) \leq k$.*

Let $\text{poly}(\cdot)$ denote the set of functions that can be bounded by non-decreasing polynomial functions in their arguments. Furthermore, $\text{fpt}(x_1, \dots, x_t; k_1, \dots, k_t)$ and $\text{xp}(x_1, \dots, x_t; k_1, \dots, k_t)$ denote those functions bounded by $f(k_1, \dots, k_t) \cdot p(x_1, \dots, x_t)$ and $p(x_1, \dots, x_t)^{f(k_1, \dots, k_t)}$, respectively, for any non-decreasing computable function f in k_1, \dots, k_t and $p \in \text{poly}(x_1, \dots, x_t)$.

► **Definition 8** ((T, S) -PAC Learnability). *Let $T(n, s, 1/\varepsilon, 1/\delta, k, \ell), S(n, s, 1/\varepsilon, 1/\delta, k, \ell)$ be any two functions taking on integer values, and non-decreasing in all of their arguments.*

A parameterized learning problem $\mathcal{L} = (\mathcal{C}, \rho, \{\mathcal{R}_k\}_{k \in \mathbb{N}}, \lambda)$ is (T, S) -PAC learnable if there is a PAC learning algorithm for \mathcal{L} that runs in time $\mathcal{O}(T(n, s, 1/\varepsilon, 1/\delta, k, \ell))$ and queries the oracle at most $\mathcal{O}(S(n, s, 1/\varepsilon, 1/\delta, k, \ell))$ times.

We denote the set of parameterized learning problems that are (T, S) -PAC learnable by $\text{PAC}[T, S]$. This is extended to sets of functions \mathbf{S}, \mathbf{T} through setting $\text{PAC}[T, S] = \bigcup_{S \in \mathbf{S}, T \in \mathbf{T}} \text{PAC}[T, S]$.

► **Definition 9.** *Define the complexity classes as follows:*

$$\text{FPT-PAC}_{\text{time}} = \text{PAC}[\text{fpt}, \text{poly}],$$

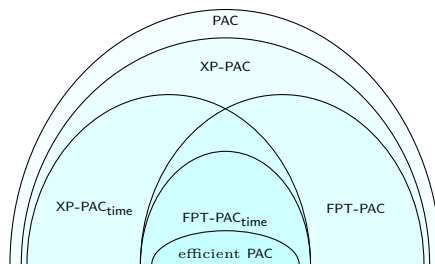
$$\text{FPT-PAC} = \text{PAC}[\text{fpt}, \text{fpt}],$$

$$\text{XP-PAC}_{\text{time}} = \text{PAC}[\text{xp}, \text{poly}],$$

$$\text{XP-PAC} = \text{PAC}[\text{xp}, \text{xp}],$$

where we fix

$$\text{poly} = \text{poly}(n, s, 1/\varepsilon, 1/\delta, k, \ell), \text{fpt} = \text{fpt}(n, s, 1/\varepsilon, 1/\delta; k, \ell), \text{xp} = \text{xp}(n, s, 1/\varepsilon, 1/\delta; k, \ell).$$



■ **Figure 1** A schematic view of the parameterized learning classes defined in Definition 9.

There are examples of natural problems falling into each of these classes [6]. In addition to the above, there is a fifth class that may be considered here: $\text{PAC}[\text{xp}, \text{fpt}]$. However, we are not aware of any natural problems residing there that are not given by the “lower” classes.

Figure 1 provides an overview of these complexity classes and their relationships.

2.4 Consistency Checking for PAC-Learning

We now recall the results tying the complexity of (parameterized) PAC-learning to (parameterized) consistency checking. We have already shown that a consistency checking problem can be transformed into a learning problem, by viewing the hidden solution as the representation of the hidden concept; the same operation can also be done the other way around. Moreover, this transformation can be performed while respecting the parameters. Let $\mathcal{P}_{\text{cons}}$ be a consistency checking problem, and let $\mathcal{P}_{\text{learn}}$ be the respective learning problem. Consider a parameterized version $(\mathcal{P}_{\text{cons}}, \kappa + \lambda)$ of $\mathcal{P}_{\text{cons}}$, where κ maps solutions $S \in \{0, 1\}^*$ to natural numbers, and λ maps lists of labeled instances $((I_1, \lambda_1), \dots, (I_t, \lambda_t))$, $I_i \in \{0, 1\}^*$, $\lambda_i \in \{0, 1\}$, to natural numbers. The parameterized learning problem is then $(\mathcal{P}_{\text{learn}}, \kappa, \lambda')$, where κ is given by the same function as the parameterization of representations, as representations of concepts are exactly the solutions in the original search problem, and $\lambda'(\mathcal{D})$ for a distribution \mathcal{D} is the maximum value of $\lambda(\mathcal{L})$, where \mathcal{L} is any set of labeled instances produced by sampling from \mathcal{D} .

It is well-known that, under the assumption that the hypothesis space is not too large, there is an equivalence between a learning problem being PAC-learnable and the corresponding consistency checking problem being solvable in randomized polynomial time [17]. Brand, Ganian and Simonov proved a generalization of this equivalence in the parameterized sense [6], which we recall next

► **Theorem 10** (Corollary of Theorem 3.17 [6]). *Let $\mathcal{P}_{\text{cons}}$ be a parameterized consistency checking problem, and $\mathcal{P}_{\text{learn}} = (\mathcal{C}, \rho, \kappa, \lambda)$ be its matching parameterized learning problem, where λ depends only on the support of the distribution.*

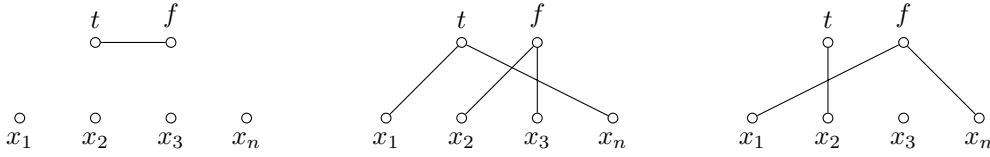
If $\mathcal{P}_{\text{learn}}$ is in FPT-PAC, then $\mathcal{P}_{\text{cons}}$ is in FPT.

Similarly, if $\mathcal{P}_{\text{learn}}$ is in XP-PAC, then $\mathcal{P}_{\text{cons}}$ is in XP.

► **Theorem 11** (Corollary of Theorem 3.19 [6]). *Let $\mathcal{P}_{\text{cons}}$ be a parameterized consistency checking problem, and $\mathcal{P}_{\text{learn}} = (\mathcal{C}, \rho, \kappa, \lambda)$ be its matching parameterized learning problem. Denote the set of representations of concepts in $C \in \mathcal{C}$ of arity n with $\kappa(C) = k$ by $\mathcal{H}_{n,k}$.*

If $\mathcal{P}_{\text{cons}}$ is in FPT and $\log |\mathcal{H}_{n,k}| \in \text{fpt}(n; k)$, then \mathcal{L} is in FPT-PAC_{time}.

Similarly, if $\mathcal{P}_{\text{cons}}$ is in XP and $\log |\mathcal{H}_{n,k}| \in \text{xp}(n; k)$, then \mathcal{L} is in XP-PAC_{time}.



■ **Figure 2** For the SAT instance $\varphi = (x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_n) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_n})$, $n = 4$ the correspondent CONSCHECK: 2-COLORING instance $\mathcal{I} = \{V, \{(E_+, 1), (E_{C_1}, 0), (E_{C_2}, 0)\}\}$, $V = \{t, f, x_1, x_2, x_3, x_n\}$.

The theorems above allow us to automatically transfer parameterized algorithmic upper and lower bounds for the consistency checking into upper and lower bounds for parameterized learning problems, respectively. If a parameterized consistency checking problem is efficiently solvable by a parameterized algorithm, by Theorem 11 we get that the parameterized learning problem is efficiently solvable. Note that in the problems considered in this paper the solution is always a set of vertices/edges, or a partition into such sets, thus $\log |\mathcal{H}_{n,k}|$ is always polynomial.

On the other hand, Theorem 11 tells us that an efficient algorithm for a parameterized learning problem implies an efficient algorithm for the corresponding parameterized consistency checking problem. Turning this around, we see that lower bounds on consistency checking imply lower bounds for learning. That is, if $\mathcal{P}_{\text{cons}}$ is W[1]-hard, then $\mathcal{P}_{\text{learn}}$ is not in FPT-PAC_{time} unless FPT = W[1].

3 Partitioning Problems: 2-Coloring and Split Graphs

We begin our investigation by using two basic vertex bipartition problems on graphs to showcase some of the unexpected complexity-theoretic behavior of consistency checking problems. Let us first consider 2-COLORING, i.e., the problem of partitioning the vertex set into two independent sets. There are two natural ways one can formalize 2-COLORING as a search problem: either one asks for a vertex set X such that both X and the set of vertices outside of X are independent (i.e., they form a proper 2-coloring), or one asks for two independent sets X, Y which form a bipartition of the vertex set. Here, we consider the former variant since it has a slightly smaller hypothesis space⁵.

CONSCHECK: 2-COLORING
Input: $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ where for each $i \in [t]$, $G_i = (V, E_i)$ is a graph and $\lambda_i \in \{0, 1\}$.
Output: A set $X \subseteq V$ such that for each $i \in [t]$, $(X, V \setminus X)$ forms a proper 2-coloring of G_i if and only if $\lambda_i = 1$.

As our first result, we show that CONSCHECK: 2-COLORING is NP-hard.

► **Theorem 12.** *There is no polynomial-time algorithm that solves CONSCHECK: 2-COLORING unless P = NP.*

Proof. We present a reduction that takes an n -variable instance φ of the SATISFIABILITY problem (SAT) and constructs an instance \mathcal{I} of CONSCHECK: 2-COLORING which admits a solution if and only if φ is satisfiable. Let \mathcal{C} denote the set of clauses of φ .

⁵ In general, the precise definition of the sought-after object can be of great importance in the context of consistency checking; this is related to the well-known fact that the selection of a hypothesis space can have a fundamental impact on PAC learnability. However, in our case the proofs provided in this section can also be used to obtain the same results for the latter variant.

Construction. First, we set the vertex set V in \mathcal{I} to be $\{f, t, x_1, x_2, \dots, x_n\}$. For each clause $C \in \mathcal{C}$, we construct an edge set E_C as follows. For each $i \in [n]$, if a true (false) assignment of x_i satisfies C , then we add the edge tx_i (fx_i) to E_C . For each such edge set E_C , we set $\lambda_C = 0$. Finally, we add to \mathcal{I} a positive sample $(E_+, 1)$ such that $E_+ = \{tf\}$. An illustration is provided in Figure 2.

Correctness. Suppose, given an instance φ of SAT, that the reduction described above returns $\mathcal{I} = \{V, (E_i, \lambda_i)\}_{i \in \mathcal{C} \cup \{+\}}$ as an instance of CONSCHECK: 2-COLORING.

Assume that φ admits a satisfying assignment $\mathcal{A}: \{x_i\}_{i \in [n]} \rightarrow \{\text{True}, \text{False}\}$. Consider the coloring $\chi: V \rightarrow \{\text{blue}, \text{red}\}$ such that $\chi(t) = \text{red}$, $\chi(f) = \text{blue}$, and for each $i \in [n]$, $\chi(x_i) = \text{red}$ if and only if $\mathcal{A}(x_i) = \text{True}$.

First, the sample $(E_+, 1)$ of \mathcal{I} is consistent with the coloring χ , since its only edge ft was colored properly. Then, for each $C \in \mathcal{C}$, the sample $(E_C, 0)$ must be consistent with χ , i.e., there exists at least one edge in E_C with same colored endpoints. Indeed, there must exist a variable x_i such that $\mathcal{A}(x_i)$ satisfies φ .

Then, by the construction of \mathcal{I} instance, if $x_i = \text{True}$ ($x_i = \text{False}$) satisfies C then $tx_i \in E_C$ ($fx_i \in E_C$) and hence both x_i and t are **red** (both x_i and f are **blue**) under the constructed coloring χ .

For the other direction, suppose that there is a coloring $\chi: V \rightarrow \{\text{blue}, \text{red}\}$ that is consistent with the instance \mathcal{I} of CONSCHECK: 2-COLORING. Then $\chi(t) \neq \chi(f)$ due to the construction of $(E_+, 1) \in \mathcal{I}$; without loss of generality, let $\chi(t) = \text{red}$, $\chi(f) = \text{blue}$. We retrieve a variable assignment \mathcal{A} for φ in the following way. Recall that for each $C \in \mathcal{C}$, the coloring χ is consistent with the sample $(E_C, 0)$. Since the edge ft has a proper coloring, at least one vertex x_i has an edge to either t or f such that both its endpoints are colored the same way. If this edge is $x_i f$ ($x_i t$), then let $\mathcal{A}(x_i) = \text{False}$ ($\mathcal{A}(x_i) = \text{True}$). If this only results in a partial assignment, we extend this to a complete assignment of all variables in φ by assigning the remaining variables arbitrarily.

We conclude by arguing that the resulting assignment \mathcal{A} has to satisfy φ . Let us consider an arbitrary clause $C \in \mathcal{C}$ and an edge in the corresponding edge set E_C with same colored endpoints, w.l.o.g. $x_i f$. Then, by the way we defined the assignment, $\mathcal{A}(x_i) = \text{False}$. But by our construction, the edge $x_i f \in E_C$ only if $x_i = \text{False}$ satisfies the clause C . Thus, the clause C is satisfied by the assignment \mathcal{A} . Following the same argument, each clause $C \in \mathcal{C}$, and accordingly the instance φ , is satisfied. ◀

It is worth noting that the graphs constructed by the reduction underlying Theorem 12 are very simple – in fact, even the graph induced by the union of all edges occurring in the instances of CONSCHECK: 2-COLORING produced by the reduction has a vertex cover number of 2. This essentially rules out tractability via most standard structural graph parameters. A similar observation can also be made for most other consistency checking lower bounds obtained within this article.

As an immediate corollary of Theorem 12, we obtain that the corresponding learning problem is not efficiently PAC-learnable [2]. To provide a concrete example of the formal transition from consistency checking to the corresponding learning problem described in Section 2.2, we state the problem: In 2-COLORING LEARNING, we are given (1) a set V of vertices, a confidence measure δ and an accuracy measure ε , (2) have access to an oracle that can be queried to return labeled samples of the form (E, λ) where E is an edge set over V and $\lambda \in \{0, 1\}$ according to some hidden distribution, and (3) are asked to return a vertex subset $X \subseteq V$, whereas a sample E is evaluated as positive for X if and only if $(X, V \setminus X)$ forms a 2-coloring on (V, E) .

► **Corollary 13.** 2-COLORING LEARNING is not efficiently PAC-learnable unless $P = NP$.

While the intractability of consistency checking for CONSCHECK: 2-COLORING might already be viewed as surprising, let us now consider the related problem of partitioning the vertex set into one independent set and one clique – i.e., the SPLIT GRAPH problem. As a graph search problem, SPLIT GRAPH is well-known to be polynomially tractable [11]. Following the same line of reasoning as for 2-COLORING, we formalize the corresponding search problem below. Let a pair of vertex subsets $(X \subseteq V, Y \subseteq V)$ be a *split* in a graph $G = (V, E)$ if (X, Y) is a bipartition of V such that X is a clique and Y is an independent set.

CONSCHECK: SPLIT GRAPH

Input: $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ where for each $i \in [t]$, $G_i = (V, E_i)$ is a graph and $\lambda_i \in \{0, 1\}$.

Output: A set $X \subseteq V$ such that for each $i \in [t]$, $(X, V \setminus X)$ is a split in G_i if and only if $\lambda_i = 1$.

Unlike CONSCHECK: 2-COLORING, CONSCHECK: SPLIT GRAPH turns out to be tractable.

► **Theorem 14** (★). CONSCHECK: SPLIT GRAPH can be solved in time $\mathcal{O}(|\mathcal{I}|^3)$.

Naturally, one can formalize the learning problem for CONSCHECK: SPLIT GRAPH in an analogous way as was done for 2-COLORING LEARNING. Since the hypothesis bound of Theorem 11 holds here as well, Theorem 14 implies:

► **Corollary 15.** SPLIT GRAPH LEARNING is efficiently PAC-learnable.

Let us now conclude the section by revisiting the polynomial-time intractability of CONSCHECK: 2-COLORING through the lens of parameterized complexity theory. Naturally, there are many parameterizations one may consider in the setting – as an exercise that follows the same exhaustive-branching ideas as those used for VERTEX COVER [6, Lemma 6.1], one could for instance attempt to parameterize by the size of the smaller color class in the sought-after coloring, whereas a fixed-parameter algorithm in this setting (based on exhaustive branching) would yield a FPT-PAC_{time} algorithm for 2-COLORING LEARNING in the corresponding parameterization of the concept. In this article, we instead showcase a less straightforward fixed-parameter algorithm for the problem when parameterized by the number of negative samples on the input (which in turn corresponds to a parameterization of the distribution in the learning setting [6]). It will later turn out that the same parameterization can be used to achieve fixed-parameter tractability for several other consistency checking problems as well, albeit the individual techniques used vary from problem to problem.

Let $t^- = |\{(E_i, \lambda_i)_{i \in [t]} \mid \lambda_i = 0\}|$ be the number of negative samples in an input instance \mathcal{I} .

► **Theorem 16** (★). CONSCHECK: 2-COLORING is fixed-parameter tractable when parameterized by the number t^- of negative samples.

4 Consistency Checking for Selected Edge Search Problems

In this section, we perform a parameterized analysis of consistency checking for three natural and extensively studied edge search problems on graphs: MATCHING, (k -)PATH and EDGE CLIQUE COVER. We formalize the parameterized consistency checking formulations of these three problems below; recall that a set $\mathcal{F} = \{F_1, \dots, F_\ell\}$ is an *edge clique cover* if each F_i , $i \in [\ell]$ is the edge set of a clique in the graph and each edge in the graph is contained in at least one F_i , $i \in [\ell]$ [7, Subsection 2.2.3].

CONSCHECK: MATCHING

Input: $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ where for each $i \in [t]$, $G_i = (V, E_i)$ is a graph and $\lambda_i \in \{0, 1\}$, and an integer k .

Parameter: k .

Output: A set $F \subseteq \binom{V}{2}$ of size k such that for each $i \in [t]$, F forms a matching in G_i if and only if $\lambda_i = 1$.

CONSCHECK: (k -)PATH

Input: $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ where for each $i \in [t]$, $G_i = (V, E_i)$ is a graph and $\lambda_i \in \{0, 1\}$, and an integer k .

Parameter: k .

Output: A set $F \subseteq \binom{V}{2}$ of size k such that for each $i \in [t]$, F forms a path in G_i if and only if $\lambda_i = 1$.

CONSCHECK: EDGE CLIQUE COVER

Input: $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ where for each $i \in [t]$, $G_i = (V, E_i)$ is a graph and $\lambda_i \in \{0, 1\}$, and an integer k .

Parameter: k .

Output: A set $\mathcal{F} \subseteq 2^{\binom{V}{2}}$ of size k such that for each $i \in [t]$, \mathcal{F} forms an edge clique cover in G_i if and only if $\lambda_i = 1$.

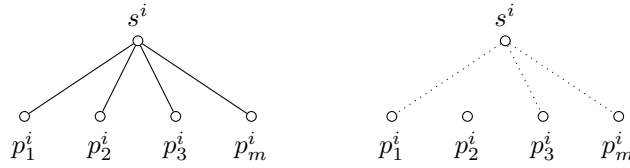
An observant reader may notice that in the first of the three problems above, we consider solution size as a parameter even though the corresponding search problem of finding a maximum matching in a graph is polynomial-time tractable. This is due to the fact that, as it turns out, **MATCHING** in the consistency checking regime is not polynomial-time tractable unless $P = NP$. In fact, we show an even stronger (and more surprising) result:

► **Theorem 17.** **CONSCHECK: MATCHING** *does not admit a fixed-parameter algorithm unless* $FPT = W[2]$.

Proof. We present a reduction that given an instance $(\mathcal{U}, \mathcal{F}, k')$ of the classical **SET COVER** problem [7], constructs an instance \mathcal{I} of **CONSCHECK: MATCHING** which admits a solution if and only if $(\mathcal{U}, \mathcal{F}, k')$ is a yes-instance. An instance $(\mathcal{U}, \mathcal{F}, k')$ of **SET COVER** is a family $\mathcal{F} = \{F_1, \dots, F_m\}$ of m subsets over the n -element universe $\mathcal{U} = \{u_1, \dots, u_n\}$, and we are asked whether there exists a k' -element subset of \mathcal{F} whose union contains all of \mathcal{U} .

Construction. We construct the instance $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ of **CONSCHECK: MATCHING** as follows, with the parameter set to $k = k'$. Let the unique positive sample in \mathcal{I} be the edge set E_1 such that the graph (V, E_1) is a set of k disjoint stars, whereas for each $i \in [k]$ the graph (V, E_i) contains a center s^i adjacent to pendants p_1^i, \dots, p_m^i . Next, for each element $u_j \in \mathcal{U}$, $j \in [n]$, we add a negative sample (V, E_{j+1}) into \mathcal{I} which only contains non-edges between the centers of stars and the leaves (of the same star) corresponding to the sets containing that element; formally, $E_{j+1} = \binom{V}{2} \setminus \{s^i p_\ell^i \mid i \in [k], u_j \in F_\ell\}$. This completes the construction of \mathcal{I} (see also Figure 3).

Correctness. If \mathcal{I} admits a solution Q , then Q must be a matching in (V, E_1) of size k and hence can only contain a single edge from each of the k stars. Hence, $Q = \{s^1 p_{\alpha(1)}, s^2 p_{\alpha(2)}, \dots, s^k p_{\alpha(k)}\}$ for some mapping α . Moreover, since Q is not a matching in (V, E_{j+1}) for any $j \in [n]$, the set $\{F_{\alpha(1)}, \dots, F_{\alpha(k)}\}$ is a set cover for $(\mathcal{U}, \mathcal{F}, k)$. At the same time, given a set cover $\{F_{\beta(1)}, \dots, F_{\beta(k)}\}$ (for some mapping β), we can construct a



■ **Figure 3** Reducing from SET COVER, the CONS CHECK: MATCHING instance has a positive sample with k ($i \in [k]$) stars as shown on the left; for each $u_j \in \mathcal{U}$, the correspondent NO-instance is a complete graph but excluding $\{s^i p_\ell^i \mid i \in [k], u_j \in F_\ell\}$. So, as an example, if $m = 4$ and $u_j \in F_\ell$ for any $\ell \in \{1, 3, m\}$, then for all $i \in [k]$, the dotted edges are out of the construction.

solution for \mathcal{I} by taking $\{s^1 p_{\beta(1)}, \dots, s^k p_{\beta(k)}\}$. This yields a reduction from SET COVER to the problem of deciding the existence of a solution for CONS CHECK: MATCHING; in particular, this means that a fixed-parameter algorithm for CONS CHECK: MATCHING would imply $\text{FPT} = \text{W}[2]$. ◀

A similar reduction also allows us to establish the intractability of consistency checking for PATH.

▶ **Theorem 18** (\star). CONS CHECK: (k -)PATH *does not admit a fixed-parameter algorithm unless* $\text{FPT} = \text{W}[2]$.

However, we show that the third problem under consideration – EDGE CLIQUE COVER – does not become more difficult in the consistency checking regime.

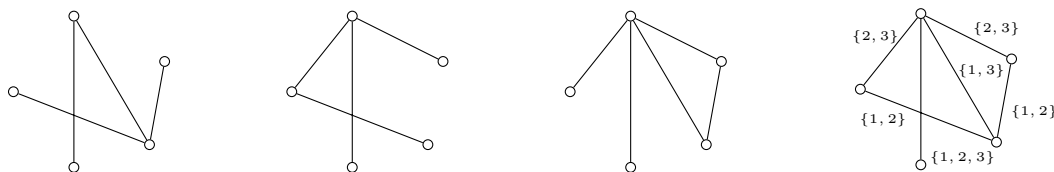
▶ **Theorem 19** (\star). CONS CHECK: EDGE CLIQUE COVER *admits a fixed-parameter algorithm which runs in time* $\mathcal{O}(2^{2^k} \cdot |\mathcal{I}|)$.

Given the fixed-parameter intractability of CONS CHECK: MATCHING and CONS CHECK: (k -)PATH w.r.t. the solution size alone, it is natural to ask whether one could solve these problems at least when the number of negative samples is small, similarly as was done in Theorem 16 for 2-COLORING. We conclude this section by answering this question positively, albeit the algorithmic techniques used here are different from Theorem 16. In fact, it turns out that an adaptation of the classical color-coding technique suffices in this case [7, Subsections 5.2 and 5.6]. For both problems, the task essentially boils down to intersecting all positive samples into one, and then looking for a solution where the set of k edges is not contained in any negative sample. After assuming that all vertices of the solution receive distinct colors, we can perform dynamic programming to find a colorful solution, and while doing so we also store information about which negative samples are already “dealt with”, i.e., which negative samples do not contain the edges in the partial solution.

The proof of Theorem 20 builds on the color-coding algorithm for k -PATH, but otherwise the arguments are fairly similar for both following theorems.

▶ **Theorem 20** (\star). CONS CHECK: MATCHING *admits an algorithm which runs in time* $2^{\mathcal{O}(k+t^-)} \cdot |\mathcal{I}|^{\mathcal{O}(1)}$; *in particular, the problem is fixed-parameter tractable when parameterized by* $k + t^-$.

▶ **Theorem 21** (\star). CONS CHECK: (k -)PATH *admits an algorithm which runs in time* $2^{\mathcal{O}(k+t^-)} \cdot |\mathcal{I}|^{\mathcal{O}(1)}$; *in particular, the problem is fixed-parameter tractable when parameterized by* $k + t^-$.



■ **Figure 4** CONS CHECK: INDEPENDENT SET[degree] instance with G_1, G_2 and G_3 ; and \mathcal{G} for the reformulation of CONS CHECK: INDEPENDENT SET[degree].

5 Consistency Checking for Selected Vertex Search Problems

In the final technical section of this article, we focus our attention on consistency checking for two prominent vertex search problems in parameterized algorithmics: INDEPENDENT SET and DOMINATING SET. As mentioned in the introduction, both problems are believed to be fixed-parameter intractable (the former is $W[1]$ -hard while the latter is $W[2]$ -hard), and so for the purposes of this article we restrict our attention to bounded-degree input graphs – or, more precisely, we consider the maximum degree as an additional parameter⁶. We formalize the consistency checking problems below.

CONS CHECK: INDEPENDENT SET[degree]

Input: Integers k, d , and $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ where for each $i \in [t]$, $G_i = (V, E_i)$ is a graph of degree at most d and $\lambda_i \in \{0, 1\}$.

Parameter: $k + d$.

Output: A set $X \subseteq V$ of size k such that for each $i \in [t]$, X forms an independent set in G_i if and only if $\lambda_i = 1$.

CONS CHECK: DOMINATING SET[degree]

Input: Integers k, d , and $\mathcal{I} = \{V, (E_i, \lambda_i)_{i \in [t]}\}$ where for each $i \in [t]$, $G_i = (V, E_i)$ is a graph of degree at most d and $\lambda_i \in \{0, 1\}$.

Parameter: $k + d$.

Output: A set $X \subseteq V$ of size k such that for each $i \in [t]$, X forms a dominating set in G_i if and only if $\lambda_i = 1$.

Once again, the complexity-theoretic properties of these problems turn out to be very different from those of their simpler graph search analogues. In particular, consistency checking for INDEPENDENT SET is fundamentally harder than for the other two problems.

► **Theorem 22** (★). *There is no fixed-parameter algorithm for CONS CHECK: INDEPENDENT SET[degree] unless $\text{FPT} = W[2]$.*

► **Theorem 23** (★). *CONS CHECK: DOMINATING SET[degree] can be solved by a fixed-parameter algorithm running in time $\mathcal{O}(2^{kd}) \cdot |\mathcal{I}|$.*

Similarly to Theorems 16, 20 and 20, we turn our attention to whether the lower bound for CONS CHECK: INDEPENDENT SET[degree] can be overcome if the number of negative samples is bounded by the parameter. While the $W[2]$ -hardness reduction of Theorem 22

⁶ We remark that all of the obtained results and proofs carry over also to the case where the maximum degree is considered to be an arbitrary fixed constant

does not hold if we are given a bound on the number of samples, it turns out that – unlike for 2-COLORING, MATCHING and (k) -PATH – consistency checking for INDEPENDENT SET remains fixed-parameter intractable even under this additional restriction.

► **Theorem 24** (\star). *There is no fixed-parameter algorithm for CONSCHECK: INDEPENDENT SET[degree] even when the number t^- of negative samples is assumed to be an additional parameter, unless $\text{FPT} = \text{W}[1]$.*

While restricting the number of negative samples alone is insufficient to achieve tractability, we conclude by showing that restricting the total number of samples allows for a fixed-parameter algorithm that solves the problem via a combination of multi-step exhaustive branching and color coding.

► **Theorem 25** (\star). *CONSCHECK: INDEPENDENT SET[degree] admits an algorithm which runs in time $(kdt)^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$; in particular, it is fixed-parameter tractable when parameterized by $k + d + t$.*

6 Concluding Remarks

This article can be seen as a “brief expedition into the forgotten island of consistency checking” – a place where SPLIT GRAPH and EDGE CLIQUE COVER are tractable but 2-COLORING and MATCHING are not, and where on bounded-degree graphs INDEPENDENT SET is $\text{W}[2]$ -hard while DOMINATING SET admits a fixed-parameter algorithm.

To conclude on a more serious note, we remark that our understanding of parameterized consistency checking – and, more broadly, of sample complexity – is still in its infancy. Even in the setting of PAC learning considered here, we so far know very little about which learning problems belong to the classes FPT-PAC and XP-PAC. Still, we hope that the results and techniques presented in this article can contribute to bridging the gap between the parameterized (time) complexity and the sample complexity research fields. A natural target for future work in this direction would be to further deepen our understanding of problems such as learning CNF and DNF formulas [17, 2, 6] or juntas [15].

References

- 1 Sushant Agarwal, Nivasini Ananthkrishnan, Shai Ben-David, Tosca Lechner, and Ruth Uerner. Open problem: Are all VC-classes CPAC learnable? In Mikhail Belkin and Samory Kpotufe, editors, *Conference on Learning Theory, COLT 2021, 15-19 August 2021, Boulder, Colorado, USA*, volume 134 of *Proceedings of Machine Learning Research*, pages 4636–4641. PMLR, 2021. URL: <http://proceedings.mlr.press/v134/open-problem-agarwal21b.html>.
- 2 Michael Alekhnovich, Mark Braverman, Vitaly Feldman, Adam R. Klivans, and Toniann Pitassi. The complexity of properly learning simple concept classes. *J. Comput. Syst. Sci.*, 74(1):16–34, 2008. doi:10.1016/j.jcss.2007.04.011.
- 3 Vikraman Arvind, Johannes Köbler, and Wolfgang Lindner. Parameterized learnability of juntas. *Theor. Comput. Sci.*, 410(47-49):4928–4936, 2009. doi:10.1016/j.tcs.2009.07.003.
- 4 Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM*, 36(4):929–965, 1989. doi:10.1145/76359.76371.
- 5 Olivier Bousquet, Steve Hanneke, Shay Moran, and Nikita Zhivotovskiy. Proper learning, helly number, and an optimal SVM bound. In Jacob D. Abernethy and Shivani Agarwal, editors, *Conference on Learning Theory, COLT 2020, 9-12 July 2020, Virtual Event [Graz, Austria]*, volume 125 of *Proceedings of Machine Learning Research*, pages 582–609. PMLR, 2020. URL: <http://proceedings.mlr.press/v125/bousquet20a.html>.

- 6 Cornelius Brand, Robert Ganian, and Kirill Simonov. A parameterized theory of PAC learning. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023*. AAAI Press, 2023. to appear. URL: <https://arxiv.org/abs/2304.14058>.
- 7 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshantov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 8 Peter Damaschke and Azam Sheikh Muhammad. Competitive group testing and learning hidden vertex covers with minimum adaptivity. *Discret. Math. Algorithms Appl.*, 2(3):291–312, 2010. doi:10.1142/S179383091000067X.
- 9 Reinhard Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, Berlin, Heidelberg, 5th edition, 2017. doi:10.1007/978-3-662-53622-3.
- 10 Eduard Eiben, Sebastian Ordyniak, Giacomo Paesani, and Stefan Szeider. Learning small decision trees with large domain. In *The 32nd International Joint Conference on Artificial Intelligence (IJCAI-23), August 19–25, 2023, Macao, S.A.R.* International Joint Conferences on Artificial Intelligence Organization, 2023. to appear.
- 11 Peter L. Hammer and Bruno Simeone. The splittance of a graph. *Comb.*, 1(3):275–284, 1981. doi:10.1007/BF02579333.
- 12 Steve Hanneke. The optimal sample complexity of PAC learning. *J. Mach. Learn. Res.*, 17:38:1–38:15, 2016. URL: <http://jmlr.org/papers/v17/15-389.html>.
- 13 M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.
- 14 Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. Adaptive computation and machine learning. MIT Press, 2012. URL: <http://mitpress.mit.edu/books/foundations-machine-learning-0>.
- 15 Elchanan Mossel, Ryan O’Donnell, and Rocco A. Servedio. Learning juntas. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9–11, 2003, San Diego, CA, USA*, pages 206–212. ACM, 2003. doi:10.1145/780542.780574.
- 16 Sebastian Ordyniak and Stefan Szeider. Parameterized complexity of small decision tree learning. In *Proceeding of AAAI-21, the Thirty-Fifth AAAI Conference on Artificial Intelligence*, pages 6454–6462. AAAI Press, 2021. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/16800>.
- 17 Leonard Pitt and Leslie G. Valiant. Computational limitations on learning from examples. *J. ACM*, 35(4):965–984, 1988. doi:10.1145/48014.63140.
- 18 Oleg Sheyner and Jeannette M. Wing. Tools for generating and analyzing attack graphs. In *FMCO*, volume 3188 of *Lecture Notes in Computer Science*, pages 344–372. Springer, 2003. URL: <https://www.cs.cmu.edu/~scenariograph/sheynerwing04.pdf>.
- 19 L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. doi:10.1145/1968.1972.
- 20 Steffen van Bergerem, Martin Grohe, and Martin Ritzert. On the parameterized complexity of learning first-order logic. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS ’22*, pages 337–346, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3517804.3524151.
- 21 Jeannette M. Wing. Attack graph generation and analysis. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1128817.1128822.


Finding Degree-Constrained Acyclic Orientations

Jaroslav Garvardt  

Philipps-Universität Marburg, Germany
Friedrich-Schiller-Universität Jena, Germany

Malte Renken  

Technische Universität Berlin, Germany

Jannik Schestag  

Philipps-Universität Marburg, Germany
Technische Universiteit Delft, The Netherlands
Friedrich-Schiller-Universität Jena, Germany

Mathias Weller  

Technische Universität Berlin, Germany

Abstract

We consider the problem of orienting a given, undirected graph into a (directed) acyclic graph such that the in-degree of each vertex v is in a prescribed list $\lambda(v)$. Variants of this problem have been studied for a long time and with various applications, but mostly without the requirement for acyclicity. Without this requirement, the problem is closely related to the classical GENERAL FACTOR problem, which is known to be NP-hard in general, but polynomial-time solvable if no list $\lambda(v)$ contains large “gaps” [Cornuéjols, J. Comb. Theory B, 1988]. In contrast, we show that deciding if an *acyclic* orientation exists is NP-hard even in the absence of such “gaps”.

On the positive side, we design parameterized algorithms for various, natural parameterizations of the acyclic orientation problem. A special case of the orientation problem with degree constraints recently came up in the context of reconstructing evolutionary histories (that is, phylogenetic networks). This phylogenetic setting imposes additional structure onto the problem that can be exploited algorithmically, allowing us to show fixed-parameter tractability when parameterized by either the treewidth of G (a smaller parameter than the frequently employed “level”), by the number of vertices v for which $|\lambda(v)| \geq 2$, by the number of vertices v for which the highest value in $\lambda(v)$ is at least 2. While the latter result can be extended to the general degree-constraint acyclic orientation problem, we show that the former cannot unless $\text{FPT}=\text{W}[1]$.

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

Keywords and phrases Graph Orientation, Phylogenetic Networks, General Factor, NP-hardness, Parameterized Algorithms, Treewidth

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.19

Funding *Jaroslav Garvardt*: Partially supported by the Carl Zeiss Foundation within the project “Interactive Inference”.

Jannik Schestag: Supported by the German Academic Exchange Service (DAAD), project 57556279.

Acknowledgements This work was initiated on the research retreat of the Algorithmics and Computational Complexity group of TU Berlin, held in Darlingerode in September 2022. We thank André Nichterlein and Till Fluschnik for helpful discussions and ideas and our anonymous reviewers for pointing out many small errors in previous versions of the paper.



© Jaroslav Garvardt, Malte Renken, Jannik Schestag, and Mathias Weller;
licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 19; pp. 19:1–19:14

Leibniz International Proceedings in Informatics



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

1 Introduction

The de-facto standard approach to reconstructing phylogenetic trees from genomic data employs heuristic local-search in the space of all rooted evolutionary trees [5]. The success of this method is highly impacted by the choice of neighborhood (as one would expect from a local-search approach), and existing research in this direction is vast [10, 17, 18, 2]. Recently, efforts have been made to use this technique to reconstruct phylogenetic *networks*, thereby supporting reticulate/hybridizing evolution. As a possible neighborhood of a rooted network N , Molloy et al. [15] proposed to consider all reorientations of (subnetworks of) N . The question whether a given undirected graph has a valid orientation into a phylogenetic network and the enumeration of such networks are fundamental computational problems in this context. We model these problems as the search for acyclic orientations of a given undirected graph, such that the in-degree of each vertex v is in a prescribed list $\lambda(v)$.

The problem of orienting a given undirected graph G , subject to certain degree-related constraints, has been long studied with various applications [8, 6, 7], but mostly without the requirement for acyclicity. Without this requirement, the problem is closely related to the classical GENERAL FACTOR problem, introduced by Lovász [12, 13], which asks for an (undirected) subgraph of G satisfying the given degree constraints. Cornuéjols [4] showed that GENERAL FACTOR is polynomial-time solvable if the maximum gap size¹ is at most one, and NP-hard otherwise, even if the maximum degree in the input graph is three. Both of these results can be easily transferred to the problem of finding (possibly cyclic) degree-constrained orientations (to reduce GENERAL FACTORS to the orientation problem, subdivide each edge of the input graph G with a vertex v with $\lambda(v) = \{0, 2\}$; in the other direction, subdivide each edge with a vertex v with $\lambda(v) = \{1\}$, see Theorem 5). Deciding whether there is an *acyclic* orientation constraint by specific lower bounds $f(v)$ for the in-degree of each vertex v (that is, $\lambda(v) = \{f(v), f(v) + 1, \dots\}$) has been shown to be NP-hard, even if $f(v) \in \{0, 1\}$ for all but one vertex [11]. A generalization of the problem with weighted edges and allowing edge *and* vertex deletions to satisfy prescribed weight-constraints has been considered by Mathieson and Szeider [14], showing parameterized results with respect to the number of deleted elements and the maximum integer in any of the given lists.

The phylogenetic setting, which is the main motivation for our work, imposes structure on the allowed in-degree lists. Indeed, phylogenetic networks (which, for the purpose of this work, are rooted directed acyclic graphs) are made up of four types of vertices: (1) the (unique) root, with in-degree zero, (2) leaves with out-degree zero and in-degree one, (3) tree-nodes with in-degree one, and (4) reticulations with out-degree one. Thus, if the input network is orientable into a phylogenetic network, all vertices v but one must satisfy $\lambda(v) \subseteq \{1, \deg(v) - 1\}$. Most commonly, phylogenetic networks are “binary”, that is, all nodes of type (3) and (4) have degree (that is, in-degree plus out-degree) three, in which case $\lambda(v) \subseteq \{1, 2\}$ and no vertex can be of type (3) and (4) at the same time. While these lists do not have gaps, we cannot use GENERAL FACTOR to solve this case, since phylogenetic networks must be acyclic. Bulteau et al. [3] showed a polynomial-time algorithm that computes an orientation of a given undirected graph of maximum degree 3 into a (binary) network and proved NP-hardness if the input graph has degree ≥ 5 , leaving open the degree-4 case. If the in-degree of every vertex v is fixed, that is, $|\lambda(v)| = 1$ for all vertices v , an algorithm of Huber et al. [9] can produce an orientation in linear time. The authors also considered the problem of orienting a

¹ The *gap size* is the maximum size of any “gap” appearing in $\lambda(v)$ for any vertex v , that is, $\max_{v,i} \{\lambda_{i+1}^v - \lambda_i^v - 1\}$ when taking $\lambda(v) = \{\lambda_1^v < \lambda_2^v < \dots\}$.

given undirected graph in such a way, that the resulting network falls in one of various classes of binary phylogenetic networks. They showed fixed-parameter tractability with respect to the “level”² of the input graph.

Our contributions

We analyze the parameterized complexity of the aforementioned degree-constraint orientation problems. When cyclic orientations are permitted, the polynomial-time algorithm for GENERAL FACTOR [4] can be applied; this approach can be extended to show fixed-parameter tractability for the parameter “combined number of gaps of size ≥ 2 in all degree lists”. Interestingly, such a result is unlikely for the case of acyclic orientations. While this follows already from the known NP-hardness proof for acyclic orientations [11], we strengthen the result to maintain NP-hardness for gapless inputs even for the phylogenetic version on input graphs of maximum degree three.

Since phylogenetic networks can be expected to be tree-like, algorithms for graphs of low treewidth are attractive for this application. We show that the phylogenetic version of the orientation problem can be solved in $O(8^{\text{tw}} \cdot \text{tw}! \cdot \text{tw}^2 \cdot n)$ time on n -vertex graphs of treewidth tw . For the cyclic and acyclic orientation problems with unrestricted λ , this approach yields running times of $O(q^{2\text{tw}} \cdot \text{tw}^2 \cdot n)$ and $O(q^{2\text{tw}} \cdot \text{tw}! \cdot \text{tw}^2 \cdot n)$ time, where q is the maximum allowed in-degree. We justify adding the second parameter q by proving that the problems with unrestricted λ are W[1]-hard with respect to the treewidth alone.

A general observation for phylogenetic networks is that the number of reticulation events is small compared to the overall size of the phylogeny. This means that we expect to see only few vertices v whose list $\lambda(v)$ contains numbers greater than one. We call these vertices “potential reticulations” and show that a corresponding acyclic orientation of an n -vertex graph with r potential reticulations can be computed in $O(2^r \cdot r \cdot n^2 + n^2 \cdot q)$ time.

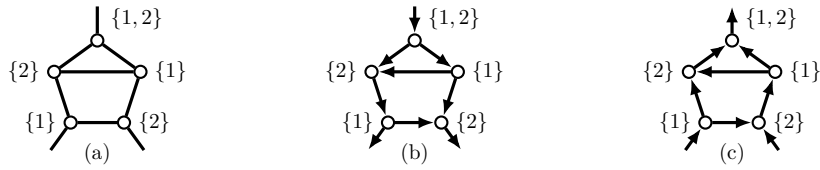
Motivated by the observation that the acyclic orientation problem is linear-time solvable if every degree list $\lambda(v)$ contains just a single entry [9], we say that a vertex v is *hazy* if $|\lambda(v)| > 1$ and develop an algorithm computing a corresponding acyclic orientation of a graph with m edges and h hazy vertices in $O(2^h \cdot h \cdot m)$ time.

2 Preliminaries

We use the notations $[a, b] := \{a, a + 1, \dots, b\}$, $[n] := [1, n]$, and $[n]_0 := [0, n]$. By default, graphs are assumed to be simple and undirected. For a vertex or edge x of a graph $G = (V(G), E(G))$, $G - x$ denotes the graph obtained from G by deleting said vertex or edge. The *induced subgraph* $G[V']$ is obtained from G by deleting all vertices in $V(G) \setminus V'$. An *orientation* of G is a directed graph G^\rightarrow with the same vertex set $V(G^\rightarrow) = V(G)$, such that its arc set $A(G^\rightarrow)$ includes for every edge $\{v, w\} \in E(G)$ exactly one of the arcs (v, w) , (w, v) . A directed graph G^\rightarrow is *acyclic* if it contains no directed cycle. A total order σ of the vertices of G^\rightarrow is a *topological order* if $(v, w) \in A(G^\rightarrow)$ implies $v <_\sigma w$ for all vertices v, w ; in this case we write $\sigma \in \text{Top}(G^\rightarrow)$. Every total order σ of $V(G)$ induces an acyclic orientation of G for which σ is a topological order. The notation $A <_\sigma B$ means that $a <_\sigma b$ for all $a \in A, b \in B$.

² The level of an undirected graph is the maximum over all biconnected components of the size of a smallest feedback edge set in that biconnected component.

19:4 Finding Degree-Constrained Acyclic Orientations



■ **Figure 1** (a) shows the “copy gadget” used in the proof of Lemma 3. The vertices are annotated with their degree lists λ . (b) and (c) show the only two acyclic λ -abiding orientations of (a).

We denote the set of neighbors of v in an (undirected) graph G by $N_G(v)$, the degree by $\deg_G(v) := |N_G(v)|$, the in-degree in G^\rightarrow by $\deg_{G^\rightarrow}^-(v)$ and the out-degree by $\deg_{G^\rightarrow}^+(v)$. A vertex v is a *source* in a directed graph G^\rightarrow if $\deg_{G^\rightarrow}^-(v) = 0$. If clear from context, the subscripts may be omitted. The DEGREE-CONSTRAINED ORIENTATION problem is defined as follows.

DEGREE-CONSTRAINED ORIENTATION (DCO)

Input: A graph $G = (V, E)$, and a function $\lambda : V \rightarrow 2^{\mathbb{N}}$.

Question: Is there an orientation of G such that $\deg^-(v) \in \lambda(v)$ for each $v \in V$?

A feasible solution of an instance of DCO is called λ -*abiding orientation*. The variant of DCO where only acyclic orientations are allowed is denoted DEGREE-CONSTRAINED ACYCLIC ORIENTATION (DCAO). When writing each set $\lambda(v)$ as $\lambda(v) =: \{\lambda_1^v < \lambda_2^v < \dots\}$, we say that $\lambda(v)$ has a k -*gap* at position i if $\lambda_{i+1}^v - \lambda_i^v > k$ for $k \geq 1$. Finally, we consider a special case of the DCAO problem that arises in phylogenetics: In PHYLOGENETIC DEGREE-CONSTRAINED ORIENTATION (PDCO), we have

- a unique *root* vertex r with $\lambda(r) = \{0\}$,
- a non-empty $\lambda(v) \subseteq \{1, \deg_G(v) - 1\}$ for each vertex $v \neq r$ with $\deg_G(v) > 1$, and
- $\lambda(v) = \{1\}$ for each vertex $v \neq r$ with $\deg_G(v) = 1$ (called *leaves*).

If the input graph is disconnected, we can solve the problem on each connected component individually. Feasible λ -abiding solutions of the connected components can easily be combined to a solution for the entire graph. Therefore, throughout the paper we assume that the input graph G is connected.

3 NP-Hardness of PDCO

Recall that we can decide in polynomial time whether a graph of maximum degree three can be oriented into a phylogenetic network [3]. Essentially, this requires that all vertices v (except the root and the leaves) have $\lambda(v) = \{1, \deg(v) - 1\}$. We show that, if $\lambda(v)$ is only required to be a *subset* of $\{1, \deg(v) - 1\}$, the problem becomes NP-hard. In particular, we reduce the NP-hard [16] MONOTONE EXACT 1 IN 3 SAT (MX3SAT) to PDCO. In this problem, the input is a CNF-formula Φ without negations where each clause contains at most three literals and the question is whether there is an assignment of the variables such that in each clause exactly one variable is assigned true.

Our reduction makes use of a “copy gadget” allowing us to multiply the information whether a variable is set to true or false. Each such gadget consists of five degree-3 vertices as shown in Figure 1(a). We refer to the three edges leaving any such gadget as its *top edge* and its two *bottom edges*, where the top edge is the one attached to the vertex v with $\lambda(v) = \{1, 2\}$. It is not difficult to verify that the only two possible acyclic orientations of the copy gadget are the ones shown in Figure 1(b) and (c). This implies the following observation.

► **Observation 1.** *Let (G, λ) be an input for the DCO problem such that G contains a copy of the gadget as an induced subgraph. Let G^\rightarrow be a λ -abiding acyclic orientation of G . Then, all bottom edges are oriented towards the gadget in G^\rightarrow if and only if the top edge is oriented away from the gadget in G^\rightarrow .*

► **Construction 2.** *Let Φ be an instance of MONOTONE EXACT 1 IN 3 SAT with n variables x_1, x_2, \dots, x_n and m clauses C_1, C_2, \dots, C_m . For each variable x_i , let $r_i \in [m]$ be the number of occurrences of x_i in Φ , let $\rho_i : [r_i] \rightarrow [m]$ be an arbitrary total order of the indices of clauses that contain x_i and let $\mathcal{X}_i := \{C_{\rho_i(1)}, \dots, C_{\rho_i(r_i)}\}$ be the set of clauses that contain x_i .*

We construct from Φ an instance (G, λ) of PDCO as follows. For each variable x_i , we add a vertex v_i with $\lambda(v_i) = \{1, 2\}$ and, for each clause C_j , we add a vertex w_j with $\lambda(w_j) = \{1\}$. For each variable x_i we add $r_i - 1$ many copy gadgets to G in the following way: The first copy gadget of x_i connects with its top edge to the vertex v_i . The top edge of any subsequent copy gadget of x_i is identified with the right bottom edge of the previous copy gadget. The left bottom edge of the j -th copy gadget of x_i attaches to the clause vertex $w_{\rho_i(j)}$ belonging to the j -th clause $C_{\rho_i(j)}$ in \mathcal{X}_i . In this way, we create a chain of $r_i - 1$ many copy gadgets, where the last one connects with its two bottom edges to the clause vertices $w_{\rho_i(r_i-1)}$ and $w_{\rho_i(r_i)}$ of the last two clauses in \mathcal{X}_i . In the corner case that $r_i = 1$, we attach v_i directly to $w_{\rho_i(1)}$.

To ensure that there is a unique root, we add a vertex u_0 with $\lambda(u_0) = \{0\}$ and vertices u_i with $\lambda(u_i) = \{1\}$ for all $i \in [n]$ and we add the edges $\{u_i, u_{i+1}\}$ for all $0 \leq i \leq n - 1$, as well as $\{u_i, v_i\}$ for all $i \in [n]$. Finally, for each degree-2 vertex z in the construction, add a private neighbor ℓ_z with $\lambda(\ell_z) = \{1\}$.

Note that the definition of λ for the variable vertices simulates the two possible states of the variable and, for the clause vertices, it forces exactly one variable in the clause to be true. Further, note that the vertex u_0 and all ℓ_z have degree one and all variable vertices v_i , all vertices u_i and all vertices of a copy gadget have degree three. Note also that each clause C_j in Φ contains exactly three variables and, thus, the corresponding clause vertex w_j is connected to exactly three copy gadgets. Hence, every vertex in G has degree at most three. Finally, note that λ has no gaps of any size.

► **Lemma 3.** *Let Φ be an instance of MX3SAT and let (G, λ) be the instance of PDCO constructed by Construction 2 for Φ . Then, Φ is satisfiable if and only if G has a λ -abiding acyclic orientation.*

Proof. “ \Rightarrow ”: Suppose that Φ has a satisfying assignment $\beta : \{x_1, x_2, \dots, x_n\} \rightarrow \{\text{true}, \text{false}\}$. We can construct a λ -abiding acyclic orientation G_β^\rightarrow for (G, λ) as follows:

- The edge $\{u_0, u_1\}$ is oriented towards u_1 .
- For each $i \in [n]$, the edge $\{z, \ell_z\}$ for each ℓ_z is oriented towards ℓ_z and the edges $\{u_i, v_i\}$ and $\{u_i, u_{i+1}\}$ are directed away from u_i .
- For each variable x_i with $\beta(x_i) = \text{false}$, the edge between v_i and the first copy gadget of x_i is oriented towards v_i . Further, each copy gadget of x_i is oriented as shown in Figure 1(c), implying that the edge between any such copy gadgets and any clause vertex w_j of a clause C_j containing x_i is oriented away from w_j .
- Analogously, for each variable x_i with $\beta(x_i) = \text{true}$, the edge between v_i and the first copy gadget of x_i is oriented away from v_i . Further, each copy gadget of x_i is oriented as shown in Figure 1(b), implying that the edge between any such copy gadgets and any clause vertex w_j of a clause C_j containing x_i is oriented towards w_j .

19:6 Finding Degree-Constrained Acyclic Orientations

Since each clause C_j contains exactly one variable that is assigned true by β , exactly one of the edges incident to w_j is oriented towards w_j in G_β^\rightarrow , fulfilling the given in-degree list of w_j . One can verify that G_β^\rightarrow satisfies all in-degree lists of all vertices in copy-gadgets (see Figure 1) as well as the in-degree lists of all u_i, v_i , and all ℓ_z . Moreover, G_β^\rightarrow is acyclic since (a) each copy gadget is oriented in an acyclic way, (b) no directed cycle can contain both bottom arcs of any copy gadget, and (c) no directed cycle can contain any v_i due to the orientation of the edges $\{u_i, v_i\}$ for all $i \in [n]$. Therefore, G_β^\rightarrow is a λ -abiding acyclic orientation and (G, λ) is a yes-instance.

“ \Leftarrow ”: Suppose that G has a λ -abiding acyclic orientation G^\rightarrow . We define a truth assignment $\beta : \{x_1, x_2, \dots, x_n\} \rightarrow \{\text{true}, \text{false}\}$ as follows for each variable x_i :

$$\beta(x_i) := \begin{cases} \text{false} & \text{if } v_i \text{ has an arc incoming from the first copy gadget of } x_i \text{ in } G^\rightarrow \\ \text{true} & \text{otherwise.} \end{cases}$$

By construction of G , for each variable x_i , all edges between copy gadgets of x_i and clause vertices are oriented the same way as the edge between v_i and the first copy gadget of x_i in G^\rightarrow (the edge incident with v_i is oriented away from v_i if and only if the edges between the copy gadgets of x_i and the clause vertices are oriented away from their copy gadget). Thus, for each variable x_i , we have $\beta(x_i) = \text{true}$ if and only if, for each clause $C_j \in \mathcal{X}_i$, the edge between w_j and the corresponding copy gadget of x_i is oriented towards w_j . Since G^\rightarrow is a solution, each clause vertex w_j has exactly one incoming arc in G^\rightarrow , with the other edges oriented away from w_j . Thus, the assignment β satisfies every clause of Φ exactly once, implying that Φ is a yes-instance. \blacktriangleleft

Since PDCO is a special case of DCAO, the previous reduction implies the following.

► **Corollary 4.** *PDCO and DCAO are NP-hard, even if the maximum degree is 3 and the instance contains no gaps.*

4 Parameterized Algorithms

4.1 Number of gaps

► **Theorem 5.** *DCO can be solved in polynomial time, when the instance does not contain 2-gaps.*

Proof. Let G be any graph and $\lambda : V(G) \rightarrow 2^{\mathbb{N}}$. We obtain G' from G by subdividing every edge e once with a new vertex γ_e of degree 2. Extend λ to $V(G')$ by assigning $\lambda(\gamma_e) = 1$ for every vertex $\gamma_e \in V(G') \setminus V(G)$. By a result of Cornuéjols [4], we can in polynomial time find a subgraph G'' of G' with $\deg_{G''}(v) \in \lambda(v)$ for each $v \in V(G')$.

Note that, for any $e = \{v, w\} \in E(G)$, the subgraph G'' contains exactly one of the edges $\{v, \gamma_e\}, \{w, \gamma_e\}$. We use this to define an orientation of G , by directing $e = \{v, w\}$ towards w if and only if $\{w, \gamma_e\} \in E(G'')$. It is easy to verify that this orientation satisfies $\deg^-(v) = \deg_{G''}(v)$ for every vertex $v \in V(G)$ and is thus a solution for an instance of DCO. \blacktriangleleft

A simple branching algorithm gives us the following result in regards to the total number of 2-gaps appearing in all $\lambda(v)$ combined, which we denote by gaps_2 .

► **Theorem 6.** *DCO can be solved in $2^{\text{gaps}_2} \cdot n^{\mathcal{O}(1)}$ time on n -vertex graphs.*

Proof. Let (G, λ) be any given instance. For any vertex v let k be the number of 2-gaps in $\lambda(v)$. Note that $\lambda(v)$ can be partitioned into $k + 1$ subsets, none of which has any 2-gaps. By restricting $\lambda(v)$ to one of these subsets for every vertex v , we obtain a polynomial-time solvable subproblem according to Theorem 5. As there are at most 2^{gaps_2} such subproblems, the claimed time bound follows. \blacktriangleleft

4.2 Treewidth

► **Theorem 7.** *Let an n -vertex graph G be given together with a tree decomposition of width tw . Let further $\max_\lambda = \max_{v \in V(G)} \max \lambda(v)$ be the maximum admitted in-degree. Then the instance (G, λ) of*

- DCO can be solved in $\mathcal{O}((\max_\lambda)^{2\text{tw}} \cdot \text{tw}^2 \cdot n)$ time.
- DCAO can be solved in $\mathcal{O}((\max_\lambda)^{2\text{tw}} \cdot \text{tw}! \cdot \text{tw}^2 \cdot n)$ time;
- PDCO can be solved in $\mathcal{O}(8^{\text{tw}} \cdot \text{tw}! \cdot \text{tw}^2 \cdot n)$ time;

Theorem 7 follows directly from the following lemma, which we prove by a dynamic programming approach. This proof is deferred to a long version of this paper.

► **Lemma 8.** *Let an n -vertex graph G be given together with a tree decomposition of width tw . Let $d_1, d_2 \in \mathbb{N}$ with $\lambda(v) \subseteq [d_1]_0 \cup [\deg(v) - d_2, \deg(v)]$ for each vertex v . Then the instance (G, λ) of*

- DCO can be solved in $\mathcal{O}((d_1 + d_2 + 2)^{2\text{tw}} \cdot \text{tw}^2 \cdot n)$ time;
- DCAO can be solved in $\mathcal{O}((d_1 + d_2 + 2)^{2\text{tw}} \cdot \text{tw}! \cdot \text{tw}^2 \cdot n)$ time.

4.3 Number of hazy vertices

We say a vertex v is *settled* if $|\lambda(v)| = 1$ and *hazy* otherwise. We denote by h the number of hazy vertices. If every vertex is settled, DCAO (not necessarily connected) can be solved in $\mathcal{O}(m)$ time by repeatedly picking a vertex v with $\lambda(v) = \{0\}$, deleting it from the graph and subtracting 1 from each of its neighbors' desired in-degrees until we reach a trivial instance with a) $\lambda(v) \neq \{0\}$ for each vertex v or b) G only contains a single vertex v with $\lambda(v) = \{0\}$. Consequently, PDCO can be solved in $\mathcal{O}(2^h \cdot m)$ time since every hazy vertex admits only two possible in-degrees.

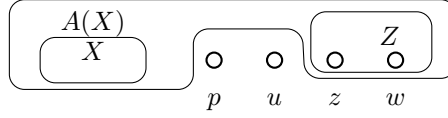
In the following, we want to show that DCAO too is fixed-parameter tractable with respect to h . To this end, let (G, λ) be an instance of DCAO and let H and S be the set of hazy and settled vertices, respectively. For $s \in S$, we will not distinguish between $\lambda(s)$ and its only element for the sake of brevity.

We define the *closure* $A(X)$ of a set $X \subseteq H$ as the smallest superset of X that contains every vertex $v \in S$ with $|N_G(v) \cap A(X)| \geq \lambda(v)$. Observe that the closure can be computed in $\mathcal{O}(n + m)$ time: As long as there remains a vertex $v \in X \cup \{s \in S \mid \lambda(s) = 0\}$, add v to $A(X)$, decrement $\lambda(w)$ for all $w \in N_G(v) \cap S$, and delete v . Note that this also implies that $A(X)$ is uniquely defined.

► **Observation 9.** *Let $X \subseteq H$. The closure $A(X)$ can be computed in $\mathcal{O}(n + m)$ time.*

A subset X of H is called *feasible* if there exists an order σ of the vertices such that $A(X) \prec_\sigma V(G) \setminus A(X)$ and the orientation G^\rightarrow induced by σ satisfies $\deg_{G^\rightarrow}^-(v) \in \lambda(v)$ for all $v \in A(X)$. The order σ is then called a *feasible order* of X . If we know which subsets of H are feasible, then we can solve DCAO as shown by the following lemma.

► **Lemma 10.** *(G, λ) is a yes-instance of DCAO if and only if H is a feasible subset of the hazy vertices H and $A(H) = V(G)$.*



■ **Figure 2** Illustration of the proof of Lemma 11. Vertices are ordered left to right according to σ .

Proof. Let G^\rightarrow be a λ -abiding acyclic orientation of G . Suppose for contradiction that $Q := V(G) \setminus A(H) \neq \emptyset$. By definition of the closure, every vertex of Q must have at least one incoming edge from another vertex in Q . Thus, the induced subgraph $G^\rightarrow[Q]$ does not contain a source, and hence must contain a cycle. This proves $A(H) = V(G)$ and thus also that H is a feasible subset. The reverse implication is immediate. ◀

We compute the feasible subsets by dynamic programming, based on the following lemma.

► **Lemma 11.** *Let $X \cup \{p\}$ be a feasible subset of hazy vertices H and σ a feasible order of $X \cup \{p\}$ with induced orientation G^\rightarrow . If $X <_\sigma \{p\}$, then X has a feasible order $\sigma' \in \text{Top}(G^\rightarrow)$ with $A(X) <_{\sigma'} \{p\} <_{\sigma'} V(G) \setminus (A(X) \cup \{p\})$.*

Proof. Figure 2 illustrates this proof.

Let G^\rightarrow be the orientation induced by σ . Let $z \in A(X)$ be chosen σ -minimal with $p <_\sigma z$. If no such z exists, then we are done since σ is a feasible order of X , then. Since $X <_\sigma \{p\}$, we cannot have $z \in X$. Therefore $z \in S = V(G) \setminus H$ and $|N_G(z) \cap A(X)| \geq \lambda(z)$ holds. Since $V_{<p} := \{v \in V \mid v <_\sigma p\} \subseteq A(X \cup \{p\})$, any vertex $v \in V_{<p}$ must have $\deg_{G^\rightarrow}(v) \in \lambda(v)$. This implies $V_{<p} \subseteq A(X)$.

Now let Z be the connected component of $G[A(X) \setminus V_{<p}]$ containing z . We claim that G^\rightarrow directs no edge from $V \setminus A(X)$ to Z . Hence we may modify σ by moving Z in front of p without affecting the induced orientation G^\rightarrow .

To prove the claim, assume for contradiction that G^\rightarrow contains an arc (u, w) with $p \leq_\sigma u <_\sigma z$ and $w \in Z$. Then define $A' := A(X) \setminus Z_{\geq w}$ where $Z_{\geq w} = \{v \in Z \mid v \geq_\sigma w\}$. Observe that no vertex v in A' has an incoming edge from $Z_{\geq w} \subseteq V \setminus A'$. Hence, $|N_G(v) \cap A'| = |N_G(v) \cap V_{<p}| < \deg_{G^\rightarrow}(v) = \lambda(v)$. This means that A' contradicts the minimality of $A(X)$.

Therefore, Z can be moved in front of p in the topological order. If this is followed by repeating the above steps as long as some valid choice of z exists, we eventually obtain σ' as claimed. ◀

► **Theorem 12.** *DCAO can be solved in $\mathcal{O}(2^h \cdot h \cdot m)$ time on an m -edge graph with h hazy vertices.*

Proof. Let (G, λ) be a given DCAO-instance and let H be the set of hazy vertices. Using the Iverson bracket notation³, we define a dynamic programming table $T[X] := [X \text{ is feasible}]$. By Lemma 10 and Observation 9, the answer to the input instance can be computed in linear time from $T[H]$. Because $A(\emptyset)$ contains no hazy vertices, we can check in linear time whether there is a λ -abiding orientation of $G[A(\emptyset)]$ as outlined in the beginning of this section. This yields $T[\emptyset]$.

It remains to compute $T[X]$ recursively for any nonempty $X \subseteq H$. To this end, we iterate over all $p \in X$. Define $X' := X \setminus \{p\}$ and $\mu := |N_G(p) \cap A(X')|$. If $T[X'] \neq 1$ or $\mu \notin \lambda(p)$ then continue with the next choice of p . Otherwise we temporarily replace $\lambda(p)$ by μ and

³ For a proposition P , $[P]$ is defined to be 1 if P holds and 0 otherwise.

orient all edges between $A(X')$ and $V(G) \setminus A(X')$ away from $A(X')$. We then check in linear time whether this orientation can be extended to a λ -abiding orientation of $G[A(X) \setminus A(X')]$. If this is the case, we set $T[X] = 1$. Otherwise, after trying all choices of p , we set $T[X] = 0$.

To see that this algorithm computes $T[X]$ correctly, suppose first that we end up with $T[X] = 1$. Since $T[X'] = 1$, by induction there is a feasible order σ' of X' . Take also a topological order σ'' of the constructed orientation of $G[A(X) \setminus A(X')]$. Produce a new order σ of $V(G)$ by first taking $A(X')$ in the order given by σ' , followed by $A(X) \setminus A(X')$ in the order given by σ'' , and finally all remaining vertices in an arbitrary order. It is not difficult to check that σ is a feasible order of X .

Now suppose conversely that X has a feasible order σ with induced orientation G^\rightarrow of G . By Lemma 11, there is some choice of p for which X' has a feasible order $\sigma' \in \text{Top}(G^\rightarrow)$ which puts p immediately after $A(X')$. In particular, $\deg_{G^\rightarrow}^-(p) = |N_G(p) \cap A(X')|$ and $T[X'] = 1$ by induction. Hence, the iteration which considers p will produce $T[X] = 1$.

With regards to the running time, there are clearly 2^h entries to compute. For each choice of p we only need $\mathcal{O}(m)$ time. This gives $\mathcal{O}(2^h \cdot h \cdot m)$ time overall. \blacktriangleleft

4.4 Potential Reticulations

Throughout this section, let (G, λ) be an instance of DCAO and let R be the set of *potential reticulations* of G , that is, $R := \{v \in V \mid \max \lambda(v) \geq 2\}$. Consequently, $\lambda(w) \subseteq \{0, 1\}$ for each $w \in V(G) \setminus R$. This lets us assume that $G - R$ is acyclic as, otherwise, no λ -abiding orientation can be acyclic. Thus, we call every connected component T in $G - R$ a *tree*. For any $v \in R$, let $\mathcal{T}(v)$ denote the set of trees containing neighbors of v in G and, for any tree T , let $\mathcal{T}^{-1}(T)$ denote the set of vertices $v \in R$ with $T \in \mathcal{T}(v)$. Further, let $\mathcal{T}_0(v)$ denote the set of trees in $\mathcal{T}(v)$ that contain vertices v with $0 \in \lambda(v)$.

While parameterizing with the number of potential reticulations is particularly motivated for the phylogenetic version of the orientation problem, the algorithm from Section 4.3 can be used to solve PDCO if $|R|$ is small since, in the phylogenetic setting, all hazy vertices are potential reticulations. Therefore, we will focus on the DCAO problem here. Our algorithm for solving DCAO first applies a series of reduction rules to simplify the instance and allow us to draw important observations. We then proceed with a dynamic programming over subsets of the set R , running in $\mathcal{O}^*(2^{|R|})$ time.

► **Lemma 13.** *Let T be a tree in G . Let G^\rightarrow be an acyclic orientation of G with $\deg_{G^\rightarrow}^-(v) \in \lambda(v)$ for each $v \in V(T)$. Let $\sigma \in \text{Top}(G^\rightarrow)$ and let r_T be the minimum of $V(T)$ with respect to σ . In G^\rightarrow ,*

- (i) *either r_T is a source or r_T has a unique parent $v \in R$,*
- (ii) *each vertex $v \in V(T)$ with $v \neq r_T$ has a unique parent u and $u \in V(T)$,*
- (iii) *for each vertex $v \in V(T)$, there is a directed r_T - v -path, and*
- (iv) *if any vertex $v \in V(T)$ has an incoming arc (u, v) with $u \in R$, then $v = r_T$ and $u <_\sigma v$, and*
- (v) *for each $u, v \in \mathcal{T}^{-1}(T)$ with $u <_\sigma v$, all edges between T and v are oriented towards v .*

Proof. (i): Suppose r_T is not a source in G^\rightarrow . Since r_T is minimum with respect to σ , it has a parent in R . Moreover, r_T cannot have two parents in G^\rightarrow since $\lambda(r_T) \subseteq \{0, 1\}$ and $\deg_{G^\rightarrow}^-(r_T) \in \lambda(r_T)$.

(ii): Assume towards a contradiction that T contains a vertex $v \neq r_T$ with no parent in $V(T)$. Since T is a tree, it contains $|V(T)| - 1$ edges and, thus, the sum of in-degrees in $G^\rightarrow[V(T)]$ is $|V(T)| - 1$. Since r_T and v both have in-degree 0 in $G^\rightarrow[V(T)]$, the pigeonhole principle implies that there is a vertex w with in-degree at least two in $G^\rightarrow[V(T)]$, contradicting $\deg_{G^\rightarrow}^-(w) \in \lambda(w)$.

19:10 Finding Degree-Constrained Acyclic Orientations

(iii): This follows immediately from (ii) and the fact that T is acyclic even in G .

(iv): $v = r_T$ follows from (i) and (ii) and $u <_\sigma V(T)$ then follows from (iii).

(v): Assume towards a contradiction that there is an arc (v, w) in G^\rightarrow with $w \in V(T)$. By (iv), w is the root of T , so $u <_\sigma v <_\sigma w \leq_\sigma V(T)$. However, since $u \in \mathcal{T}^{-1}(T)$, there is also an arc (u, w') in G^\rightarrow with $w' \in V(T)$, contradicting (iv). ◀

In the following, we call the vertex r_T described in Lemma 13 the *root* of T in G^\rightarrow .

► **Reduction Rule 14.** *If G contains some v with $\lambda(v) = \emptyset$, then return “no”.*

► **Reduction Rule 15.** *Let $u \in V$ be a vertex with $\lambda(u) = \{0\}$. For each neighbor v of u , decrement all numbers in $\lambda(v)$ and delete u .*

Note that all trees T that do not contain a vertex v with $0 \in \lambda(v)$ have to receive an incoming arc from a vertex in R . If this vertex is unique for some T , then we can already orient this edge.

► **Reduction Rule 16.** *Let T be a tree in G such that $0 \notin \lambda(v)$ for all $v \in V(T)$. If $\mathcal{T}^{-1}(T) = \emptyset$, then return “no”. If there is some u with $\mathcal{T}^{-1}(T) = \{u\}$ and $|N_G(u) \cap V(T)| = 1$, then remove T from G .*

► **Reduction Rule 17.** *Let T be a tree in G and let $u \in R$. Let $X_T(u)$ denote the set of edges between u and T and let $\ell := |X_T(u)| \geq 2$. Then remove all edges in $X_T(u)$ and decrease all numbers in $\lambda(u)$ by ℓ .*

Correctness of Reduction Rule 17. We show that, in any λ -abiding orientation G^\rightarrow of G , all edges of $X_T(u)$ are directed towards u . Towards a contradiction, assume that some G^\rightarrow contains the arc (u, v) for some $v \in V(T)$. By Lemma 13(iv), v is the root of T in G^\rightarrow and $v \leq_\sigma T$ for all topological orders σ of G^\rightarrow . But then, all edges in $X_T(u)$ are oriented away from u in G^\rightarrow , contradicting Lemma 13(iv). ◀

In the following, we assume that G is reduced with respect to the reduction rules presented so far. In particular, for each $v \in R$ and each $T \in \mathcal{T}(v)$, there is a single edge between v and T in G and each tree T either contains a vertex u with $0 \in \lambda(u)$ or has at least two vertices in $\mathcal{T}^{-1}(T)$.

Dynamic programming on the subsets of R . Next, we describe a dynamic program that can decide whether an acyclic λ -abiding orientation exists for G . As usual, it can be augmented to actually construct such an orientation. Our dynamic programming table DP stores $\text{DP}[Q] = 1$ for $Q \subseteq R$ if and only if there is an acyclic orientation G^\rightarrow of G such that (a) $\deg_{G^\rightarrow}^-(v) \in \lambda(v)$ for all $v \in V \setminus (R \setminus Q)$ and (b) the vertices of Q precede the vertices of $R \setminus Q$ in some topological order of G^\rightarrow . Let us remark that \emptyset fulfills the conditions (a) and (b), so $\text{DP}[\emptyset] = 1$. Further, if $\text{DP}[R] = 1$, then G admits an acyclic λ -abiding orientation.

In the following, let $Q \subseteq R$ and suppose that G admits an acyclic λ -abiding orientation G^\rightarrow with a topological order σ satisfying $Q <_\sigma R \setminus Q$, that is, all vertices of Q precede all other vertices of R in σ . Let v be the maximum of Q with respect to σ , let $T \in \mathcal{T}(v)$ and let e be the edge in G between v and T . If $T \in \mathcal{T}(u)$ for some $u <_\sigma v$ then, by Lemma 13(v), e must be directed towards v in G^\rightarrow . Otherwise e may or may not be directed towards v in G^\rightarrow . Any vertex v whose list $\lambda(v)$ does not contradict this will be considered as a possible choice for a “last vertex of Q ” in the dynamic program.

► **Definition 18.** Let $Q \subseteq R$ and let $v \in Q$. Let $\alpha_Q(v) := \{T \in \mathcal{T}(v) \mid \mathcal{T}^{-1}(T) \cap Q \neq \{v\}\}$ be the set of trees that have an edge to v but also to another vertex in Q , and let $N^Q(v) := N(v) \cap Q$. If $\lambda(v)$ contains a number z with $|\alpha_Q(v)| \leq z - |N^Q(v)| \leq |\mathcal{T}(v)|$, we call v eligible with respect to Q .

The computation of $\text{DP}[Q]$ is then given by the following recursion:

$$\text{DP}[Q] := \begin{cases} 1 & \text{if } Q = \emptyset \\ 1 & \text{if } Q \text{ contains some } v \text{ that is eligible wrt. } Q \text{ and } \text{DP}[Q \setminus \{v\}] = 1 \\ 0 & \text{otherwise.} \end{cases}$$

► **Lemma 19.** Let $Q \subseteq R$. The definition of $\text{DP}[Q]$ matches its semantics, that is, $\text{DP}[Q] = 1$ if and only if there is an acyclic orientation G^{\rightarrow} of G such that

- (a) $\deg_{G^{\rightarrow}}^-(v) \in \lambda(v)$ for all $v \in V \setminus (R \setminus Q)$ and
- (b) the vertices of Q precede the vertices of $R \setminus Q$ in some topological order of G^{\rightarrow} .

Proof. First, note that the lemma holds for $Q = \emptyset$. Thus, by induction, suppose that the lemma holds for all Q' with $|Q'| = |Q| - 1$.

“ \Rightarrow ”: Let $\text{DP}[Q] = 1$, that is, Q contains some v that is eligible with respect to Q and $\text{DP}[Q'] = 1$ where $Q' := Q \setminus \{v\}$. By induction hypothesis, there is some orientation $G^{\rightarrow'}$ of G such that $\deg_{G^{\rightarrow'}}^-(u) \in \lambda(u)$ for all $u \in V \setminus (R \setminus Q')$ and the vertices of Q' precede the vertices of $R \setminus Q'$ in some topological order σ of $G^{\rightarrow'}$. Further, since v is eligible with respect to Q , there is some $t \in \mathbb{N}$ with $|\alpha_Q(v)| \leq t \leq |\mathcal{T}(v)|$ and $|N^Q(v)| + t \in \lambda(v)$. Hence, there is a size- t set \mathcal{X} with $\alpha_Q(v) \subseteq \mathcal{X} \subseteq \mathcal{T}(v)$.

Now, we modify $G^{\rightarrow'}$ into a new orientation G^{\rightarrow} as follows: First, for each $T \in \mathcal{X} \setminus \alpha_Q(v)$, pick any vertex $w \in V(T)$ with $0 \in \lambda(w)$ and orient all edges incident with a vertex in T away from w . Note that w exists since G is reduced with respect to Reduction Rule 16 and due to condition (b). The orientation is well-defined since T is a tree. Also note that the in-degrees of vertices in Q' remain unchanged since no vertex of Q' is adjacent to any vertex in such a T (since $T \notin \alpha_Q(v)$). Second, orient all edges between v and vertices $w \in R \setminus Q$ away from v .

Since no vertex in Q' has become a descendant of v and no vertex in $R \setminus Q$ has become an ancestor of v , we conclude that there is a topological order π of G^{\rightarrow} with $Q' <_{\pi} v <_{\pi} R \setminus Q$. This implies condition (b). Further, as all vertices in Q' precede v in π and v has exactly one arc incoming from each $T \in \mathcal{X}$, we conclude that $\deg_{G^{\rightarrow}}^-(v) = |N^{Q'}(v)| + t \in \lambda(v)$. This implies condition (a).

“ \Leftarrow ”: Let G^{\rightarrow} be an acyclic orientation of G with topological order σ such that conditions (a) and (b) are satisfied. Let v be the maximum with respect to σ of Q and let P denote the set of parents of v in G^{\rightarrow} . Clearly, we have $P \cap Q = N^Q(v)$. Further, by Lemma 13(iv), all edges between v and trees in $\alpha_Q(v)$ are oriented towards v in G^{\rightarrow} . Thus, $|N^Q(v)| + |\alpha_Q(v)| \leq \deg_{G^{\rightarrow}}^-(v) \leq |N^Q(v)| + |\mathcal{T}(v)|$. By condition (a), we have $\deg_{G^{\rightarrow}}^-(v) \in \lambda(v)$, implying that v is eligible with respect to Q . Further, since condition (a) holds for Q , it also holds for $Q' := Q \setminus \{v\}$ and, since v has been chosen as the maximum of Q with respect to σ , condition (b) also holds for Q' . By induction hypothesis, $\text{DP}[Q'] = 1$, implying $\text{DP}[Q] = 1$ by definition of $\text{DP}[Q]$. ◀

19:12 Finding Degree-Constrained Acyclic Orientations

Running Time. For the running time, first note that the reduction rules can be exhaustively applied in $\mathcal{O}((n+m) \cdot \max_\lambda)$ time. Second, for any set $Q \subseteq R$ and vertex $v \in Q$, eligibility of v with respect to Q can be checked in $\mathcal{O}(\deg_G(v) \cdot |Q|)$ with a linear-time preprocessing to determine $|\mathcal{T}(v)|$ for each v and $\mathcal{T}^{-1}(T)$ for each T . Thus, in total, the table can be computed in $\mathcal{O}(2^{|R|} \cdot |R| \cdot m + (n+m) \cdot \max_\lambda)$ time.

► **Theorem 20.** *DCAO can be solved in $\mathcal{O}(2^{|R|} \cdot |R| \cdot m + (n+m) \cdot \max_\lambda)$ time.*

5 Hardness with respect to treewidth

By Theorem 7, DCAO and DCO are XP with respect to treewidth. In this section, we prove also a corresponding negative result. Both, DCO and DCAO, are W[1]-hard with respect to treewidth.

► **Theorem 21.** *DCO is W[1]-hard with respect to treewidth.*

Proof. We reduce from MATCHING WITH LOWER AND UPPER QUOTAS (MLQ) which is known to be W[1]-hard with respect to treewidth [1, Thm. 6]. An instance of MLQ consists of a bipartite graph $G = (A \cup B, E)$, an integer $k \in \mathbb{N}$, and lower and upper quotas $\ell, u : B \rightarrow \mathbb{N}$. A solution to this MLQ instance is any subgraph F of G with $|E(F)| \geq k$ such that every vertex in $V(F) \cap A$ has degree 1 and every vertex $v \in V(F) \cap B$ has degree $\deg_F(v) \in [\ell(v), u(v)]$.

We construct a graph $D = (V(G) \cup \{\star\}, E')$ from G by connecting vertex \star to every vertex in A . Define $\lambda : V(D) \rightarrow 2^{\mathbb{N}}$ by

$$\begin{aligned} \lambda(\star) &:= \{0, \dots, |A| - k\}, \\ \lambda(a) &:= \{\deg_D(a) - 1\} \quad \forall a \in A, \\ \lambda(b) &:= \{0\} \cup [\ell(b), u(b)] \quad \forall b \in B. \end{aligned}$$

Note that $\text{tw}(D) \leq \text{tw}(G) + 1$ since $D - \{\star\} = G$. We claim that (D, λ) is a yes-instance of DCO if and only if (G, k, ℓ, u) is a yes-instance of MLQ.

“ \Rightarrow ”: Let α be an orientation of G such that every vertex $v \in V(D)$ has $\deg^-(v) \in \lambda(v)$. Then we define F to be the subgraph of G induced by those edges that are directed from A towards B . Clearly, this subgraph satisfies $\deg_F(b) \in [\ell(b), u(b)]$ for each $b \in B$ and $\deg_F(a) = 1$ for each $a \in A$. Also, since every $a \in A$ has a single outwards-directed edge, and at most $|A| - k$ of these must be directed towards \star , there are at least k edges in F .

“ \Leftarrow ”: Let F be a solution to MLQ. Then we construct an orientation of D by orienting each edge in $A \times B$ towards B if and only if that edge is contained in F . Furthermore, we orient each edge $\{\star, a\}$ with $a \in A$ towards a if and only if $a \in V(F)$. Clearly, our assumption on F then gives $\deg^-(b) \in \lambda(b)$ for every $b \in B$. Each vertex $a \in A$ has $\deg^+(a) = 1$, either because of an edge to B (if $a \in V(F)$), or because of an edge towards \star (if $a \notin V(F)$); thus $\deg^-(a) \in \lambda(a)$. Finally, $\deg^-(\star) = |A \setminus V(F)| \leq |A| - k$. This concludes the proof. ◀

► **Theorem 22.** *DCAO is W[1]-hard with respect to treewidth.*

Proof. We reduce from an DCO instance (G, λ) . To this end, subdivide each edge of G twice, and set $\lambda'(v) = \{0, 2\}$ for the newly created degree-2 vertices. Let G' be the resulting graph and $\lambda'(v) = \lambda(v)$ for all $v \in V(G)$. It is easy to see that there is a natural bijection between the λ -abiding orientations of G and the λ' -abiding orientations of G' . Since any λ' -abiding orientation of G' must be acyclic and $\text{tw}(G') = \text{tw}(G)$, this proves the claim. ◀

6 Conclusion

We analyzed three variants of graph orientation within the framework of parameterized complexity. With PDCO and DCAO, two of these variants require that a solution is acyclic. The requirement of acyclicity is hardly considered in literature but arises naturally when building up of phylogenetic networks.

We showed that PDCO and DCAO are NP-hard even if the maximum degree of the input graph is three and the potential in-degrees are consecutive. On the positive side, we established FPT-algorithms for DCAO and PDCO with respect to the number of vertices which have more than one option as in-degree, and with respect to the number of vertices having potentially two incoming edges.

Even though DCO and DCAO are $W[1]$ -hard when parameterized by the treewidth, all three problems are solvable in polynomial time if the input graph has constant treewidth. Therefore it is natural to ask whether DCO, DCAO or PDCO can be solved in polynomial time on graph classes more general than graphs with constant treewidth, such as planar graphs. To strengthen the result that DCO is FPT when parameterized by the total number of 2-gaps, it would be good to investigate whether DCO is FPT with respect to the number of vertices having a 2-gap, or a new parameter h -gap-index where the h -gap-index, similar to the h -vertex-index, is the smallest number h such that at least h vertices have at least 2-gaps.

References

- 1 Ashwin Arulselvan, Ágnes Cseh, Martin Groß, David F. Manlove, and Jannik Matuschke. Matchings with lower Quotas: Algorithms and Complexity. *Algorithmica*, 80(1):185–208, 2018. doi:10.1007/s00453-016-0252-6.
- 2 Laurent Bulteau and Mathias Weller. Parameterized Algorithms in Bioinformatics: An Overview. *Algorithms*, 12(12):256, 2019. doi:10.3390/A12120256.
- 3 Laurent Bulteau, Mathias Weller, and Louxin Zhang. On turning a Graph into a Phylogenetic Network. working paper or preprint, 2019. URL: <https://hal.science/hal-04085424>.
- 4 Gérard Cornuéjols. General Factors of Graphs. *Journal of Combinatorial Theory Series B*, 45(2):185–198, 1988. doi:10.1016/0095-8956(88)90068-8.
- 5 Joseph Felsenstein. *Inferring Phylogenies*. Sinauer Associates, 2 edition, 2003.
- 6 András Frank. On the Orientation of Graphs. *Journal of Combinatorial Theory Series B*, 28(3):251–261, 1980. doi:10.1016/0095-8956(80)90071-4.
- 7 Harold N. Gabow. Upper degree-constrained partial Orientations. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 554–563, 2006. URL: <https://dl.acm.org/doi/10.5555/1109557.1109618>.
- 8 András Gyárfás and András Frank. How to orient the edges of a graph. In *Proceedings of the Fifth Hungarian Combinatorial Colloquium, Keszthely*, pages 353–362, 1976. URL: https://users.renyi.hu/~gyarfas/Cikkek/09_FrankGyarfas_HowToOrientTheEdgesOfAGraph.pdf.
- 9 Katharina T. Huber, Leo van Iersel, Remie Janssen, Mark Jones, Vincent Moulton, Yukihiro Murakami, and Charles Semple. Orienting undirected Phylogenetic Networks, 2019. arXiv:1906.07430.
- 10 Daniel H. Huson, Regula Rupp, and Céline Scornavacca. *Phylogenetic Networks - Concepts, Algorithms and Applications*. Cambridge University Press, 2010.
- 11 Zoltán Király and Dömötör Pálvölgyi. Acyclic orientations with degree constraints, 2018. arXiv:1806.03426.
- 12 László Lovász. The Factorization of Graphs. In *Proceedings of the Calgary International Conference on Combinatorial Structures and their Applications*, 1970.

19:14 Finding Degree-Constrained Acyclic Orientations

- 13 László Lovász. The Factorization of Graphs. II. *Acta Mathematica Academiae Scientiarum Hungarica*, 23:223–246, 1972. doi:10.1007/BF01889919.
- 14 Luke Mathieson and Stefan Szeider. Editing Graphs to satisfy degree constraints: A parameterized approach. *Journal of Computer and System Sciences*, 78(1):179–191, 2012. doi:10.1016/j.jcss.2011.02.001.
- 15 Erin K Molloy, Arun Durvasula, and Sriram Sankararaman. Advancing admixture Graph estimation via maximum likelihood Network Orientation. *Bioinformatics*, 37(Supplement 1):i142–i150, July 2021. doi:10.1093/bioinformatics/btab267.
- 16 Thomas J. Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of the 10th Annual ACM symposium on Theory of computing (STOC)*, pages 216–226, 1978. doi:10.1145/800133.804350.
- 17 Feng Shi, Qilong Feng, Jianer Chen, Lusheng Wang, and Jianxin Wang. Distances between Phylogenetic Trees: A survey. *Tsinghua Science and Technology*, 18(5):490–499, 2013. doi:10.1109/TST.2013.6616522.
- 18 Katherine St. John. Review Paper: The shape of Phylogenetic Treespace. *Systematic Biology*, 66(1):e83–e94, June 2016. doi:10.1093/sysbio/syw025.

Graph Clustering Problems Under the Lens of Parameterized Local Search

Jaroslav Garvardt  

Institute of Computer Science, Friedrich Schiller University Jena, Germany

Nils Morawietz  

Institute of Computer Science, Friedrich Schiller University Jena, Germany

André Nichterlein  

Technische Universität Berlin, Germany

Mathias Weller  

Technische Universität Berlin, Germany

Abstract

CLUSTER EDITING is the problem of finding the minimum number of edge-modifications that transform a given graph G into a cluster graph G' , that is, each connected component of G' is a clique. Similarly, in the CLUSTER DELETION problem, we further restrict the sought cluster graph G' to contain only edges that are also present in G . In this work, we consider the parameterized complexity of a local search variant for both problems: LS CLUSTER DELETION and LS CLUSTER EDITING. Herein, the input also comprises an integer k and a partition \mathcal{C} of the vertex set of G that describes an initial cluster graph G^* , and we are to decide whether the “ k -move-neighborhood” of G^* contains a cluster graph G' that is “better” (uses less modifications) than G^* . Roughly speaking, two cluster graphs G_1 and G_2 are k -move-neighbors if G_1 can be obtained from G_2 by moving at most k vertices to different connected components.

We consider parameterizations by $k + \ell$ for some natural parameters ℓ , such as the number of clusters in \mathcal{C} , the size of a largest cluster in \mathcal{C} , or the cluster-vertex-deletion number (cvd) of G . Our main lower-bound results are that LS CLUSTER EDITING is W[1]-hard when parameterized by k even if \mathcal{C} has size two and that both LS CLUSTER DELETION and LS CLUSTER EDITING are W[1]-hard when parameterized by $k + \ell$, where ℓ is the size of the largest cluster of \mathcal{C} . On the positive side, we show that both problems admit an algorithm that runs in $k^{\mathcal{O}(k)} \cdot \text{cvd}^{3k} \cdot n^{\mathcal{O}(1)}$ time and either finds a better cluster graph or correctly outputs that there is no better cluster graph in the k -move-neighborhood of the initial cluster graph.

As an intermediate result, we also obtain an algorithm that solves CLUSTER DELETION in $\text{cvd}^{\text{cvd}} \cdot n^{\mathcal{O}(1)}$ time.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis; Theory of computation \rightarrow Parameterized complexity and exact algorithms

Keywords and phrases parameterized local search, permissive local search, FPT, W[1]-hardness

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.20

Funding *Jaroslav Garvardt*: Partially supported by the Carl Zeiss Foundation within the project “Interactive Inference”.

Nils Morawietz: Partially supported by the DFG, project OPERAH, KO 3669/5-1.

1 Introduction

Graph-based data clustering is a fundamental task with numerous applications [40]. Within this broad setting, we focus on the approach of modifying an input graph into a cluster graph (that is, a disjoint union of cliques) with as few edge modifications as possible. Herein, edges may be deleted or inserted, leading to the well-known CLUSTER EDITING or CORRELATION



© Jaroslav Garvardt, Nils Morawietz, André Nichterlein, and Mathias Weller; licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 20; pp. 20:1–20:19

Leibniz International Proceedings in Informatics



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

CLUSTERING [2, 4, 41] problem. If only edge deletions are allowed, then the problem is called CLUSTER DELETION [41]. One important advantage of the graph-modification approach is that the number of clusters is not part of the input but is determined implicitly.

CLUSTER DELETION and, in particular, CLUSTER EDITING are highly relevant in practice, with application areas ranging from bioinformatics [4] to data mining [2] and psychology [46]. Unfortunately, it is NP-hard to decide whether a given graph is at most a given number of modifications away from being a cluster graph [41]. Therefore, efforts have been made to circumvent this hardness. One particularly successful approach are parameterized algorithms [8, 13, 18, 23, 26, 37]. Given the amount of research and the practical relevance, it is no surprise that CLUSTER EDITING was selected as the problem for the sixth installment of the parameterized implementation challenge PACE 2021 [32]. The results revealed the strength of local search for CLUSTER EDITING: The top ten submissions in the heuristic track all involve local search. Moreover, the top three submissions always returned a solution less than 1.001 times larger than the best known solution, that is, the relative error is below 10^{-3} . This is in stark contrast to the best known polynomial-time approximation having an approximation factor of 2.06 [12], thus having a relative error that is three orders of magnitude larger!

In this work we complement the results of the PACE 2021 heuristic track with a theoretical study of the local search problems associated with CLUSTER EDITING and CLUSTER DELETION. More precisely, we study the following question: Can we improve a given clustering¹ of the input graph G by “moving” at most k vertices to different clusters? Many local search algorithms submitted to PACE 2021 try to move vertices between clusters to improve their solution [3, 7, 22, 32, 42]. For CLUSTER EDITING we are free to move any vertex in any cluster (inserting missing edges within a cluster and deleting edges between clusters) while, for CLUSTER DELETION, we have to ensure that there are no missing edges within any cluster we create. The respective local search versions of the problems are called LS-CLUSTER EDITING and LS-CLUSTER DELETION (see Section 2 for precise problem definitions).

Related Work. In the more general setting, our work fits into the theme of *parameterized local search*. Unfortunately, most parameterized local search problems turn out to be W[1]-hard with respect to their local search radius k [9, 15, 17, 21, 27, 28, 33, 39, 43]. Consequently, one often tries to combine k with some structural parameter ℓ in the hope of obtaining an FPT algorithm. Experimental evaluation showed that such algorithms often achieve very good solutions quickly [19, 20, 25, 29, 31].

Dörnfelder et al. [15] already proved the W[1]-hardness of a local search version of CLUSTER EDITING with a different local neighborhood: they search for a better solution by modifying at most k edges in the given solution. Recently, Luo et al. [38] considered the closely related DYNAMIC CLUSTER EDITING problem, in which a given clustering \mathcal{C} for a graph G has to be adapted while the graph G changes dynamically, keeping the modification distance between them low. In this setting, the main question is whether minor changes to \mathcal{C} are sufficient to produce a good clustering for the (slightly) changed new graph. Since, in this setting, the old graph G is actually irrelevant for the computational problem, there are only minor technical differences to LS-CLUSTER EDITING. Luo et al. [38] measure the distance to \mathcal{C} by the number of vertices moving to a different cluster (they call this “matching distance”) – the same measure we use. They analyzed the parameterized complexity of

¹ A clustering can be viewed as an equivalence relation (“which vertices will end up in the same cluster?”) or, equivalently, a partition of the vertex set of G .

DYNAMIC CLUSTER EDITING with respect to the solution size ℓ (number of edges to modify in G) and the number k of changes allowed to be done to the given solution. They obtained W[1]-hardness for each of the parameters k and ℓ individually and fixed-parameter tractability with respect to the combined parameter $k + \ell$.

Besides the work on DYNAMIC CLUSTER EDITING, there are numerous works on CLUSTER EDITING, including search-tree based algorithms [8, 23], kernelizations [10, 13, 18, 26], and above lower-bound parameterizations [6, 37]. For CLUSTER DELETION, there are search-tree based algorithms [30, 45] and problem kernels [10, 11].

Our Results. We consider the number k of vertices allowed to be moved to different clusters, since this number can be expected to be small (mostly one-digit values in PACE 2021 local search submissions). We show that LS-CLUSTER DELETION and LS-CLUSTER EDITING are – like many other local search problems – W[1]-hard with respect to the local search radius k , so we try to combine k with different structural parameters. If a parameter combination $k + \ell$ allows for fixed-parameter tractability, then we aim for running times of the form $\ell^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ as advocated by Komusiewicz and Morawietz [34]. The motivation is that k is expected to be much smaller than even ℓ which, in turn, is hopefully considerably smaller than n . Thus, the resulting algorithms are expected to be very efficient in practice, which is particularly important since local search subroutines are called excessively in the solvers (see for example the PACE 2021 solvers [32]).

We combine k with the maximum degree Δ , the maximum size of any clique in the given solution, or the cluster vertex deletion number cvd , that is, the number of vertices to remove to obtain a cluster graph. In Section 3, we present algorithms with running times of the form $(\ell \cdot k)^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ for $\ell = \Delta$ (see Theorem 3.3) and $\ell = \text{cvd}$ (see Corollary 3.8 and Theorem 3.9). As an intermediate result, we obtain an algorithm solving CLUSTER DELETION in $\text{cvd}^{\text{cvd}} \cdot n^{\mathcal{O}(1)}$ time (see Theorem 3.6). We complement the algorithms for LS-CLUSTER DELETION and LS-CLUSTER EDITING with lower bounds in Section 4. In particular, we show that LS-CLUSTER EDITING is W[1]-hard with respect to the sum of k , the maximum cluster size, and the degeneracy of the input graph G (see Theorem 4.2) and that LS-CLUSTER EDITING is W[1]-hard with respect to k in the restricted case that the given clustering consists of only two clusters (see Theorem 4.5). LS-CLUSTER DELETION is also W[1]-hard with respect to the sum of k and the maximum cluster size (see Theorem 4.6). Moreover, the employed reductions also show that neither LS-CLUSTER EDITING nor LS-CLUSTER DELETION can be solved in $f(k) \cdot n^{o(k)}$ time unless the ETH fails. This shows that in the above mentioned algorithms the $\mathcal{O}(k)$ in the exponent cannot be replaced by $o(k)$.

Due to space restriction, proofs of statements marked with (\star) are deferred to a full version.

2 Preliminaries

For details about relevant definitions of parameterized complexity such as fixed-parameter tractability, W[1]-hardness, parameterized reductions, kernelization, and ETH, we refer to the standard monographs [14, 16].

Let X and Y be sets. We denote by $A \oplus B := (A \setminus B) \cup (B \setminus A)$ the *symmetric difference* between A and B . A *partition* \mathcal{P} of X is a collection of non-empty and pairwise disjoint subsets of X such that $\cup_{P \in \mathcal{P}} P = X$. Moreover, for an integer k , we denote by $\binom{X}{k}$ the collection of all size- k subsets of X . Let $G = (V, E)$ be a graph. For a vertex set $S \subseteq V$, we denote by $E_G(S) := \binom{S}{2} \cap E$ the edges of G between the vertices of S . If G is clear from the

context, we may omit the subscript. A partition \mathcal{C} of V is called a *clustering* of G . Each vertex set C of \mathcal{C} is called a *cluster*. For a clustering \mathcal{C} , we denote by $E(\mathcal{C}) := \bigcup_{C \in \mathcal{C}} \binom{C}{2}$ the edges inside the clusters of \mathcal{C} .

We call a function $\chi: V \rightarrow \mathbb{N}$ a *coloring* of V . We say that color $i \in \mathbb{N}$ is *used by* χ if there is at least one vertex $v \in V$ with $\chi(v) = i$. For a coloring χ , let \mathcal{C}_χ denote the clustering of G where for each color i used by χ , $\chi^{-1}(i) = \{v' \in V \mid \chi(v') = i\}$ is a cluster of \mathcal{C}_χ . We call χ a *cluster-coloring* of \mathcal{C}_χ and we call \mathcal{C}_χ the *clustering* of χ . Note that each clustering has infinitely many cluster-colorings and that all these cluster-colorings are identical with respect to isomorphism. Here, two colorings χ and χ' are *isomorphic* if there is a bijection $f: \mathbb{N} \rightarrow \mathbb{N}$ such that $\chi = f \circ \chi'$.

Let χ and χ' be colorings of V . We denote by $\text{Move}(\chi, \chi') := \{v \in V \mid \chi(v) \neq \chi'(v)\}$ the vertices that receive different colors under χ and χ' . Moreover, we set $\text{move}(\chi, \chi') := |\text{Move}(\chi, \chi')|$. Two colorings χ and χ' are *k-move-neighbors* if $\text{move}(\chi, \chi') \leq k$. Analogously, two clusterings \mathcal{C} and \mathcal{C}' are *k-move-neighbors* if there is a cluster-coloring χ of \mathcal{C} and a cluster-coloring χ' of \mathcal{C}' such that χ and χ' are *k-move-neighbors*. We also denote by $\text{move}(\mathcal{C}, \mathcal{C}')$ the smallest integer k such that \mathcal{C} and \mathcal{C}' are *k-move-neighbors*. Note that $\text{move}(\mathcal{C}, \mathcal{C}')$ can be computed in polynomial time [38]. For a clustering \mathcal{C} we define $\text{cost}(\mathcal{C}) := |E(\mathcal{C}) \oplus E|$. A clustering \mathcal{C}' is *improving* over a clustering \mathcal{C} , if $\text{cost}(\mathcal{C}') < \text{cost}(\mathcal{C})$.

For a function $f: A \rightarrow B$ and $C \subseteq A$ we denote by $f|_C$ the function f restricted to C .

► **Observation 2.1.** *Let G be a graph and let \mathcal{C} and \mathcal{C}' be clusterings of G with $\text{move}(\mathcal{C}, \mathcal{C}') \leq k$. Then, $|\mathcal{C} \cap \mathcal{C}'| \geq |\mathcal{C}| - 2k$.*

In this work, we consider the parameterized complexity of the following problems.

LS-CLUSTER DELETION

Input: An undirected graph $G = (V, E)$, an integer k and a clustering \mathcal{C} of G with $E(\mathcal{C}) \subseteq E$.

Question: Is there an improving *k-move-neighbor* \mathcal{C}' of \mathcal{C} for G with $E(\mathcal{C}') \subseteq E$?

LS-CLUSTER EDITING

Input: An undirected graph $G = (V, E)$, an integer k and a clustering \mathcal{C} of G .

Question: Is there an improving *k-move-neighbor* \mathcal{C}' of \mathcal{C} for G ?

Further, we also analyze the *permissive* version of both problems, that is, the problem, where we want to find any better solution or correctly output that the given solution has no improving *k-move-neighbor*. Thus, the permissive problem variants allow to return better clusterings even if these are not *k-move-neighbors* of \mathcal{C} .

3 Algorithms for Permissive Problem variants

In this section, we present our algorithmic results. We start with the parameterization maximum degree Δ and k . A first observation allows us to assume that any given clustering in our instance of LS-CLUSTER EDITING consists exclusively of clusters that have diameter at most two in the input graph. Note that for LS-CLUSTER DELETION, the input clusters in the given clustering must all have diameter one, that is, they are cliques in the input graph.

► **Observation 3.1.** *Let G be a graph and let \mathcal{C} be a clustering of G . If there is a cluster $C \in \mathcal{C}$ that has diameter at least three in G , then there is an improving 1-move-neighbor \mathcal{C}' of \mathcal{C} .*

Proof. Let $u, v \in C$ be two vertices of distance at least three in G . Thus, we have $N(u) \cap N(v) = \emptyset$. Assume without loss of generality that $|N(v) \cap C| \leq |N(u) \cap C|$.

We show that $C' := (C \setminus \{C\}) \cup \{C \setminus \{v\}, \{v\}\}$ is an improving 1-move-neighbor of C . Clearly, C' is a 1-move-neighbor of C as only v moves to a previously empty cluster. Moreover, this move costs $|N(v) \cap C|$ edge deletions but saves at least $|N(u) \cap C| + 1$ many edge insertions. Since $|N(v) \cap C| \leq |N(u) \cap C|$, it follows that C' is improving over C . ◀

With the knowledge that the given clusters are connected, it is not too hard to see that we can restrict ourselves to looking for solutions where the vertices that change clusters are not too far apart from each other; otherwise we can simply ignore parts of the changes and obtain a better clustering by moving even fewer vertices. The formal statement is as follows:

► **Lemma 3.2** (★). *Let G be a graph and C a clustering and C' an improving clustering for C . Let C_1, \dots, C_q be the clusters in C that change in C' (that is, $\{C_1, \dots, C_q\} := C \setminus C'$). If $G[C_1 \cup \dots \cup C_q]$ is not connected, then there is an improving clustering C^* of C with $\text{move}(C, C^*) < \text{move}(C, C')$. Moreover, if $E(C') \subseteq E$, then $E(C^*) \subseteq E$.*

With Observation 3.1 and Lemma 3.2 we can show that checking for solutions for LS-CLUSTER EDITING and LS-CLUSTER DELETION boils down to finding subgraphs of bounded size in graphs of bounded degrees for which we can apply an algorithm of Komusiewicz and Sommer [35]. Overall this results in the following statement.

► **Theorem 3.3** (★). *LS-CLUSTER EDITING and LS-CLUSTER DELETION can be solved in $\Delta^{8k}(6ek)^{2k+1}n^{\mathcal{O}(1)}$ time.*

We remark that an algorithm solving LS-CLUSTER EDITING or LS-CLUSTER DELETION in $f(\Delta) \cdot n^{\mathcal{O}(1)}$ time is unlikely: Setting $k := n$ and applying this algorithm repeatedly until no improvement is found would solve CLUSTER EDITING or CLUSTER DELETION in $f(\Delta) \cdot n^{\mathcal{O}(1)}$ time; this is unlikely as both problems are NP-hard for constant maximum degree [36].

Parameterization by cluster vertex deletion number and k . In the remainder of the section we provide two algorithms exploiting small modulators to cluster graphs – one for LS-CLUSTER DELETION in Section 3.1 and one for LS-CLUSTER EDITING in Section 3.2. For both of our algorithms, we use the following notation. We say that a vertex set $M \subseteq V$ is a (*cluster*) *modulator* of G if $G[V \setminus M]$ is a cluster graph. Let \mathcal{B} be the connected components of $G[V \setminus M]$. We call each clique $B \in \mathcal{B}$ a *bag*. We denote by $\text{cvd}(G)$ the cluster vertex deletion number, that is, the size of a smallest cluster modulator of G . If the graph G is clear from the context, we may simply write cvd .

3.1 An Algorithm for LS-Cluster Deletion

Let $G = (V, E)$ be a graph and let C be a clustering of G . Moreover, let $S \subseteq V$ and let C_S be a clustering of $G[S]$. We say that C *extends* C_S if for each cluster $C \in C_S$ there is a cluster $C' \in C$ with $C' \cap S = C$.

► **Lemma 3.4.** *Let $G = (V, E)$ be a graph and let M be a cluster modulator of G . Moreover, let C_M be a given clustering of $G[M]$ with $E(C_M) \subseteq E$. In $3^{|C_M|} \cdot n^{\mathcal{O}(1)}$ time, one can find a best clustering C of G among all clusterings C' of G with $E(C') \subseteq E$ that extend C_M .*

Proof. Note that for each clustering \mathcal{C} of G that extends \mathcal{C}_M , the edges between vertices of M in both $E(\mathcal{C})$ and $E(\mathcal{C}_M)$ are equal. Consequently, the task can also be reformulated as follows: find a clustering \mathcal{C}^* of G that maximizes the number of edges having at least one endpoint in $V \setminus M$ among all clusterings \mathcal{C} of G with $E(\mathcal{C}) \subseteq E$ that extend \mathcal{C}_M . In the following, we describe a dynamic program solving this reformulated task.

Fix an arbitrary ordering of the bags and let B_i denote the i th bag of \mathcal{B} according to this ordering. The dynamic programming table T has entries of type $T[X, i]$ with $X \subseteq \mathcal{C}_M$ and $i \in [0, |\mathcal{B}|]$. For $X \subseteq \mathcal{C}_M$ and $i \in [0, |\mathcal{B}|]$, let V_X denote the union of all clusters of X and let V_X^i denote the union of V_X and the first i bags. The entry $T[X, i]$ stores the maximal number of edges having at least one endpoint in $V_X^i \setminus M$ of any clustering \mathcal{C}_X^i of $G[V_X^i]$ with $E(\mathcal{C}) \subseteq E$ that extends X . Intuitively, this entry stores the best way to distribute the vertices of the first i bags among the clusters of X .

To compute an entry $T[X, i]$, we iterate over all subsets X' of X and check for the best way to distribute the vertices of the first $i - 1$ bags among the clusters of X' and the best way to distribute the vertices of the i th bag among the clusters of $X \setminus X'$.

Formally, for each $X \subseteq \mathcal{C}_M$, we set $T[X, 0] := 0$ and for each $i \in [1, |\mathcal{B}|]$, we set

$$T[X, i] := \max_{X' \subseteq X} T[X', i - 1] + \text{gain}_i^{X \setminus X'}.$$

Here, $\text{gain}_i^{X \setminus X'}$ is the number of edges having at least one endpoint in bag B_i of the best way to distribute the vertices of B_i among the clusters of $X \setminus X'$. This recurrence is correct because no cluster C in a clustering \mathcal{C} of G can contain vertices of two distinct bags without violating $E(\mathcal{C}) \subseteq E$.

In the following, we describe how to compute gain_i^Y for each $i \in [1, |\mathcal{B}|]$ and each $Y \subseteq \mathcal{C}_M$.

▷ **Claim 3.5.** In $3^{|\mathcal{C}_M|} \cdot n^{\mathcal{O}(1)}$ time, the values gain_i^Y can be computed for all $i \in [1, |\mathcal{B}|]$ and all $Y \subseteq \mathcal{C}_M$.

Proof. The computation of this value relies on the following observation: Let \mathcal{C} be a best clustering of $G[V_Y \cup B_i]$ with $E(\mathcal{C}) \subseteq E$ that extends Y . Moreover, let C be a largest cluster of \mathcal{C} . Then C contains all vertices of B_i that are adjacent to all vertices of $C \cap M$. This is true, since if there would be a cluster C' in \mathcal{C} containing a vertex v of $\{v \in B_i \mid C \subseteq N(v)\}$, then $\mathcal{C}' := (\mathcal{C} \setminus \{C, C'\}) \cup \{C \cup \{v\}, C' \setminus \{v\}\}$ is a clustering with $E(\mathcal{C}') \subseteq E$ that improves over \mathcal{C} . The properties of \mathcal{C}' hold since C is a largest cluster of \mathcal{C} and B_i is a clique in G .

Hence, to solve this intermediate task, we can branch which cluster C of $Y \cup \{\emptyset\}$ will be extended to be the largest cluster in \mathcal{C} , add all vertices of B_i to C that may fit into this cluster and solve the task recursively.

This can also be done by a dynamic program. We introduce the dynamic programming table D_i with entries of type $D_i[Y, Z]$ with $Y \subseteq \mathcal{C}_M$ and $Z \subseteq Y$.

For each set $Z \subseteq Y$, we let $\text{Remain}_Y^Z := B_i \setminus \left(\bigcup_{C \in Y \setminus Z} \{v \in B_i \mid C \subseteq N(v)\} \right)$ denote the set of vertices of B_i that do not fit in any cluster of $Y \setminus Z$. The entry $D_i[Y, Z]$ stores the maximal number of edges having at least one endpoint in Remain_Y^Z of any clustering \mathcal{C}_Z^i of $G[\text{Remain}_Y^Z \cup \bigcup_{C \in Y} C]$ with $E(\mathcal{C}) \subseteq E$ that extend Z .

For each $Y \subseteq \mathcal{C}_M$, we set $D_i[Y, \emptyset] := \binom{|\text{Remain}_Y^\emptyset|}{2}$ and for each non-empty $Z \subseteq Y$, we set

$$D_i[Y, Z] := \max \left(\binom{|\text{Remain}_Y^Z|}{2}, \max_{C \in Z} \binom{|\text{Fit}_C|}{2} + |\text{Fit}_C| \cdot |C| + D_i[Y, Z \setminus \{C\}] \right),$$

where $\text{Fit}_C := \{v \in \text{Remain}_Y^Z \mid C \subseteq N(v)\}$ denotes the set of vertices of Remain_Y^Z that fit into the cluster C .

This recurrence is correct by the above observation: in each optimal clustering \mathcal{C}_Z^i of $G[\text{Remain}_Y^Z \cup \bigcup_{C \in Y} C]$ with $E(\mathcal{C}) \subseteq E$ that extend Z , there is a largest cluster $C' \in \mathcal{C}_Z^i$ that contains all vertices of $\text{Fit}_{C' \cap M}$.

Finally, for each $Y \subseteq \mathcal{C}_M$, we set $\text{gain}_i^Y := D_i[Y, Y]$. Since for each $i \in [1, |\mathcal{B}|]$, the table D_i contains $3^{|\mathcal{C}_M|}$ entries and each such entry can be computed in $n^{\mathcal{O}(1)}$ time, the values gain_i^Y can be computed for all $i \in [1, |\mathcal{B}|]$ and all $Y \subseteq \mathcal{C}_M$ in the stated running time.

Moreover, note that a corresponding clustering can be computed via traceback. \triangleleft

Let \mathcal{C}^* be any best clustering of G among all clusterings \mathcal{C} of G with $E(\mathcal{C}) \subseteq E$ that extend \mathcal{C}_M . Then, the number of edges of $E(\mathcal{C}^*)$ having at least one endpoint in any bag is stored in $T[\mathcal{C}_M, |\mathcal{B}|]$. Moreover, a corresponding clustering can be found via traceback.

Since for each $i \in [0, |\mathcal{B}|]$ and each $X \subseteq \mathcal{C}_M$, the table entry $T[X, i]$ can be computed in $2^{|X|} \cdot n^{\mathcal{O}(1)}$ time and there are $2^{|\mathcal{C}_M|}$ choices for X , each table entry of T can be computed in time $\sum_{i=1}^{|\mathcal{C}_M|} \binom{|\mathcal{C}_M|}{i} \cdot 2^i \cdot n^{\mathcal{O}(1)} \subseteq 3^{|\mathcal{C}_M|} \cdot n^{\mathcal{O}(1)}$. \blacktriangleleft

Note that this implies the following FPT-algorithm for CLUSTER DELETION when parameterized by cvd : First, compute a minimum cluster modulator M of G in $1.811^{\text{cvd}} \cdot n^{\mathcal{O}(1)}$ time [44]. Second, iterate over all possible clusterings of $G[M]$ and apply the algorithm behind Lemma 3.4. This implies a running time of $1.811^{\text{cvd}} \cdot \mathbf{B}_{\text{cvd}} \cdot n^{\mathcal{O}(1)}$, where \mathbf{B}_{cvd} is the cvd -th Bell number which denotes the number of partitions of a set of size cvd . Since for each $n \in \mathbb{N}$, $\mathbf{B}_n < (\frac{n}{\ln(n+1)})^n$ [5], this implies the following.

► **Theorem 3.6.** *CLUSTER DELETION can be solved in $\text{cvd}^{\text{cvd}} \cdot n^{\mathcal{O}(1)}$ time.*

Next, we show how we can use Lemma 3.4 to obtain a permissive algorithm for LS-CLUSTER DELETION when parameterized by cvd and k .

► **Theorem 3.7.** *Let G be a graph and let M be a given cluster modulator of G . Moreover, let \mathcal{C} be a clustering of G with $E(\mathcal{C}) \subseteq E$ and let $k \in \mathbb{N}$. In $|\mathcal{M}|^{2k} \cdot \binom{|M|}{k} \cdot k^k \cdot 3^{4k} \cdot n^{\mathcal{O}(1)}$ time, one can find a clustering \mathcal{C}' of G with $E(\mathcal{C}') \subseteq E$ that is at least as good as a best clustering \mathcal{C}^* of G with $E(\mathcal{C}^*) \subseteq E$ and $\text{move}(\mathcal{C}, \mathcal{C}^*) \leq k$.*

Proof. Let \mathcal{C}^* be a clustering of G that maximizes $|E(\mathcal{C}^*)|$ among all clusterings \mathcal{C}'' of G with $E(\mathcal{C}'') \subseteq E$ and $\text{move}(\mathcal{C}, \mathcal{C}'') \leq k$. Moreover, let $\mathcal{M} := \{C \in \mathcal{C} \mid C \cap M \neq \emptyset\}$ and $\mathcal{M}^* := \{C \in \mathcal{C}^* \mid C \cap M \neq \emptyset\}$ denote the sets of clusters intersecting M , of \mathcal{C} and \mathcal{C}^* , respectively.

Let $\mathcal{C}_M := \{C \cap M \mid C \in \mathcal{M}\}$ denote the clusters of \mathcal{M} restricted to the vertices of M . Similarly, let $\mathcal{C}_M^* := \{C \cap M \mid C \in \mathcal{M}^*\}$. Note that $\text{move}(\mathcal{C}, \mathcal{C}^*) \leq k$ implies $\text{move}(\mathcal{C}_M, \mathcal{C}_M^*) \leq k$. Hence, to find a clustering \mathcal{C}' of G that is at least as good as \mathcal{C}^* , it suffices to enumerate all clusterings \mathcal{C}'_M of $G[M]$ with $\text{move}(\mathcal{C}_M, \mathcal{C}'_M) \leq k$ (which includes \mathcal{C}_M^*) and to compute, for each such clustering \mathcal{C}'_M , any best clustering for G that extends \mathcal{C}'_M . As the latter task can be done in $3^{|\mathcal{C}'_M|} \cdot n^{\mathcal{O}(1)}$ time by Lemma 3.4, it remains to describe how one can enumerate all such clusterings \mathcal{C}'_M of $G[M]$.

This can be done in $\binom{|M|}{k} \cdot (|\mathcal{M}| + k)^k \cdot n^{\mathcal{O}(1)}$ time by iterating over all possible subsets $M' \subseteq M$ of size k and iterating over all possible ways to move these k vertices into any of the clusters of \mathcal{M} (including the clusters where these vertices came from) or opening a new cluster.

Hence, this algorithm runs in $\binom{|M|}{k} \cdot (|\mathcal{M}| + k)^k \cdot 3^{|\mathcal{M}| + k} \cdot n^{\mathcal{O}(1)}$ time. Note that this is not the desired running time since $|\mathcal{M}|$ occurs in the exponent of the running time and might be much larger than k .

To still obtain the desired running time, we perform some initial branching if $|\mathcal{M}| > 2k$. The idea behind this initial branching relies on Observation 2.1. Since at most k vertices were moved to obtain \mathcal{M}^* from \mathcal{M} , Observation 2.1 implies $|\mathcal{M} \cap \mathcal{M}^*| \geq |\mathcal{M}| - 2k$. In other words, there is a subset $\mathcal{M}' \subseteq \mathcal{M}$ of size at most $2k$ such that $\mathcal{M} \setminus \mathcal{M}' \subseteq \mathcal{M} \cap \mathcal{M}^*$. This implies that all edge modifications having at least one endpoint in any cluster of $\mathcal{M} \setminus \mathcal{M}'$ are identical in both $E(\mathcal{C})$ and $E(\mathcal{C}^*)$. Hence, by applying for each subset $\mathcal{M}' \subseteq \mathcal{M}$ of size at most $2k$ the above described algorithm on the graph $G[V \setminus (\cup_{C \in \mathcal{M} \setminus \mathcal{M}'} C)]$, we find a clustering \mathcal{C}' with $E(\mathcal{C}') \subseteq E$ which is at least as good as \mathcal{C}^* . This initial branching can be done in $|\mathcal{M}|^{2k} \cdot n^{\mathcal{O}(1)} \subseteq |M|^{2k} \cdot n^{\mathcal{O}(1)}$ time.

Since for each such branching-instance, there are at most $2k$ clusters containing vertices of M , the whole running time evaluates to $|M|^{2k} \cdot \binom{|M|}{k} \cdot (3k)^k \cdot 3^{3k} \cdot n^{\mathcal{O}(1)} = |M|^{2k} \cdot \binom{|M|}{k} \cdot k^k \cdot 3^{4k} \cdot n^{\mathcal{O}(1)}$ time. \blacktriangleleft

Since a cluster modulator can be 2-approximated in polynomial time [1], Theorem 3.7 implies the following:

► **Corollary 3.8.** *The permissive version of LS-CLUSTER DELETION can be solved in $(k^k + 2^{\mathcal{O}(k)} \cdot \text{cvd}^{3k}) \cdot n^{\mathcal{O}(1)}$ time.*

Proof. Let $I := (G = (V, E), k, \mathcal{C})$ be an instance of LS-CLUSTER DELETION. First, check in $1.811^k \cdot n^{\mathcal{O}(1)}$ time, whether G has a cluster modulator of size at most k [44]. If this is the case, one can find an optimal clustering \mathcal{C}' for G with $E(\mathcal{C}') \subseteq E$ in time $\text{cvd}^{\text{cvd}} \cdot n^{\mathcal{O}(1)} \subseteq k^k \cdot n^{\mathcal{O}(1)}$ due to Theorem 3.6. Otherwise, $k < \text{cvd}$. In this case, one can 2-approximate a cluster modulator M in polynomial time [1] and find a clustering \mathcal{C}' of G with $E(\mathcal{C}') \subseteq E$ that is at least as good as a best clustering \mathcal{C}^* of G with $E(\mathcal{C}^*) \subseteq E$ and $\text{move}(\mathcal{C}, \mathcal{C}^*) \leq k$ in time $|M|^{2k} \cdot \binom{|M|}{k} \cdot k^k \cdot 3^{4k} \cdot n^{\mathcal{O}(1)}$ due to Theorem 3.7. Since $k < \text{cvd} \leq |M|$, we get that $\binom{|M|}{k} \cdot k^k \leq |M|^k \cdot \frac{k^k}{k!} \leq |M|^k \cdot 2^{\mathcal{O}(k)}$. Hence, the running time of the case $k < \text{cvd}$ evaluates to $2^{\mathcal{O}(k)} \cdot |M|^{3k} \cdot n^{\mathcal{O}(1)} \subseteq 2^{\mathcal{O}(k)} \cdot (2 \cdot \text{cvd})^{3k} \cdot n^{\mathcal{O}(1)} = 2^{\mathcal{O}(k)} \cdot \text{cvd}^{3k} \cdot n^{\mathcal{O}(1)}$ time. In both cases, the running time is upper-bounded by $(k^k + 2^{\mathcal{O}(k)} \cdot \text{cvd}^{3k}) \cdot n^{\mathcal{O}(1)}$ time. \blacktriangleleft

3.2 An Algorithm for LS-Cluster Editing

In this subsection, we present a permissive algorithm for LS-CLUSTER EDITING with a running time similar to the one for LS-CLUSTER DELETION.

► **Theorem 3.9.** *Let $G = (V, E)$ be a graph, let \mathcal{C} be a clustering of G , and let $k \in \mathbb{N}$. In $2^{\mathcal{O}(k)} \cdot k^k \cdot \text{cvd}^{3k} \cdot n^{\mathcal{O}(1)}$ time, one can find a clustering that improves over \mathcal{C} or correctly output that there is no clustering \mathcal{C}' of G with $\text{move}(\mathcal{C}, \mathcal{C}') \leq k$ that improves over \mathcal{C} .*

To give a better intuition for the following algorithm, we switch to the interpretation of LS-CLUSTER EDITING where the initial solution is a cluster coloring $\chi_{\mathcal{C}}$ of \mathcal{C} . That is, if the given coloring $\chi_{\mathcal{C}}$ can be improved within the k -move-neighborhood, then we need to find *any* coloring χ^* improving over $\chi_{\mathcal{C}}$. This coloring χ^* is not required to be in the k -move-neighborhood of $\chi_{\mathcal{C}}$.

Let $I := (G = (V, E), k, \chi_{\mathcal{C}})$ be an instance of LS-CLUSTER EDITING, let M be a cluster modulator of G . Let $\alpha \in \mathbb{N}$ be a color used by $\chi_{\mathcal{C}}$. We say that α is a *modulator color* if $\chi_{\mathcal{C}}^{-1}(\alpha) \cap M \neq \emptyset$. Otherwise, we say that α is a *bag color*. In the remainder of this section, we denote by col_{Mod} and col_{Bag} the set of modulator colors and bag colors of $\chi_{\mathcal{C}}$, respectively. Note that these sets of colors are defined with respect to the initial coloring $\chi_{\mathcal{C}}$ of the LS-CLUSTER EDITING-instance I . Recall that each bag $B \in \mathcal{B}$ is a clique in G and a connected component of $G[V \setminus M]$. Fix an arbitrary ordering of the bags and let B_i denote the i th bag of \mathcal{B} according to this ordering.

We first show that we can improve the initial coloring in polynomial time unless it has some properties that we will exploit in the following.

► **Observation 3.10.** *If there is a bag color α such that vertices of two distinct bags receive color α under χ_C , then one can find a coloring χ' of V in polynomial time that improves over χ_C .*

► **Observation 3.11.** *If there is a bag $B \in \mathcal{B}$ such that two vertices of B receive distinct bag colors under χ_C , then one can find a coloring χ' of V in polynomial time that improves over χ_C .*

Hence, we assume in the following, that for each bag color $\alpha \in \text{col}_{\text{Bag}}$, $\chi_C^{-1}(\alpha)$ contains only vertices of a single bag and that for each bag $B_i \in \mathcal{B}$, there is at most one bag color $\alpha_i \in \text{col}_{\text{Bag}}$ with $\chi_C^{-1}(\alpha_i) \subseteq B_i$.

Moreover, we assume in the following that col_{Mod} has size $\mathcal{O}(k)$. In the final algorithm we use an initial branching – similar to the one used in the algorithm behind Theorem 3.7 – to ensure that this assumption is fulfilled.

Let χ' be a coloring of V and let χ_{int} be the coloring that agrees with χ_C on all vertices of $V \setminus M$ and that agrees with χ' on all vertices of M . We call χ_{int} the *intermediate coloring* for χ' . The idea behind this definition is the following.

► **Observation 3.12.** *Let χ' be a coloring of V . Moreover, let χ_{int} be the intermediate coloring for χ' . It holds that $\text{move}(\chi_C, \chi_{\text{int}}) + \text{move}(\chi_{\text{int}}, \chi') = \text{move}(\chi_C, \chi')$.*

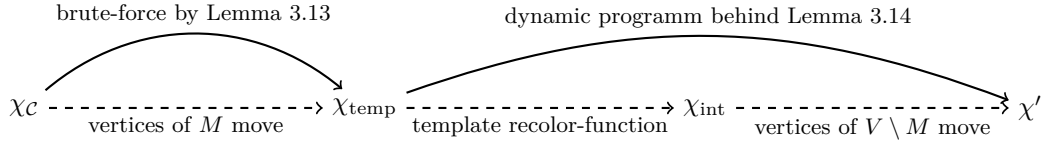
Suppose that there is an improving k -move-neighbor χ' of χ_C . Then, to find a coloring of V that improves over χ_C , it is sufficient to iterate over all colorings χ_{int} with $\text{move}(\chi_C, \chi_{\text{int}}) \leq k$ that agree with χ_C on all vertices of $V \setminus M$, and to check whether there is a coloring χ' that improves over χ_C with $\text{move}(\chi_{\text{int}}, \chi') \leq k$ such that χ_{int} and χ' agree on all vertices of M , that is, where χ_{int} is the intermediate coloring for χ' . Unfortunately, such an approach exceeds the desired running time for our algorithm since there are $n^{\mathcal{O}(k)}$ possibilities for the colorings χ_{int} due to the fact that there may be up to $\Theta(n)$ many bag colors and each vertex of M may receive any color. To obtain the desired running time, we instead only iterate over “template colorings”. Here, a coloring χ_{temp} of V is a *template coloring* if

- $\text{move}(\chi_C, \chi_{\text{temp}}) \leq k$,
- χ_{temp} agrees with χ_C on all vertices of $V \setminus M$, and
- no vertex of M receives a color of col_{Bag} under χ_{temp} .

For a template coloring χ_{temp} , let col_{Move} denote the colors of $\mathbb{N} \setminus (\text{col}_{\text{Mod}} \cup \text{col}_{\text{Bag}})$ that are used by χ_{temp} . We call the colors of col_{Move} the *moving colors* of χ_{temp} . Note that only vertices of M may receive a moving color under χ_{temp} and that there are at most k moving colors.

Figure 1 gives an overview over the considered types of colorings and they way we use them to find a coloring that improves over χ_C .

The idea behind template colorings is that a template coloring χ_{temp} may represent intermediate colorings for many colorings χ' in the following way: For a coloring χ' of V , we say that a template coloring χ_{temp} is *quasi-intermediate* for χ' if there is a “template recolor-function” $f: \mathbb{N} \rightarrow \mathbb{N}$ such that $f \circ \chi_{\text{temp}}$ is the intermediate coloring for χ' . Herein, a function $f: \mathbb{N} \rightarrow \mathbb{N}$ is a *template recolor-function* if f preserves identity on all colors of $\mathbb{N} \setminus \text{col}_{\text{Move}}$ and where $f|_{\text{col}_{\text{Move}}}$ maps each color of col_{Move} to some color of $\mathbb{N} \setminus \text{col}_{\text{Mod}}$ injectively. Essentially, this means that each of the moving colors of χ_{temp} may be identified with any bag color. Informally, this is due to the fact that each vertex that receives a moving



■ **Figure 1** An overview over the four different kinds of considered colorings. For the initial coloring χ_C and an improving coloring χ' with $\text{move}(\chi_C, \chi') \leq k$, there is a template coloring χ_{temp} and an intermediate coloring χ_{int} such that there is a template recolor-function between χ_{temp} and χ_{int} . To find a coloring at least as good as χ' , we first brute-force all possible choices of the template coloring χ_{temp} and afterwards search for the best coloring for which χ_{temp} is quasi-intermediate. This is done by a dynamic program that simultaneously finds the best template recolor-function and the best way to distribute the bag vertices by using at most k moves.

color under χ_{temp} already changed its color and we may move all vertices of that moving color together to any bag color while preserving the move-distance to χ_C . Note that, for each coloring χ' of V with $\text{move}(\chi_C, \chi') \leq k$, there is a template coloring χ_{temp} which is quasi-intermediate for χ' . In contrast to intermediate colorings, we can show that we can enumerate a maximal set \mathcal{X} of pairwise non-isomorphic template colorings in the desired running time.

► **Lemma 3.13.** *One can compute a maximal set \mathcal{X} of pairwise non-isomorphic template colorings in time $(|\text{col}_{\text{Mod}}| + k)^k \cdot |M|^k \cdot n^{\mathcal{O}(1)}$.*

Proof. Recall that for each template coloring χ_{temp} , $\text{move}(\chi_C, \chi_{\text{temp}}) \leq k$. Moreover, by the definition of a template coloring, $\text{Move}(\chi_C, \chi_{\text{temp}}) \subseteq M$. Hence, to obtain a maximal set \mathcal{X} of pairwise non-isomorphic template colorings, consider all possible subsets of M of size at most k and consider all possible ways of assigning colors of $\text{col}_{\text{Mod}} \cup X$ to these at most k vertices, where X is an arbitrary set of k colors from $\mathbb{N} \setminus (\text{col}_{\text{Mod}} \cup \text{col}_{\text{Bag}})$. Note that this can be done in the stated running time. ◀

Lemma 3.13 implies that, in order to find a coloring χ^* of V that improves over χ_C (provided that there is such a coloring in the k -move-neighborhood of χ_C), it suffices to do the following: For each template coloring χ_{temp} in a maximal set of pairwise non-isomorphic template colorings, find the best coloring χ' in the k -move-neighborhood of χ_C such that χ_{temp} is quasi-intermediate for χ' .

The latter task can be solved in $2^{\mathcal{O}(|\text{col}_{\text{Mod}}|+k)} \cdot n^{\mathcal{O}(1)}$ time.

► **Lemma 3.14** (\star). *Let χ_{temp} be a template coloring. In $2^{\mathcal{O}(|\text{col}_{\text{Mod}}|+k)} \cdot n^{\mathcal{O}(1)}$ time, one can find a coloring of V that improves over χ_C or correctly output that the k -move-neighborhood of χ_C does not contain a coloring χ' such that χ' improves over χ_C and where χ_{temp} is quasi-intermediate for χ' .*

We now conclude our permissive algorithm for LS-CLUSTER EDITING. As we can switch between clusterings and cluster colorings in polynomial time, this also proves Theorem 3.9.

► **Theorem 3.15.** *Let $G = (V, E)$ be a graph, let χ_C be a coloring of V , and let $k \in \mathbb{N}$. In $2^{\mathcal{O}(k)} \cdot k^k \cdot \text{cvd}^{3k} \cdot n^{\mathcal{O}(1)}$ time, one can find a coloring χ^* that improves over χ or correctly output that there is no coloring in the k -move-neighborhood of χ_C that improves over χ_C .*

Proof. First, we 2-approximate a cluster modulator M for G in polynomial time [1]. Let \mathcal{B} be the bags of $G[V \setminus M]$, let col_{Mod} and col_{Bag} be the modulator colors and bag colors of $\chi_{\mathcal{C}}$, respectively. If the condition of Observation 3.10 or Observation 3.11 applies, then we find a coloring χ^* of V that improves over χ in polynomial time. Hence, assume in the following that this is not the case.

As mentioned at the beginning of this subsection, we perform an initial branching step to ensure that the coloring $\chi_{\mathcal{C}}$ uses at most $2k$ modulator colors. Let χ' be the best coloring in the k -move-neighborhood of $\chi_{\mathcal{C}}$. Assume that χ' improves over $\chi_{\mathcal{C}}$. Since $\text{move}(\chi_{\mathcal{C}}, \chi') \leq k$, Observation 2.1 implies that there is a subset $S \subseteq \text{col}_{\text{Mod}}$ of size at least $|\text{col}_{\text{Mod}}| - 2k$ such that for each modulator color $\alpha \in S$, $\chi_{\mathcal{C}}^{-1}(\alpha) = \chi'^{-1}(\alpha)$. Hence, to find a coloring that improves over $\chi_{\mathcal{C}}$ it is sufficient to branch into all subsets $S \subseteq \text{col}_{\text{Mod}}$ of size at least $|\text{col}_{\text{Mod}}| - 2k$ and ask for a coloring $\hat{\chi}'$ of $\hat{V} := V \setminus (\cup_{\alpha \in S} \chi_{\mathcal{C}}^{-1}(\alpha))$ that improves over $\hat{\chi} := \chi_{\mathcal{C}}|_{\hat{V}}$ with respect to the subgraph $G[\hat{V}]$. Note that this branching takes $|\text{col}_{\text{Mod}}|^{2k} \cdot n^{\mathcal{O}(1)} \leq |M|^{2k} \cdot n^{\mathcal{O}(1)}$ time.

Hence, in the following, we assume that col_{Mod} has size at most $2k$. Next, we iterate over a maximal set \mathcal{X} of pairwise non-isomorphic template colorings and compute for each such template coloring χ_{temp} a coloring χ^* which is at least as good as a best coloring χ'' in the k -move-neighborhood of $\chi_{\mathcal{C}}$ where χ_{temp} is quasi-intermediate for χ'' . The latter task can be done in $2^{\mathcal{O}(|\text{col}_{\text{Mod}}| + k)} \cdot n^{\mathcal{O}(1)}$ time due to Lemma 3.14 for each template coloring $\chi_{\text{temp}} \in \mathcal{X}$. Due to Lemma 3.13, \mathcal{X} can be computed in $(|\text{col}_{\text{Mod}}| + k)^k \cdot |M|^k \cdot n^{\mathcal{O}(1)}$ time and has size $(|\text{col}_{\text{Mod}}| + k)^k \cdot |M|^k \cdot n^{\mathcal{O}(1)}$.

This algorithm is correct, since there is a template coloring χ'_{temp} such that χ'_{temp} is quasi-intermediate for χ' . Thus, in this way, we will find a coloring at least as good as χ' .

The whole algorithm runs in $|M|^{2k} \cdot (|\text{col}_{\text{Mod}}| + k)^k \cdot |M|^k \cdot 2^{\mathcal{O}(|\text{col}_{\text{Mod}}| + k)} \cdot n^{\mathcal{O}(1)}$ time. Since we ensured with the initial branching, that col_{Mod} has size at most $2k$, this results in a running time of $2^{\mathcal{O}(k)} \cdot k^k \cdot |M|^{3k} \cdot n^{\mathcal{O}(1)}$ time. Finally, since M is a 2-approximated cluster modulator, $|M| \leq 2 \cdot \text{cvd}(G)$. Hence, we obtain the stated running time of $2^{\mathcal{O}(k)} \cdot k^k \cdot \text{cvd}(G)^{3k} \cdot n^{\mathcal{O}(1)}$ time. \blacktriangleleft

4 Lower Bounds

In this section we present several hardness results for LS-CLUSTER DELETION and LS-CLUSTER EDITING. We obtain our hardness results for LS-CLUSTER EDITING by reductions from restricted instances of DENSEST- k -SUBGRAPH, which is defined as follows

DENSEST- k -SUBGRAPH

Input: A graph $G = (V, E)$, integers k and d .

Question: Is there a subset $S \subseteq V$ of size exactly k such that $|E(S)| \geq \binom{k}{2} - d$?

Hence, we first show that DENSEST- k -SUBGRAPH provides these hardness results on the desired restricted instances.

► **Theorem 4.1** (\star). *Even if $d = \frac{k-1}{2}$, DENSEST- k -SUBGRAPH is W[1]-hard when parameterized by both k and the degeneracy of G and cannot be solved in $f(k) \cdot n^{\mathcal{O}(k)}$ time for any computable function f , unless the ETH fails. This holds even on instances where $|E(S)| < \binom{k-1}{2} - \frac{k-1}{4}$ for each vertex set S of size $k-1$.*

Based on these hardness results for DENSEST- k -SUBGRAPH, we are now able to analyze the parameterized complexity of LS-CLUSTER EDITING for the parameter combination of k plus the size of the largest cluster of the initial clustering \mathcal{C} .

► **Theorem 4.2.** *LS-CLUSTER EDITING is $W[1]$ -hard when parameterized by $k + \ell + \text{degen}$, where $\ell := \max_{C \in \mathcal{C}} |C|$ and degen denotes the degeneracy of G . Moreover, unless the ETH fails, there is no computable function f such that LS-CLUSTER EDITING can be solved in $f(k + \ell) \cdot n^{o(k+\ell)}$ time.*

Proof. We present a parameterized reduction from DENSEST- k -SUBGRAPH with the restrictions listed in Theorem 4.1. Let $I = (G = (V, E), k, d)$ be an instance of DENSEST- k -SUBGRAPH with $d = \frac{k-1}{2}$ such that $|E(S)| < \binom{k-1}{2} - \frac{k-1}{4}$ for each vertex set S of size $k-1$. We define an instance $I' := (G' := (V', E'), k, \mathcal{C})$ of LS-CLUSTER EDITING with $\max_{C \in \mathcal{C}} |C| \in \mathcal{O}(k)$ as follows: We initialize G' as G and add for each vertex $v \in V$ a set K_v of $7k + \frac{k-3}{2}$ vertices to G' such that $\{v\} \cup K_v$ is a clique in G' . Additionally, we add a clique K^* of size $7k$ to G' and add edges to G' , such that each vertex of V is adjacent to each vertex of K^* . Finally, we set $\mathcal{C} := \{K^*\} \cup \{\{v\} \cup K_v \mid v \in V\}$. Note that by definition of \mathcal{C} , each cluster $C \in \mathcal{C}$ is a clique in G' . The correctness proof is based on the following claim.

▷ **Claim 4.3.** Let $\mathcal{C}' := \{K^* \cup S\} \cup \{\{K_v\} \mid v \in S\} \cup \{\{v\} \cup K_v \mid v \in V \setminus S\}$ be a clustering of G' for some vertex set $S \subseteq V$. The improvement of \mathcal{C}' over \mathcal{C} is $2 \cdot |E_G(S)| - \binom{|S|}{2} - |S| \cdot \frac{k-3}{2}$.

Proof. We only have to consider the edges incident with at least one vertex of S in $E(\mathcal{C})$ and $E(\mathcal{C}')$. Let $F := E(\mathcal{C})$ and let $F' := E(\mathcal{C}')$. Note that the symmetric difference between F and F' are the edges of $F \oplus F' = \{\{v, x\} \mid v \in S, x \in K_v \cup K^*\} \cup \binom{S}{2}$. More precisely, $F \setminus F' = \{\{v, x\} \mid v \in S, x \in K_v\}$ and $F' \setminus F = \{\{v, x\} \mid v \in S, x \in K^*\} \cup \binom{S}{2}$. By construction, all edges of $F \oplus F'$ exist in G' , except for the edges of $\binom{S}{2} \setminus E_{G'}(S) = \binom{S}{2} \setminus E_G(S)$. Hence, the improvement of \mathcal{C}' over \mathcal{C} is

$$\sum_{v \in S} (|K^*| - |K_v|) + |E_{G'}(S)| - \left(\binom{|S|}{2} - |E_{G'}(S)| \right) = 2 \cdot |E_G(S)| - \binom{|S|}{2} - |S| \cdot \frac{k-3}{2}. \quad \triangleleft$$

Next, we show that I is a yes-instance of DENSEST- k -SUBGRAPH if and only if I' is a yes-instance of LS-CLUSTER EDITING.

(\Rightarrow) Let S be a set of size k in G such that $|E_G(S)| \geq \binom{k}{2} - d$. We set $\mathcal{C}' := \{K^* \cup S\} \cup \{\{K_v\} \mid v \in S\} \cup \{\{v\} \cup K_v \mid v \in V \setminus S\}$. Note that $\text{move}(\mathcal{C}, \mathcal{C}') = k$. We show that \mathcal{C}' is improving over \mathcal{C} . Due to Claim 4.3 and since $|E_G(S)| \geq \binom{k}{2} - d$ and $d = \frac{k-1}{2}$, the improvement of \mathcal{C}' over \mathcal{C} is at least

$$\binom{k}{2} - 2d - k \cdot \frac{k-3}{2} = \binom{k}{2} - k + 1 - k \cdot \frac{k-3}{2} = \binom{k}{2} - k \cdot \frac{k-1}{2} + 1 = 1.$$

Hence, I' is a yes-instance of LS-CLUSTER EDITING.

(\Leftarrow) Suppose that I' is a yes-instance of LS-CLUSTER EDITING. Let \mathcal{C}' be the best clustering for G' with $\text{move}(\mathcal{C}, \mathcal{C}') \leq k$. Since I' is a yes-instance of LS-CLUSTER EDITING, $\mathcal{C}' \neq \mathcal{C}$. We make some observations about the potential moves between \mathcal{C} and \mathcal{C}' . The goal is to show that there is a set $S \subseteq V$ of size at most k such that $\mathcal{C}' = \{K^* \cup S\} \cup \{K_v \mid v \in S\} \cup \{\{v\} \cup K_v \mid v \in V \setminus S\}$. To this end, we show some intermediate results.

First, we show that for each vertex $v \in V$ there is a cluster $C \in \mathcal{C}'$ with $K_v \subseteq C$. Suppose that this is not the case. Hence, there is a vertex $v \in V$ and at least two clusters C_1 and C_2 in \mathcal{C}' containing vertices of K_v . Since K_v has size more than k , at least one vertex of K_v is not moved. Assume without loss of generality that C_1 contains this vertex. Hence, each vertex of $C_2 \cap K_v$ moved to C_2 . Thus, $C_2 \cap K_v$ contains at most k vertices. Let x be an arbitrary vertex of $C_2 \cap K_v$. Since K_v has size more than $4k$, C_1 contains at least $3k$ vertices of K_v and at most k vertices of $V \setminus K_v$. By definition of K_v , the closed neighborhood of x is

exactly $K_v \cup \{v\}$. Hence, x has at most k neighbors in C_2 , at least $3k$ neighbors in C_1 and at most k non-neighbors in C_1 . Consequently, not moving x to C_2 yields a better clustering. Since \mathcal{C}' is the best clustering with $\text{move}(\mathcal{C}, \mathcal{C}') \leq k$, this is not possible.

Next, we show that there is a cluster C in \mathcal{C}' with $K^* \subseteq C$. Suppose that this is not the case. Hence, there are at least two clusters C_1 and C_2 in \mathcal{C}' containing vertices of K^* . Since K^* has size more than k , at least one vertex of K^* is not moved. Assume without loss of generality that C_1 contains this vertex. Hence, each vertex of $C_2 \cap K^*$ moved to C_2 . Thus, $C_2 \cap K^*$ contains at most k vertices. Let x be an arbitrary vertex of $C_2 \cap K^*$. Since K^* has size more than $4k$, C_1 contains at least $3k$ vertices of K^* and at most k vertices of $V' \setminus K^*$. By definition of K^* , the closed neighborhood of x is exactly $K^* \cup V$. Hence, since each cluster in \mathcal{C} contains at most one vertex of V , x has at most $k+1$ neighbors in C_2 , at least $3k$ neighbors in C_1 and at most k non-neighbors in C_1 . Consequently, not moving x to C_2 yields a better clustering. Since \mathcal{C}' is the best clustering with $\text{move}(\mathcal{C}, \mathcal{C}') \leq k$, this is not possible.

The above implies that only vertices of V moved to obtain \mathcal{C}' from \mathcal{C} .

Next, we show that for each vertex $v \in V$, the cluster C of \mathcal{C}' that contains v either contains all vertices of K^* or all vertices of K_v . Suppose that this is not the case and let v be a vertex of V such that the cluster $C \in \mathcal{C}'$ with $v \in C$ is not a superset of K^* and not a superset of K_v . Since v is only adjacent to at most one vertex in each cluster of $\mathcal{C} \setminus \{K^*, K_v \cup \{v\}\}$, v has at most k neighbors in C . Let K'_v be the cluster of \mathcal{C}' containing all vertices of K_v . Since K_v has size more than $3k$, K'_v contains at most k non-neighbors of v and at least $3k$ neighbors of v . Hence, not moving v from K'_v to C yields a better clustering. Since \mathcal{C}' is the best clustering with $\text{move}(\mathcal{C}, \mathcal{C}') \leq k$, this is not possible.

Concluding, there is a nonempty vertex set S of size at most k such that $\mathcal{C}' = \{K^* \cup S\} \cup \{K_v \mid v \in S\} \cup \{\{v\} \cup K_v \mid v \in V \setminus S\}$. More precisely, the vertices of S are exactly the vertices that are moved to obtain \mathcal{C}' from \mathcal{C} .

It remains to show that S has size k and that $E_G(S) = E_{\mathcal{C}'}(S)$ contains at least $\binom{k}{2} - d$ edges. Due to Claim 4.3 and since \mathcal{C}' is improving over \mathcal{C} , $2 \cdot |E_G(S)| - \binom{|S|}{2} - |S| \cdot \frac{k-3}{2} \geq 1$.

▷ **Claim 4.4.** S has size k .

Proof. If $|S| < k - 1$, then $2|E_G(S)| \leq 2 \cdot \binom{|S|}{2} < |S| \cdot \frac{k-3}{2} + \binom{|S|}{2} + 1$. Since $2|E_G(S)| \geq |S| \cdot \frac{k-3}{2} + \binom{|S|}{2} + 1$, we conclude $|S| \geq k - 1$.

Assume towards a contradiction that S has size exactly $k - 1$. By assumption, $|E_G(S)| < \binom{k-1}{2} - \frac{k-1}{4} = (k-1) \cdot (\frac{k-2}{2} - \frac{1}{4}) = (k-1) \cdot \frac{2k-5}{4}$. Hence, $(k-1) \cdot \frac{2k-5}{2} > 2 \cdot |E_S(G)| \geq |S| \cdot \frac{k-3}{2} + \binom{|S|}{2} + 1 = (k-1) \cdot \frac{k-3}{2} + \binom{k-1}{2} + 1 = (k-1) \cdot \frac{2k-5}{2} + 1$, a contradiction.

Consequently, S contains exactly k vertices since to obtain \mathcal{C}' from \mathcal{C} , each vertex of S moved and $\text{move}(\mathcal{C}, \mathcal{C}') \leq k$. ◁

It remains to show that $|E_G(S)| \geq \binom{k}{2} - d$. By Claim 4.3, $2|E_G(S)| \geq |S| \cdot \frac{k-3}{2} + \binom{|S|}{2} + 1 = \frac{2k^2-4k}{2} + 1 = k^2 - k - (k-1)$. Thus $|E_G(S)| \geq \frac{k^2-k}{2} - \frac{k-1}{2} = \binom{k}{2} - d$. Hence, I is a yes-instance of DENSEST- k -SUBGRAPH.

Parameter bounds. Recall that the size $\ell := \max_{C \in \mathcal{C}} |C|$ of the largest cluster in \mathcal{C} is $\mathcal{O}(k)$.

Due to Theorem 4.1, DENSEST- k -SUBGRAPH cannot be solved in $f(k) \cdot n^{\mathcal{O}(k)}$ time for any computable function f , unless the ETH fails. This implies that LS-CLUSTER DELETION cannot be solved in $f(k+\ell) \cdot |V'|^{\mathcal{O}(k+\ell)}$ time for any computable function f , unless the ETH fails, since $|V'| \in n^{\mathcal{O}(1)}$.

Next, we analyze the degeneracy degen' of G' . Let degen denote the degeneracy of G . Since each vertex of K_v for some vertex $v \in V$ has only neighbors in $K_v \cup \{v\}$, each such vertex

has degree $\mathcal{O}(k)$ in G' . Furthermore, each vertex of V has only $|K_v| + |K^*| \in \mathcal{O}(k)$ additional neighbors in G' . Hence, the degeneracy of G' is $\mathcal{O}(k + \text{degen})$. Consequently, LS-CLUSTER EDITING is W[1]-hard when parameterized by $k + \ell + \text{degen}'$, since due to Theorem 4.1, DENSEST- k -SUBGRAPH is W[1]-hard when parameterized by $k + \text{degen}$. \blacktriangleleft

Next, we show that even when the initial clustering consists only of two clusters, LS-CLUSTER EDITING remains W[1]-hard when parameterized by k .

► **Theorem 4.5** (\star). *Even when the initial clustering consists of only two clusters, LS-CLUSTER EDITING is W[1]-hard when parameterized by k and cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function f , unless the ETH fails.*

Hence, LS-CLUSTER EDITING is W[1]-hard when parameterized by the arguably most natural parameter combinations $k + \max_{C \in \mathcal{C}} |C|$ and $k + |\mathcal{C}|$.

Finally, we present our hardness results for LS-CLUSTER DELETION. As we show, these hardness results also hold for the permissive version of LS-CLUSTER DELETION.

► **Theorem 4.6.** *LS-CLUSTER DELETION*

- *is W[1]-hard when parameterized by $k + \ell$, where $\ell := \max_{C \in \mathcal{C}} |C|$ denotes the size of the largest cluster in \mathcal{C} ,*
- *cannot be solved in $f(k + \ell) \cdot n^{o(k + \ell)}$ time for any computable function f , unless the ETH fails, and*
- *does not admit a polynomial kernel when parameterized by $k + \text{vc}$, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$, where vc denotes the vertex cover number of G .*

All of this holds even if there is an optimal clustering \mathcal{C}^ with $E(\mathcal{C}^*) \subseteq E$ in the k -move-neighborhood of \mathcal{C} .*

Proof. We present a polynomial time reduction from MULTICOLORED CLIQUE, which is W[1]-hard when parameterized by the size of the sought clique [14].

MULTICOLORED CLIQUE

Input: A graph $G = (V, E)$ and an integer k such that G is k -partite.

Question: Is there a clique of size k in G ?

Let $I := (G = (V, E), k)$ be an instance of MULTICOLORED CLIQUE and let V_k be the largest part of the k -partition (V_1, \dots, V_k) of G . We define an instance $I' := (G' := (V', E'), k', \mathcal{C})$ of LS-CLUSTER DELETION as follows: Initialize G' as a copy of G . Then for each $i \in [1, k - 1]$, do the following:

- add a vertex x_i to G' and
- for each vertex $v \in V_i$, add a vertex set K_v of size $z := 2k$ to G' and turn $K_v \cup \{v\}$ and $K_v \cup \{x_i\}$ into cliques in G' .

Let $X := \{x_i \mid 1 \leq i \leq k - 1\}$. Finally, we add two additional adjacent vertices a and b to G' and make $X \cup \{a\}$ a clique in G' . This completes the construction of G' . We complete the construction of I' by setting $k' := 2k - 1$ and

$$\mathcal{C} := \{X \cup \{a\}, \{b\}\} \cup \bigcup_{v \in V \setminus V_k} \{K_v \cup \{v\}\} \cup \bigcup_{v \in V_k} \{\{v\}\}.$$

Note that by construction $E(\mathcal{C}) \subseteq E'$ and $|E(\mathcal{C})| = |V \setminus V_k| \cdot \binom{z+1}{2} + \binom{k}{2}$.

Next, we show that there is a clique of size k in G if and only if there is a clustering \mathcal{C}' for G' that improves over \mathcal{C} . We further show that each clustering \mathcal{C}' for G' that improves over \mathcal{C} is a k' -move-neighbor of \mathcal{C} .

(\Rightarrow) Let S be a clique of size k in G . Since (V_1, \dots, V_k) is a k -partition of G , for each $i \in [1, k]$, S contains exactly one vertex of V_i . For each $i \in [1, k]$, let v_i be that unique vertex of $V_i \cap S$. We set

$$\mathcal{C}' := \{S, \{a, b\}\} \cup \bigcup_{i \in [1, k-1]} \{K_{v_i} \cup \{x_i\}\} \cup \bigcup_{v \in V \setminus (V_k \cup S)} \{K_v \cup \{v\}\} \cup \bigcup_{v \in V_k \setminus \{v_k\}} \{\{v\}\}.$$

Note that by construction, $E(\mathcal{C}') \subseteq E'$ and by definition of \mathcal{C}' , $|E(\mathcal{C}')| = \binom{k}{2} + |X| \cdot \binom{z+1}{2} + |V \setminus (V_k \cup S)| \cdot \binom{z+1}{2} + 1 = |V \setminus V_k| \cdot \binom{z+1}{2} + \binom{k}{2} + 1 = |E(\mathcal{C})| + 1$. Hence, \mathcal{C}' is improving over \mathcal{C} . Moreover, by construction of \mathcal{C}' , \mathcal{C}' and \mathcal{C} are k' -move-neighbors.

(\Leftarrow) Let \mathcal{C}' be a best clustering for G' with $E(\mathcal{C}') \subseteq E'$ and suppose that \mathcal{C}' improves over \mathcal{C} . Before we show that there is a clique of size k in G , we prove some properties of the clustering \mathcal{C}' .

First, we show that for each vertex $v \in V \setminus V_k$, there is a cluster C'_v in \mathcal{C}' with $K_v \subseteq C'_v$. Suppose that there are at least two clusters C'_v and C''_v in \mathcal{C}' with $K_v \cap C'_v \neq \emptyset$ and $K_v \cap C''_v \neq \emptyset$ and suppose that $|C'_v| \geq |C''_v|$. Let v' be an arbitrary vertex of $C''_v \cap K_v$. Recall that by construction of G' , v' has the same closed neighborhood as any other vertex of K_v . Since $E(\mathcal{C}') \subseteq E'$, each vertex of C'_v is part of the closed neighborhood of v' . Hence, the clustering $\mathcal{C}'' := (\mathcal{C} \setminus \{C'_v, C''_v\}) \cup \{C'_v \cup \{v'\}, C''_v \setminus \{v'\}\}$ fulfills $E(\mathcal{C}'') \subseteq E'$ and improves over \mathcal{C}' . Since \mathcal{C}' is a best clustering for G' with $E(\mathcal{C}') \subseteq E'$, no such two clusters exist and thus for each vertex $v \in V \setminus V_k$, there is a cluster C'_v in \mathcal{C}' with $K_v \subseteq C'_v$.

Next, we show that for each $i \in [1, k-1]$ and each vertex $v \in V_i$, the cluster C'_v is either $K_v \cup \{v\}$ or $K_v \cup \{x_i\}$. Suppose that this is not the case and let $i \in [1, k-1]$ and let v be a vertex of V_i such that $C'_v \not\subseteq \{K_v \cup \{v\}, K_v \cup \{x_i\}\}$. Note that this implies that $C'_v = K_v$. Furthermore, let C'' be the cluster of \mathcal{C}' that contains v . Since $E(\mathcal{C}') \subseteq E'$ and v has only neighbors in $V \cup K_v$, the cluster C'' is a clique in G . Moreover, since G is k -partite, this implies that C'' has size at most k . Hence, the clustering $\mathcal{C}'' := (\mathcal{C} \setminus \{C'_v, C''\}) \cup \{K_v \cup \{v\}, C'' \setminus \{v\}\}$ fulfills $E(\mathcal{C}'') \subseteq E'$ and improves over \mathcal{C}' . Since \mathcal{C}' is a best clustering for G' with $E(\mathcal{C}') \subseteq E'$, this implies that for each $i \in [1, k-1]$ and each vertex $v \in V_i$, the cluster C'_v is either $K_v \cup \{v\}$ or $K_v \cup \{x_i\}$.

Note that this implies that for each $i \in [1, k-1]$, there is at most one vertex $v_i \in V_i$ for which $C'_{v_i} \neq K_{v_i} \cup \{v_i\}$. Intuitively, if such a vertex v_i moves out of its cluster from \mathcal{C} , then the vertex x_i has to move into the original cluster of v_i .

Let $S' \subseteq V'$ be a minimal set of vertices that have to move to obtain \mathcal{C}' from \mathcal{C} . Moreover, let $S := S' \cap (V \setminus V_k)$. By the above, for each $i \in [1, k-1]$, S contains at most one vertex of V_i . Recall that each vertex of V_k has neighbors only in $V \setminus V_k$ and can thus only be in a cluster of \mathcal{C}' with a (potentially empty) subset of vertices of S . Hence, S' contains no vertex of V_k . In the following, we show that there is a vertex $v_k \in V_k$ such that $S \cup \{v_k\}$ is a clique of size k in G .

To this end, we analyze the number of edges in the clusters $\mathcal{C}_S \subseteq \mathcal{C}'$ that contain vertices of $S \cup V_k$ and have size at least two and the number of edges in the clusters $\mathcal{C}_X \subseteq \mathcal{C}'$ that have size at least two and contain only vertices of $X \cup \{a, b\}$. By the above, each cluster C of \mathcal{C}_S contains only vertices of $S \cup V_k$ and C contains at most one vertex of V_k , since V_k is an independent set in G' . Note that this implies that each cluster of \mathcal{C}_S contains at least one vertex of S . Furthermore, note that $E(\mathcal{C}') = \bigcup_{v \in V \setminus V_k} \binom{C'_v}{2} \cup E(\mathcal{C}_S) \cup E(\mathcal{C}_X)$. Since for each vertex $v \in V \setminus V_k$, C'_v has size $z+1$, $|E(\mathcal{C}')| = |V \setminus V_k| \cdot \binom{z+1}{2} + |E(\mathcal{C}_S)| + |E(\mathcal{C}_X)|$. Moreover, since \mathcal{C}' is improving over \mathcal{C} , $|E(\mathcal{C}')| - |E(\mathcal{C})| = |E(\mathcal{C}_S)| + |E(\mathcal{C}_X)| - \binom{k}{2} \geq 1$. In the following, we show that $|E(\mathcal{C}_S)| + |E(\mathcal{C}_X)| \leq \binom{k}{2} + 1$ and that $|E(\mathcal{C}_S)| + |E(\mathcal{C}_X)| \leq \binom{k}{2}$ if there is no vertex $v_k \in V_k$ such that $S \cup \{v_k\}$ is a clique of size k in G . To this end, we

analyze the size of $E(\mathcal{C}_S)$ and the size of $E(\mathcal{C}_X)$ separately.

First, we show that if \mathcal{C}_S has size at least two, then $|E(\mathcal{C}_S)| < \binom{|S|+1}{2}$. This follows inductively by the fact that each cluster of \mathcal{C}_S has size at least two and contains at most one vertex of V_k , and for each $i \geq 2$ and each $j \geq 2$, $\binom{i}{2} + \binom{j}{2} < \binom{i+j-1}{2}$. Hence, $E(\mathcal{C}_S)$ has size at least $\binom{|S|+1}{2}$ if and only if there is a vertex $v_k \in V_k$ such that $\mathcal{C}_S = \{S \cup \{v_k\}\}$. Moreover, this implies that $|E(\mathcal{C}_S)| \leq \binom{|S|+1}{2}$.

Second, we analyze the size of $E(\mathcal{C}_X)$. Since for each $i \in [1, k-1]$, if there is a vertex $v_i \in V_i \cap S$, then the vertex x_i moves to the cluster containing all vertices of K_{v_i} , that is, x_i is not contained in any cluster of \mathcal{C}_X . Hence, at most $k-1-|S|$ vertices of X are contained in clusters of \mathcal{C}_X . Since b is adjacent only to a , this implies that $|E(\mathcal{C}_X)| \leq \binom{k-|S|}{2} + 1$.

Finally, we show that for some vertex $v_k \in V_k$, $S \cup \{v_k\}$ is a clique of size k in G . Assume that $|S| < k-1$. Hence, by the above $|E(\mathcal{C}_S)| + |E(\mathcal{C}_X)| \leq \binom{|S|+1}{2} + \binom{k-|S|}{2} + 1 < \binom{k}{2} + 1$ and thus $|E(\mathcal{C}_S)| + |E(\mathcal{C}_X)| \leq \binom{k}{2}$. Since $|E(\mathcal{C}')| > |E(\mathcal{C})|$, this is not possible.

Consequently, $|S| = k-1$. Hence, by the above $|E(\mathcal{C}_S)| + |E(\mathcal{C}_X)| \leq \binom{|S|+1}{2} + \binom{k-|S|}{2} + 1 = \binom{k}{2} + 1$. Hence, $|E(\mathcal{C}')| \leq |E(\mathcal{C})| + 1$. Since \mathcal{C}' improves over \mathcal{C} , $|E(\mathcal{C}_S)| + |E(\mathcal{C}_X)| = \binom{k}{2} + 1$. As shown before, $|E(\mathcal{C}_S)| + |E(\mathcal{C}_X)| = \binom{k}{2} + 1$ only holds if \mathcal{C}_S consists of a single cluster $\mathcal{C}' := S \cup \{v_k\}$ for some vertex $v_k \in V_k$. Hence, there is a clique of size k in G and I is a yes-instance of MULTICOLORED CLIQUE.

Moreover, note that this implies that \mathcal{C}' is a k' -move-neighbor of \mathcal{C} , since only the $k-1$ vertices of S , the $k-1$ vertices of X , and either a or b changed their cluster.

Parameter bounds. Let $\ell := \max_{\mathcal{C} \in \mathcal{C}} |\mathcal{C}|$. Recall that since $z = 2k$, $\ell = 2k+1$. Since MULTICOLORED CLIQUE is W[1]-hard when parameterized by k and cannot be solved in $f(k) \cdot n^{o(k)}$ time for any computable function f , unless the ETH fails [14], this implies that LS-CLUSTER DELETION is W[1]-hard when parameterized by $k' + \ell$ and cannot be solved in $f(k' + \ell) \cdot |V'|^{o(k' + \ell)}$ time for any computable function f , unless the ETH fails, since $|V'| \in n^{O(k)}$. Moreover, note that $V' \setminus V_k$ is a vertex cover of G' of size $|V \setminus V_k| \cdot (2k+1) + k+1$. Since MULTICOLORED CLIQUE does not admit a polynomial kernel when parameterized by $k + |V \setminus V_k|$, unless $\text{NP} \subseteq \text{coNP/poly}$ [24], this implies that LS-CLUSTER DELETION does not admit a polynomial kernel when parameterized by $k' + \text{vc}(G')$ unless $\text{NP} \subseteq \text{coNP/poly}$. ◀

Note that due to the last restriction, the permissive version of LS-CLUSTER DELETION shares the same W[1]-hardness and the same ETH-based lower bound, meaning that also for permissive LS-CLUSTER DELETION, the algorithms in Section 3 are essentially optimal.

5 Conclusion

We analyzed the parameterized complexity for LS-CLUSTER EDITING and LS-CLUSTER DELETION, leaving some open questions for future work: First, what is the complexity of LS-CLUSTER DELETION with respect to the combined parameter number $|\mathcal{C}|$ of clusters and k ? Second, can we show lower bounds for the permissive variant of LS-CLUSTER EDITING? Finally, can some of the algorithmic ideas of our work be used to improve the local-search based heuristics for CLUSTER DELETION or CLUSTER EDITING?

References

- 1 Manuel Aprile, Matthew Drescher, Samuel Fiorini, and Tony Huynh. A tight approximation algorithm for the cluster vertex deletion problem. *Math. Program.*, 197(2):1069–1091, 2023. doi:10.1007/s10107-021-01744-w.

- 2 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56:89–113, 2004. doi:10.1023/B:MACH.0000033116.57574.95.
- 3 Valentin Bartier, Gabriel Bathie, Nicolas Bousquet, Marc Heinrich, Théo Pierron, and Ulysse Prieto. PACE solver description: μ solver - heuristic track. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *LIPICs*, pages 33:1–33:3. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.IPEC.2021.33.
- 4 Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3-4):281–297, 1999. doi:10.1089/106652799318274.
- 5 Daniel Berend and Tamir Tassa. Improved bounds on bell numbers and on moments of sums of random variables. *Probability and Mathematical Statistics*, 30(2):185–205, 2010.
- 6 René van Bevern, Vincent Froese, and Christian Komusiewicz. Parameterizing edge modification problems above lower bounds. *Theory of Computing Systems*, 62(3):739–770, 2018. doi:10.1007/s00224-016-9746-5.
- 7 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 2021)*, volume 214 of *LIPICs*, pages 31:1–31:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.IPEC.2021.31.
- 8 Sebastian Böcker. A golden ratio parameterized algorithm for cluster editing. *Journal of Discrete Algorithms*, 16:79–89, 2012. doi:10.1016/j.jda.2012.04.005.
- 9 Édouard Bonnet, Yoichi Iwata, Bart M. P. Jansen, and Lukasz Kowalik. Fine-grained complexity of k -OPT in bounded-degree graphs for solving TSP. In *Proceedings of the 27th Annual European Symposium on Algorithms (ESA '19)*, volume 144 of *LIPICs*, pages 23:1–23:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- 10 Yixin Cao and Jianer Chen. Cluster Editing: Kernelization based on edge cuts. *Algorithmica*, 64(1):152–169, 2012.
- 11 Yixin Cao and Yuping Ke. Improved Kernels for Edge Modification Problems. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *Leibniz International Proceedings in Informatics (LIPICs)*, pages 13:1–13:14. Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.IPEC.2021.13.
- 12 Shuchi Chawla, Konstantin Makarychev, Tselil Schramm, and Grigory Yaroslavtsev. Near optimal LP rounding algorithm for correlation clustering on complete and complete k -partite graphs. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC '15)*, pages 219–228. ACM, 2015. doi:10.1145/2746539.2746604.
- 13 Jianer Chen and Jie Meng. A $2k$ kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211–220, 2012.
- 14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 15 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.
- 16 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- 17 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshantov, 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.
- 18 Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Peter Shaw. Efficient parameterized preprocessing for Cluster Editing. In *Proceedings of the 16th International*


- Symposium on Fundamentals of Computation Theory (FCT '07)*, volume 4639 of *LNCS*, pages 312–321. Springer, 2007. doi:10.1007/978-3-540-74240-1_27.
- 19 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 2023, 19th-25th August 2023, Macao, SAR, China*, pages 5586–5594. ijcai.org, 2023. doi:10.24963/ijcai.2023/620.
 - 20 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.
 - 21 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.
 - 22 Martin Josef Geiger. PACE solver description: A simplified threshold accepting approach for the cluster editing problem. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *LIPICs*, pages 34:1–34:2. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.IPEC.2021.34.
 - 23 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, 2005.
 - 24 Niels Grüttemeier and Christian Komusiewicz. On the relation of strong triadic closure and cluster deletion. *Algorithmica*, 82(4):853–880, 2020. doi:10.1007/s00453-019-00617-1.
 - 25 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>. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/17463>.
 - 26 Jiong Guo. A more effective linear kernelization for cluster editing. *Theoretical Computer Science*, 410(8-10):718–726, 2009. doi:10.1016/j.tcs.2008.10.021.
 - 27 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.
 - 28 Jiong Guo, Danny Hermelin, and Christian Komusiewicz. Local search for string problems: Brute-force is essentially optimal. *Theoretical Computer Science*, 525:30–41, 2014.
 - 29 Sepp Hartung and Rolf Niedermeier. Incremental list coloring of graphs, parameterized by conservation. *Theoretical Computer Science*, 494:86–98, 2013.
 - 30 Giuseppe F. Italiano, Athanasios L. Konstantinidis, and Charis Papadopoulos. Structural parameterization of cluster deletion. In Chun-Cheng Lin, Bertrand M. T. Lin, and Giuseppe Liotta, editors, *Proceedings of the 17th International Conference and Workshops on Algorithms and Computation (WALCOM 2023)*, volume 13973 of *Lecture Notes in Computer Science*, pages 371–383. Springer, 2023. doi:10.1007/978-3-031-27051-2_31.
 - 31 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.
 - 32 Leon Kellerhals, Tomohiro Koana, André Nichterlein, and Philipp Zschoche. The PACE 2021 parameterized algorithms and computational experiments challenge: Cluster editing. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *LIPICs*, pages 26:1–26:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.IPEC.2021.26.
 - 33 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 2023)*, volume 259 of *LIPICs*, pages 18:1–18:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.CPM.2023.18.

- 34 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 2022)*, volume 249 of *LIPICs*, pages 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.IPEC.2022.20.
- 35 Christian Komusiewicz and Frank Sommer. Enumerating connected induced subgraphs: Improved delay and experimental comparison. *Discrete Applied Mathematics*, 303:262–282, 2021.
- 36 Christian Komusiewicz and Johannes Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012. doi:10.1016/j.dam.2012.05.019.
- 37 Shaohua Li, Marcin Pilipczuk, and Manuel Sorge. Cluster editing parameterized above modification-disjoint P_3 -packings. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS '21)*, volume 187 of *LIPICs*, pages 49:1–49:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.STACS.2021.49.
- 38 Junjie Luo, Hendrik Molter, André Nichterlein, and Rolf Niedermeier. Parameterized dynamic cluster editing. *Algorithmica*, 83(1):1–44, 2021. doi:10.1007/s00453-020-00746-y.
- 39 Dániel Marx. Searching the k -change neighborhood for TSP is $W[1]$ -hard. *Operations Research Letters*, 36(1):31–36, 2008.
- 40 Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007. doi:10.1016/j.cosrev.2007.05.001.
- 41 Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004. doi:10.1007/3-540-36379-3_33.
- 42 Sylwester Swat. PACE solver description: Clues - a heuristic solver for the cluster editing problem. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *LIPICs*, pages 32:1–32:3. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.IPEC.2021.32.
- 43 Stefan Szeider. The parameterized complexity of k -flip local search for SAT and MAX SAT. *Discrete Optimization*, 8(1):139–145, 2011.
- 44 Dekel Tsur. Faster parameterized algorithm for cluster vertex deletion. *Theory Comput. Syst.*, 65(2):323–343, 2021. doi:10.1007/s00224-020-10005-w.
- 45 Dekel Tsur. Cluster deletion revisited. *Information Processing Letters*, 173:106171, 2022. doi:10.1016/j.ipl.2021.106171.
- 46 Esther Ulitzsch, Qiwei He, Vincent Ulitzsch, Hendrik Molter, André Nichterlein, Rolf Niedermeier, and Steffi Pohl. Combining clickstream analyses and graph-modeled data clustering for identifying common response processes. *Psychometrika*, 86(1):190–214, 2021. doi:10.1007/s11336-020-09743-0.

Bandwidth Parameterized by Cluster Vertex Deletion Number

Tatsuya Gima ✉ 

JSPS Research Fellow, Nagoya University, Japan

Eun Jung Kim ✉ 

Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Noleen Köhler ✉ 

Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Nikolaos Melissinos ✉ 

Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Czech Republic

Manolis Vasilakis ✉ 

Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Abstract

Given a graph G and an integer b , BANDWIDTH asks whether there exists a bijection π from $V(G)$ to $\{1, \dots, |V(G)|\}$ such that $\max_{\{u,v\} \in E(G)} |\pi(u) - \pi(v)| \leq b$. This is a classical NP-complete problem, known to remain NP-complete even on very restricted classes of graphs, such as trees of maximum degree 3 and caterpillars of hair length 3. In the realm of parameterized complexity, these results imply that the problem remains NP-hard on graphs of bounded pathwidth, while it is additionally known to be $W[1]$ -hard when parameterized by the treedepth of the input graph. In contrast, the problem does become FPT when parameterized by the vertex cover number of the input graph. In this paper, we make progress towards the parameterized (in)tractability of BANDWIDTH. We first show that it is FPT when parameterized by the cluster vertex deletion number cvd plus the clique number ω of the input graph, thus generalizing the previously mentioned result for vertex cover. On the other hand, we show that BANDWIDTH is $W[1]$ -hard when parameterized only by cvd . Our results generalize some of the previous results and narrow some of the complexity gaps.

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

Keywords and phrases Bandwidth, Clique number, Cluster vertex deletion number, Parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.21

Related Version *Full Version:* <https://arxiv.org/abs/2309.17204>

Funding Our research visit to Nagoya University, Japan was funded by PRC CNRS JSPS project PARAGA (Parameterized Approximation Graph Algorithms).

Tatsuya Gima: Partially supported by JSPS KAKENHI Grant Number JP23KJ1066.

Eun Jung Kim: Supported by ANR project ANR-18-CE40-0025-01 (ASSK).

Noleen Köhler: Supported by ANR project ANR-18-CE40-0025-01 (ASSK).

Nikolaos Melissinos: Supported by the CTU Global postdoc fellowship program.

Manolis Vasilakis: Partially supported by ANR project ANR-21-CE48-0022 (S-EX-AP-PE-AL).

Acknowledgements We would like to thank Virginia Ardévol Martínez and Yota Otachi for interesting discussions at the preliminary stages of this work.



© Tatsuya Gima, Eun Jung Kim, Noleen Köhler, Nikolaos Melissinos, and Manolis Vasilakis; licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 21; pp. 21:1–21:15

Leibniz International Proceedings in Informatics



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

1 Introduction

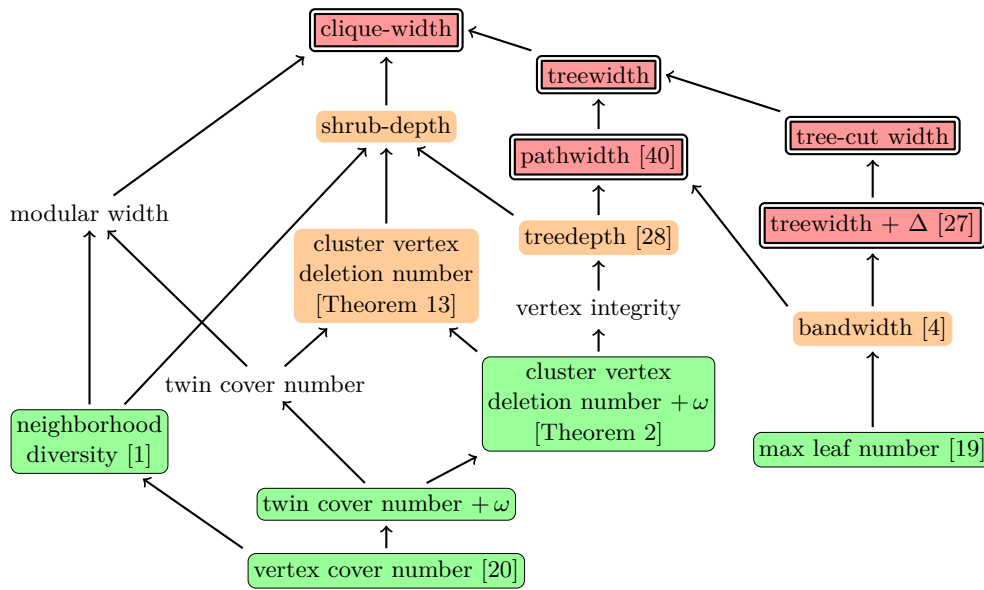
Given an undirected graph G and an integer b , BANDWIDTH asks whether there exists a bijection $\pi : V(G) \rightarrow \{1, \dots, |V(G)|\}$ of the vertices of G (called an *ordering*) such that $\max_{\{u,v\} \in E(G)} |\pi(u) - \pi(v)| \leq b$. The main motivation behind its study dates back to over half a century: a closely related problem in the field of matrix theory was first studied in the 1950's, while in the 1960's it was formulated as a graph problem, finding applications in minimizing (average) absolute error in codes, and has been extensively studied ever since [6, 11, 12, 13, 22, 30].

BANDWIDTH is long known to be NP-complete [6, 43]; as a matter of fact, it remains NP-complete even on very restricted classes of graphs, such as trees of maximum degree 3 [27], caterpillars of hair length 3 [40] and cyclic caterpillars of hair length 1 [41]. Considering these NP-hardness results, in this paper we focus on the parameterized complexity of BANDWIDTH. When parameterized by the natural parameter b , BANDWIDTH is known to be in XP [29, 44], whilst it is W[t]-hard for all positive integers t , even when the input graph is a tree [3, 4]. In fact, BANDWIDTH cannot be solved in time $f(b)n^{o(b)}$ even on trees of pathwidth at most two, unless the Exponential Time Hypothesis fails [17]. Regarding structural parameterizations, the previously mentioned results imply that BANDWIDTH is para-NP-complete when parameterized by the pathwidth or the treewidth plus the maximum degree of the input graph; the latter implies NP-completeness also on graphs of constant tree-cut width [26]. Moreover, it is known to be W[1]-hard parameterized by the treedepth of the input graph [28]. In contrast, the problem does become fixed-parameter tractable (FPT) when parameterized by the vertex cover number [20], the neighborhood diversity [1], or the max leaf number [19] of the input graph.

In the last few years, a plethora of structural parameters have been introduced, in an attempt to precisely determine the limits of tractability of algorithmic problems that are FPT by vertex cover, yet become W[1]-hard when parameterized by more general parameters, such as treewidth or clique-width. Some of the most well-studied such parameters are treedepth [42], twin cover number [24], cluster vertex deletion number [16], vertex integrity [28], shrub-depth [25], neighborhood diversity [35], and modular-width [23]. The tractability of BANDWIDTH with respect to those parameters has remained largely unexplored, with the exception of treedepth [28] and neighborhood diversity [1].

Cluster vertex deletion number lies between clique-width and vertex cover number (more precisely twin cover number), and is defined as the minimum size of a set of vertices whose removal induces a cluster graph, i.e. all of its components are cliques. It was first considered as a structural parameter in [16], and has been used to show parameterized (in)tractability results in multiple occasions ever since [2, 5, 7, 8, 33, 34, 38]. Notice that BANDWIDTH is trivial on cluster graphs; it suffices to check that the clique number is at most $b + 1$, as any optimal ordering places the vertices of every clique consecutively, for some ordering of the cliques. Therefore, its tractability when parameterized by the cluster vertex deletion number of the input graph poses a very natural question.

Our contribution. In the current work, we present both tractability and intractability results for BANDWIDTH when cluster vertex deletion number is a parameter of the problem (see Figure 1 for an overview of our results and the relationships between the structural parameters mentioned). We first prove that BANDWIDTH is FPT when parameterized by $\text{cvd} + \omega$, where cvd and ω denote the cluster vertex deletion number and clique number of the input graph respectively. This generalizes the tractability result for vertex cover number



■ **Figure 1** Our results and hierarchy of some related structural graph parameters, where ω and Δ denote the clique number and the maximum degree of the input graph, respectively. Arrows between parameters indicate generalization relations, that is, for any graph, if the parameter at the tail of an arrow is a constant then the parameter at the head of the arrow is also a constant. The reverse does not hold in this figure. The framed green, frameless orange, and double framed red rectangles indicate fixed-parameter tractable, $W[*]$ -hard, and NP-complete cases, respectively.

of [20], and follows the same idea of encoding the problem as an *integer linear program* (ILP) of a small number of variables. Solving said ILP, one can verify whether there exists any ordering π of the vertices of G such that a) $|\pi(v) - \pi(u)| \leq b$ for all $\{u, v\} \in E(G)$, and b) π is “nice”, where an ordering is nice if it has some specific properties. Proving that there exists a nice ordering π that minimizes $\max_{\{u,v\} \in E(G)} |\pi(v) - \pi(u)|$ then yields the stated result.

A natural question that arises from the previous result is whether it is necessary for both cvd and ω to be parameters of the problem in order to assure fixed-parameter tractability. Notice that **BANDWIDTH** is NP-complete even when $\omega \leq 2$, since that is the case for trees. Therefore, we proceed by studying the problem’s tractability when parameterized only by cvd . In this setting, we show that **BANDWIDTH** is $W[1]$ -hard via a reduction from **UNARY BIN PACKING**, thus positively answering the previous question. Note that the $W[1]$ -hardness of **BANDWIDTH** when parameterized by treedepth is also shown via a reduction from **UNARY BIN PACKING** [28].

Related work. **BANDWIDTH** is one of the so-called *graph layout* problems (see the survey of [14]). As far as the structural parameterized complexity of such problems is concerned, Fellows, Lokshtanov, Misra, Rosamond, and Saurabh [20] were the first to prove FPT results for a multitude of them when parameterized by the vertex cover number of the input graph, making use of ILP formulations. Since then, not much progress has been made on that front, with a notable exception being **IMBALANCE**, which was shown to be FPT when parameterized by twin cover number plus ω [39], vertex integrity [28], or tree-cut width [26], while it belongs to XP when parameterized by twin cover [39]. **MINIMUM LINEAR ARRANGEMENT** is known to be FPT parameterized by max leaf number, or edge clique number of the input graph [18],

as well as by the vertex cover number [37]. Lastly, as far as CUTWIDTH is concerned, a $2^{\mathcal{O}(\text{vc})}n^{\mathcal{O}(1)}$ time algorithm was presented in [10], improving over the ILP formulation of [20], where vc denotes the vertex cover number of the input graph.

Organization. In Section 2 we discuss the general preliminaries, followed by the FPT algorithm in Section 3 and the hardness result in Section 4. Lastly, in Section 5 we present the conclusion as well as some directions for future research. Proofs marked with (\star) are in the full version of the paper.

2 Preliminaries

Throughout the paper we use standard graph notation [15], and we assume familiarity with the basic notions of parameterized complexity [9]. We assume that \mathbb{N} is the set of all non-negative integers. All graphs considered are undirected without loops. The *clique number* of a graph G , denoted by $\omega(G)$, is the size of its largest induced clique. For $x, y \in \mathbb{Z}$, let $[x, y] = \{z \in \mathbb{Z} : x \leq z \leq y\}$, while $[x] = [1, x]$. For $\mathcal{I}_i = [a_i, b_i]$, we say that intervals $\mathcal{I}_1, \dots, \mathcal{I}_k$ *partition* interval $\mathcal{I} = [a, b]$ if $\mathcal{I} = \bigcup_{i \in [k]} \mathcal{I}_i$ and $\mathcal{I}_i \cap \mathcal{I}_j = \emptyset$, for any $1 \leq i < j \leq k$. Additionally, let $\mathcal{I}_i < \mathcal{I}_j$ if $b_i < a_j$. For a function $f : A \rightarrow B$ and $A' \subseteq A$, let $f(A') = \{f(a) \in B : a \in A'\}$. Moreover, let $\max(f(A')) = \max\{f(a) : a \in A'\}$ and $\min(f(A'))$ defined analogously.

Let G be a graph and $\pi : V(G) \rightarrow [n]$ an ordering of its vertices. We define the *stretch* of an edge $e = \{u, v\} \in E(G)$ with regard to π as $\text{stretch}_\pi(e) = |\pi(u) - \pi(v)|$. We define the *stretch* of π to be the maximum stretch of the edges of G , i.e. $\text{stretch}(\pi) = \max_{e \in E(G)} \text{stretch}_\pi(e)$. The *bandwidth* of G , denoted $\text{bw}(G)$, is the minimum stretch of any vertex ordering $\pi : V(G) \rightarrow [n]$.

► **Remark 1.** Note that the stretch of a vertex ordering is invariant under isomorphism, which means in particular that $\text{stretch}(\pi) = \text{stretch}(\pi \circ f)$ for any vertex ordering $\pi : V(G) \rightarrow [n]$ and any automorphism $f : V(G) \rightarrow V(G)$ of G .

A *cluster deletion set* of a graph G is a set $S \subseteq V(G)$ such that every component of $G - S$ is a clique. If S is a cluster deletion set, we call the components of $G - S$ *clusters*. The *cluster vertex deletion number* of a graph G , denoted by $\text{cvd}(G)$, is the size of its minimum cluster deletion set.

In the UNARY BIN PACKING problem, we are given a set of integers $A = \{a_j : j \in [n]\}$ in unary, as well as $k \in \mathbb{N}$, and are asked to determine whether there exists a partition (S_1, \dots, S_k) of A such that $\sum_{a_j \in S_i} a_j = \sum_{j \in [n]} a_j / k$ for every $i \in [k]$. UNARY BIN PACKING can be solved in time $n^{\mathcal{O}(k)}$ by employing dynamic programming, while it is known to be W[1]-hard parameterized by k [31].

The feasibility variant of *integer linear programming* (ILP) is to decide, given a set X of variables and a set C of linear constraints (i.e. inequalities) over the variables in X with integer coefficients, whether there is an assignment $\alpha : X \rightarrow \mathbb{Z}$ of the variables satisfying all constraints in C . It is known that the feasibility of an instance of (ILP) can be tested in $\mathcal{O}(p^{2.5p+o(p)} \cdot L)$ time, where p is the number of variables and L is the size of the input [21, 32, 36]. In other words, computing the feasibility of an ILP formula is FPT parameterized by the number of variables. Moreover, a solution can be computed in the same time if it exists.

3 An FPT-algorithm parameterized by cluster vertex deletion number plus clique number

In this section, we prove that BANDWIDTH is FPT when parameterized by the cluster vertex deletion number plus the clique number of the input graph.

► **Theorem 2.** *BANDWIDTH is fixed parameter tractable when parameterized by $cvd + \omega$, where cvd, ω denote the cluster vertex deletion number and clique number of the input graph respectively.*

Our proof is a generalization of the FPT result for vertex cover number from [20]. The general idea for obtaining an ILP encoding of BANDWIDTH given a vertex cover S is to augment S by a small number (dependent only on the vertex cover number) of representative vertices of every neighborhood-type. It can be easily seen that we can modify any ordering π in such a way that the leftmost and rightmost neighbor of any vertex in S is contained in this augmented set S' without increasing the stretch. For any ordering σ of S' we can decide whether we can extend σ to an ordering of $V(G)$ of stretch at most b by encoding how vertices of certain neighborhood-types are distributed into the gaps between the vertices of S' into an ILP. By ensuring that we distribute the vertices in such a way that the leftmost and rightmost neighbor of any vertex in S is contained in S' we can bound the stretch of every edge by using one linear constraint for every edge in $G[S']$.

In our setting we can use the vertex cover approach to bound the stretch of all edges incident to the deletion set. To gain control over the stretch of edges within clusters, we show that we can convert any ordering into a nice ordering without increasing the stretch. Here niceness intuitively means, that we can order the vertices in between any two vertices of S' in such a way that vertices of the same type appear consecutively, where the type now depends on the isomorphism-type of the cluster union the deletion set. This will allow us to bound the stretch of such edges by a linear constraint as well.

3.1 Types and buckets

Let G be a graph and S a cluster deletion set of G of size k . For any vertex $v \in V(G - S)$, let $N_S(v) = N(v) \cap S$ be its S -neighborhood. Let $\mathcal{K} \subseteq \mathbb{N}^{2^k}$ be the set of non-negative integer vectors κ with 2^k entries for which $\|\kappa\|_1 \leq \omega(G)$. Here $\|\cdot\|_1$ denotes the 1-norm, i.e. the sum of the absolute value of the entries. We assume that the entries of the vectors in \mathcal{K} are indexed by the subsets of S . We say that a cluster C has *cluster-type* $\kappa \in \mathcal{K}$ if $|\{v \in V(C) : N_S(v) = N\}| = (\kappa)_N$ for every $N \subseteq S$ where $(\kappa)_N$ denotes the entry of κ corresponding to N . We further let $\#\kappa$ denote the number of clusters of cluster-type κ in G . We say that a set \mathcal{C} of clusters is *representative* if it consists of $\min\{2|S|, \#\kappa\}$ distinct clusters of type κ for every cluster-type κ . We further say that a set S' is an *extended deletion set* if $S' = S \cup \bigcup_{C \in \mathcal{C}} V(C)$ for a representative set \mathcal{C} of clusters.

► **Lemma 3** (\star). *For every extended deletion set S' there is an ordering $\pi : V(G) \rightarrow [n]$ such that $\text{stretch}(\pi) = \text{bw}(G)$ and for every $s \in S$, the set S' contains vertices v_{\min}^s and v_{\max}^s , where $\pi(v_{\min}^s) = \min(\pi(N(s)))$ and $\pi(v_{\max}^s) = \max(\pi(N(s)))$, i.e. S' contains the leftmost and rightmost neighbor of s .*

We say that an ordering $\pi : V(G) \rightarrow [n]$ is S' -*extremal* if the second property in Lemma 3 is satisfied for π .

► **Observation 4.** *The size of any extended deletion set S' is at most $|S| + 2|S|\omega(G) \cdot 2^{2|S| \cdot \omega(G)}$.*

Let \mathcal{C} be a representative set of clusters, $S' = S \cup \bigcup_{C \in \mathcal{C}} V(C)$ the extended deletion set containing vertices from \mathcal{C} and S and set $k' = |S'|$. A *bucket distribution* of S' is a partition $\mathcal{B} = (B_0, \dots, B_{k'})$ of the vertices of $G - S'$. Fix a bucket distribution $\mathcal{B} = (B_0, \dots, B_{k'})$ of S' . We call the subsets B_i *buckets* of \mathcal{B} .

Let $\mathcal{T} \subseteq \mathbb{N}^{2^k \times (k'+1)}$ be the set of matrices τ with $\|\tau\|_1 \leq \omega(G)$. We assume that the rows of matrices are indexed with subsets of S and the columns with $[0, k']$. We say that a cluster $C \notin \mathcal{C}$ has *distribution-type* $\tau \in \mathcal{T}$ in \mathcal{B} if $|\{v \in V(C) \cap B_i : N_S(v) = N\}| = (\tau)_{N,i}$ for every $N \subseteq S$ and every $i \in [0, k']$. For every $\kappa \in \mathcal{K}$, let $\mathcal{T}_\kappa \subseteq \mathcal{T}$ denote the set of distribution-types τ such that $\sum_{i \in [0, k']} (\tau)_{N,i} = (\kappa)_N$ for every $N \subseteq S$, i.e., the set of $\tau \in \mathcal{T}$ such that any cluster of distribution-type τ has cluster-type κ .

► **Observation 5.** *The number of distribution-types is at most $\omega(G)^{2^{|S|} \cdot (|S'|+1)}$ for any extended deletion set S' .*

Let $\sigma : S' \rightarrow [k']$ be an ordering of the vertices of S' . We say that a vertex ordering $\pi : V(G) \rightarrow [n]$ is *compatible* with σ if for any $s_1, s_2 \in S'$ it holds that $\pi(s_1) < \pi(s_2)$ if and only if $\sigma(s_1) < \sigma(s_2)$. We say that a vertex ordering $\pi : V(G) \rightarrow [n]$ is *compatible* with σ and \mathcal{B} if π is compatible with σ and $B_0 = \{v \in V(G) : \pi(v) < \pi(\sigma^{-1}(1))\}$, $B_{k'} = \{v \in V(G) : \pi(v) > \pi(\sigma^{-1}(k'))\}$ and $B_i = \{v \in V(G) : \pi(\sigma^{-1}(i)) < \pi(v) < \pi(\sigma^{-1}(i+1))\}$ for $i \in [k' - 1]$.

3.2 Nice orderings

Let G be a graph and S a cluster deletion set of G . Furthermore, let \mathcal{C} be a representative set of clusters, $S' = S \cup \bigcup_{C \in \mathcal{C}} V(C)$ the extended deletion set containing vertices from \mathcal{C} and S and $k' = |S'|$. Additionally, we fix a bucket distribution $\mathcal{B} = (B_0, \dots, B_{k'})$ of S' and an ordering $\sigma : S' \rightarrow [k']$.

To obtain our nice ordering we use a series of exchange arguments that will not increase the stretch. We call an ordering *nice* if it has properties (Π_1) , (Π_2) and (Π_3) . We will first give some intuition regarding the properties, before defining them formally.

Assume $\pi : V(G) \rightarrow [n]$ is an optimal ordering minimizing the number of edges of maximum stretch. Furthermore, let $v \in V(C)$ be a vertex which is contained in an edge of maximum stretch with regards to π and the cluster C containing v is distributed over more than one bucket. In this case, v must be either the leftmost or the rightmost vertex of C . Assuming v is the leftmost vertex of C (the other case is analogous), we can observe that every vertex $v' \in B_i$ appearing further to the right than v must have a neighbor contained in a bucket to the right of B_i and no neighbor to the left of v . Otherwise, we can reduce the stretch of the edge containing v without increasing the stretch of any edge incident to v' (and hence reducing the number of edges of maximum stretch without increasing the maximum stretch) by exchanging v and v' . Using this observation, we can assume that each bucket is partitioned into a left, a middle and a right part and every vertex with only neighbors to the left of B_i appears in the left part and every vertex having only neighbors to the right of B_i appears in the right part. Additionally, the above observation allows us to assume that within each bucket the vertices of one cluster appear consecutively (property (Π_1)).

Now assume that $\{v, w\}$ is an edge of maximum stretch as before (v appears left of w in π) and $\{v', w'\}$ is another edge such that v' appears in the same bucket as v and w' in the same bucket as w . If v' appears before v then w' has to appear before w as $\{v, w\}$ is of maximum stretch. On the other hand, if v' appears after v then w' must appear after w as otherwise exchanging w and w' either reduces the number of edges of maximum stretch or reduces the maximum stretch itself. Hence, we can assume that the relative order of the leftmost vertices of a set of clusters is the same as the relative order of the rightmost vertices of the same clusters (property (Π_2)).

Lastly, assume that C and C' are clusters of type $\tau \in \mathcal{T}$ which are not contained in just one bucket and appear next to each other (in their leftmost bucket). Assume B_ℓ is the bucket containing the leftmost vertex of C and C' and B_r the bucket containing the rightmost vertex of C and C' . We can essentially exchange $V(C) \cap B_\ell$ with $V(C') \cap B_\ell$ and at the same time $V(C) \cap B_r$ with $V(C') \cap B_r$ if certain properties about the size of these sets hold. This allows us to order the buckets in such a way, that clusters whose intersection with the leftmost (rightmost, respectively) bucket they intersect is of the same size, appear consecutively (property (Π_3)).

To state the three properties formally we use the following notation. For a distribution-type $\tau \in \mathcal{T}$ and $i \in [0, k']$, we write τ_i to denote the column of τ which is indexed by i . We define $\text{LB}(\tau)$ to be the largest index $i \in [0, k']$ such that $\|\tau_j\|_1 = 0$ for any $j \in [0, i - 1]$, i.e. B_i is the leftmost bucket containing vertices from clusters of type τ . We define $\text{RB}(\tau)$ analogously to be the minimum index $i \in [0, k']$ such that $\|\tau_j\|_1 = 0$ for any $j \in [i + 1, k']$. Additionally, we let $\#L(\tau)$ be $\|\tau_{\text{LB}(\tau)}\|_1$ and $\#R(\tau)$ be $\|\tau_{\text{RB}(\tau)}\|_1$. For every $\ell \leq r \in [0, k']$ and every $n_L, n_R \in [0, \omega(G)]$, we define

$$\mathcal{T}^{(\ell, r, n_L, n_R)} = \{\tau \in \mathcal{T} : \text{LB}(\tau) = \ell, \text{RB}(\tau) = r, \#L(\tau) = n_L, \#R(\tau) = n_R\}.$$

► **Definition 6** (Property (Π_1)). *We say that an S' -extremal ordering $\pi : V(G) \rightarrow [n]$ which is compatible with σ and \mathcal{B} has property (Π_1) if for every $i \in [0, k']$*

1. *the vertices of $V(C) \cap B_i$ appear consecutively in π for every cluster $C \notin \mathcal{C}$,*
2. *we can partition the interval $\pi(B_i)$ into three (possibly empty) intervals $I_R^i < I_M^i < I_L^i$ such that for every $\tau \in \mathcal{T}$ and every cluster C of distribution-type τ*
 - $\pi(V(C) \cap B_i) \subseteq I_R^i$ if $\text{LB}(\tau) \neq i$ and $\text{RB}(\tau) = i$,
 - $\pi(V(C) \cap B_i) \subseteq I_L^i$ if $\text{LB}(\tau) = i$ and $\text{RB}(\tau) \neq i$,
 - $\pi(V(C) \cap B_i) \subseteq I_M^i$ if either $\text{LB}(\tau) \neq i$ and $\text{RB}(\tau) \neq i$ or $\text{LB}(\tau) = \text{RB}(\tau) = i$.

Notice that while I_R^i contains the leftmost ordered vertices of B_i , we use the index R since those vertices are the rightmost vertices of their corresponding cliques. Analogously, we use I_L^i for the rightmost ordered vertices of B_i .

► **Definition 7** (Property (Π_2)). *We say that an S' -extremal ordering $\pi : V(G) \rightarrow [n]$ which is compatible with σ and \mathcal{B} has property (Π_2) if for any two distribution-types $\tau, \tau' \in \mathcal{T}$ and any two clusters C and C' of distribution-type τ and τ' respectively, the following holds.*

- *If either $\text{LB}(\tau) = \text{LB}(\tau')$ or $\text{RB}(\tau) = \text{RB}(\tau')$, then for any $v \in V(C) \cap B_{\text{LB}(\tau)}$, $v' \in V(C') \cap B_{\text{LB}(\tau')}$, $w \in V(C) \cap B_{\text{RB}(\tau)}$, $w' \in V(C') \cap B_{\text{RB}(\tau')}$ we have that $\pi(v) < \pi(v')$ if and only if $\pi(w) < \pi(w')$.*

Lastly, we want the buckets to be ordered by distribution-types which will enable us to express the stretch within clusters by linear constraints. To achieve this, we define two orderings of distribution-types, dictating in which order (in a nice, optimal vertex ordering) cliques of a certain type will appear within a bucket. First, let $\mathcal{T}_R^i = \bigcup_{\substack{\ell \in [0, i-1], \\ n_L, n_R \in [\omega(G)]}} \mathcal{T}^{(\ell, i, n_L, n_R)}$

and define the ordering $\rho_i : \mathcal{T}_R^i \rightarrow [|\mathcal{T}_R^i|]$ in the following way. For any $\tau \in \mathcal{T}^{(\ell, i, n_L, n_R)}$, $\tau' \in \mathcal{T}^{(\ell', i, n'_L, n'_R)}$, we have that $\rho_i(\tau) < \rho_i(\tau')$ if either

- $\ell < \ell'$ or
- $\ell = \ell'$, $n_L \geq n_R$ and $n'_L < n'_R$ or
- $\ell = \ell'$, $n_L \geq n_R$, $n'_L \geq n'_R$ and $n_R < n'_R$ or
- $\ell = \ell'$, $n_L < n_R$, $n'_L < n'_R$ and $n_L > n'_L$ or
- $\ell = \ell'$, $n_L \geq n_R$, $n'_L \geq n'_R$, $n_R = n'_R$ and $\tau \leq_{\text{lex}} \tau'$ or
- $\ell = \ell'$, $n_L < n_R$, $n'_L < n'_R$, $n_L = n'_L$ and $\tau \leq_{\text{lex}} \tau'$.

Here \leq_{lex} refers to the lexicographic order on matrices in \mathcal{T} where we read the entries by lines top to bottom. However, we can replace this by any total ordering (\leq_{lex} is an arbitrary choice).

Moreover, let $\mathcal{T}_L^i = \bigcup_{\substack{r \in [i+1, k'] \\ n_L, n_R \in [\omega(G)]}} \mathcal{T}^{(i, r, n_L, n_R)}$ and define the ordering $\lambda_i : \mathcal{T}_L^i \rightarrow [|\mathcal{T}_L^i|]$ by letting $\lambda_i(\tau) < \lambda_i(\tau')$ for any $\tau \in \mathcal{T}^{(i, r, n_L, n_R)}$, $\tau' \in \mathcal{T}^{(i, r', n'_L, n'_R)}$ if either

- $r < r'$ or
- $r = r'$ and $\rho_i(\tau) < \rho_i(\tau')$.

► **Remark 8.** Note that we can compute all ρ_i and λ_i in time quadratic in the size of \mathcal{T} .

► **Definition 9** (Property (Π_3)). *We say that an S' -extremal ordering $\pi : V(G) \rightarrow [n]$ which is compatible with σ and \mathcal{B} has property (Π_3) if for every $i \in [0, k']$ we can partition the interval $\pi(B_i)$ into (possibly empty) intervals*

$$J_R^{(i,1)} < \dots < J_R^{(i,|\mathcal{T}_R^i|)} < J_M^i < J_L^{(i,1)} < \dots < J_L^{(i,|\mathcal{T}_L^i|)}$$

such that for every distribution-type $\tau \in \mathcal{T}$ and every cluster C of type τ and every $j \in [|\mathcal{T}_R^i|]$, $j' \in [|\mathcal{T}_L^i|]$,

- $\pi(V(C) \cap B_i) \subseteq J_R^{(i,j)}$ if $\rho_i(\tau) = j$ and
- $\pi(V(C) \cap B_i) \subseteq J_L^{(i,j')}$ if $\lambda_i(\tau) = j'$.

► **Lemma 10** (*). *Given an S' -extremal ordering $\pi : V(G) \rightarrow [n]$ which is compatible with σ and \mathcal{B} , there exists an S' -extremal ordering $\pi' : V(G) \rightarrow [n]$ of $\text{stretch}(\pi') \leq \text{stretch}(\pi)$ which is compatible with σ and \mathcal{B} and has properties (Π_1) , (Π_2) and (Π_3) .*

3.3 ILP formulation

Let G be a graph and S a cluster deletion set of G . Furthermore, let \mathcal{C} be a representative set of clusters, $S' = S \cup \bigcup_{C \in \mathcal{C}} V(C)$ the extended deletion set containing vertices from \mathcal{C} and S and $k' = |S'|$.

For every ordering $\sigma : S' \rightarrow k'$, we will use an ILP to determine whether there is an S' -extremal ordering $\pi : V(G) \rightarrow [n]$ of stretch at most b which is compatible with σ . The ILP has two variables x_τ, y_τ for every distribution-type $\tau \in \mathcal{T}$. The variable x_τ expresses how many clusters of $G - S'$ have distribution-type τ in an optimal S' -extremal ordering compatible with σ . The variable y_τ is an indicator variable which is 1 if and only if $x_\tau > 0$ and 0 otherwise. We further use z_i for $i \in [0, k']$ in our ILP formulation as a placeholder for the expression $\sum_{\tau \in \mathcal{T}} (x_\tau \cdot \|\tau_i\|_1)$ which expresses the number of vertices in bucket i . For an assignment $\alpha : \{x_\tau, y_\tau : \tau \in \mathcal{T}\} \rightarrow \mathbb{N}$ of the variables of our ILP, we write $\alpha(z_i)$ to stand for the expression $\sum_{\tau \in \mathcal{T}} (\alpha(x_\tau) \cdot \|\tau_i\|_1)$.

We further need the leftmost and rightmost neighbor of any vertex of S in S' , thus define $v_{\min, \sigma}^s, v_{\max, \sigma}^s \in S'$ such that $\sigma(v_{\min, \sigma}^s) = \min(\sigma(N(s)))$ and $\sigma(v_{\max, \sigma}^s) = \max(\sigma(N(s)))$, for every ordering $\sigma : S' \rightarrow k'$ and $s \in S$. Note that by choosing S to be minimum, we can assume that S contains no vertex with no neighbors in $G - S$ and hence $v_{\min, \sigma}^s$ and $v_{\max, \sigma}^s$ are well defined.

For a fixed ordering $\sigma : S' \rightarrow [k']$, we can now formulate our set of linear constraints. The first three constraints ensure that we choose the number of clusters that have a certain distribution-type in a feasible way. That is, (T1) ensures that the quantities of distribution-types corresponding to an assignment of the variables x_τ corresponds to a valid choice of allocating each available cluster in the input graph G a distribution-type. As for (T2), it ensures that $v_{\min, \sigma}^s$ is indeed the leftmost neighbor of s while $v_{\max, \sigma}^s$ is the rightmost neighbor

of s for every $s \in S$ by ensuring that any distribution-type placing a neighbor of s in a bucket left of $v_{\min, \sigma}^s$ or right of $v_{\max, \sigma}^s$ does not occur. Finally, (T3) guarantees that y_τ indeed indicates whether or not distribution-type τ is used in the solution.

(T1) For every $\kappa \in \mathcal{K}$,

$$\#\kappa = \min\{\#\kappa, 2k\} + \sum_{\tau \in \mathcal{T}_\kappa} x_\tau.$$

(T2) For every $s \in S$ and every $\tau \in \mathcal{T}$ for which $\tau_{N,i} > 0$ for some $N \ni s$ and $i \in [0, \sigma(v_{\min, \sigma}^s) - 1] \cup [\sigma(v_{\max, \sigma}^s), k']$,

$$x_\tau = 0.$$

(T3) $x_\tau \cdot (1 - y_\tau) = 0$ and $(1 - x_\tau) \cdot y_\tau \leq 0$ for every $\tau \in \mathcal{T}$.

The purpose of all remaining constraints is to ensure that for the assignment of variables, which essentially corresponds to choosing a bucket distribution \mathcal{B} , there is an S' -extremal ordering $\pi : V(G) \rightarrow [n]$ which is compatible with σ and \mathcal{B} for which $\text{stretch}(\pi) \leq b$. (DS) expresses that the stretch of edges in $G[S']$ is bounded by b .

(DS) For every $s, s' \in S'$ with $\{s, s'\} \in E(G)$, $\sigma(s) < \sigma(s')$,

$$b \geq \sigma(s') - \sigma(s) + \sum_{i \in [\sigma(s), \sigma(s') - 1]} z_i.$$

The last three constraints deal with bounding the stretch of edges within clusters. For this we assume that the S' -extremal ordering which is consistent with σ and \mathcal{B} is nice, i.e. has properties (Π_1) , (Π_2) and (Π_3) . The first constraint (C1) is necessary to bound the stretch of clusters that are fully contained in one bucket. To bound the stretch of clusters contained in multiple buckets, we have one constraint for every distribution-type $\tau \in \mathcal{T}^{(\ell, r, n_L, n_R)}$ for any $\ell < r \in [0, k']$, $n_L, n_R \in [\omega(G)]$. By property (Π_3) we know that there are intervals $J_L^{(\ell, \lambda_\ell(\tau))}$ containing all vertices from $B_\ell \cap V(C)$ and $J_R^{(r, \rho_r(\tau))}$ containing all vertices $B_r \cap V(C)$ for every cluster C of distribution-type τ . The trick now is to observe that if $n_L \geq n_R$ then the first cluster appearing in $J_L^{(\ell, \lambda_\ell(\tau))}$ observes the maximum stretch while if $n_L < n_R$ it is the last clique. Using this we can express with constraints (C2) and (C3) that the stretch of every cluster of distribution-type τ is bounded by b .

(C1) ($b \geq \omega(G) - 1$).

(C2) For every $\ell < r \in [0, k']$, $n_L \geq n_R \in [\omega(G)]$ and $\tau \in \mathcal{T}^{(\ell, r, n_L, n_R)}$,

$$b \geq y_\tau \cdot \left(\sum_{\tau' \in \lambda_\ell^{-1}([\lambda_\ell(\tau), |\mathcal{T}_L^\ell|])} \#\text{L}(\tau') \cdot x_{\tau'} + \sum_{\ell < i < r} z_i + (r - \ell) \right. \\ \left. + \sum_{\tau' \in \rho_r^{-1}([1, \rho_r(\tau) - 1])} \#\text{R}(\tau') \cdot x_{\tau'} + n_R - 1 \right).$$

(C3) For every $\ell < r \in [0, k']$, $n_L < n_R \in [\omega(G)]$ and $\tau \in \mathcal{T}^{(\ell, r, n_L, n_R)}$,

$$b \geq y_\tau \cdot \left(n_L + \sum_{\tau' \in \lambda_\ell^{-1}([\lambda_\ell(\tau) + 1, |\mathcal{T}_L^\ell|])} \#\text{L}(\tau') \cdot x_{\tau'} + \sum_{\ell < i < r} z_i + (r - \ell) \right. \\ \left. + \sum_{\tau' \in \rho_r^{-1}([1, \rho_r(\tau)])} \#\text{R}(\tau') \cdot x_{\tau'} - 1 \right).$$

► **Lemma 11** (\star). *For any ordering $\sigma : S' \rightarrow [k']$, there is an S' -extremal ordering $\pi : V(G) \rightarrow [n]$ of stretch at most b which is compatible with σ if and only if the system of linear equation $(T1, T2, T3, DS, C1, C2, C3)$ for σ admits a solution.*

Using Lemma 11 we obtain an FPT-algorithm, which computes $S, \#\kappa$ for every cluster-type κ and arbitrary picks an extended deletion set S' . The algorithm then for every ordering $\sigma : S' \rightarrow [k']$ verifies whether the ILP admits a solution in which case the input is a YES-instance of BANDWIDTH. Details are given in the full version of the paper.

► **Remark 12.** Using a minimization ILP, we can in fact *construct* an ordering of *minimum* stretch (and not just argue about the existence of an ordering of stretch at most b), since all the exchange arguments of Section 3.2 are constructive.

4 W[1]-hardness parameterized by cluster vertex deletion number

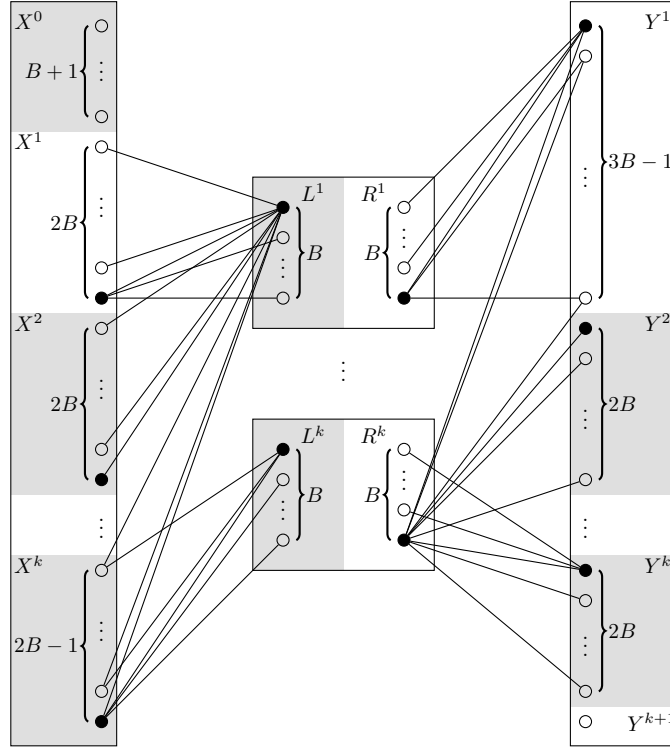
In this section, we prove that BANDWIDTH is W[1]-hard when parameterized by the cluster vertex deletion number of the input graph. In order to do so, we reduce from an instance of UNARY BIN PACKING. Before we present the details of the construction, we first give some high-level intuition.

For an instance (A, k) of UNARY BIN PACKING we want to construct an equivalent instance (G, b) of BANDWIDTH, such that $\text{cvd}(G) = f(k)$ for some function f . Roughly, the graph G consists of cliques representing the items of the UNARY BIN PACKING instance and cliques that act as delimiters separating the items contained in some bucket from the items contained in the next bucket. However, in order to guarantee that the entirety of every item clique is placed in between two consecutive delimiter cliques and that the values of the items in between two delimiter cliques add up to B (the capacity of the bins in the UNARY BIN PACKING instance (A, k)), some extra structure is needed. First we introduce two cliques of size $b + 1$ that will be used as boundaries. By making each item clique and each delimiter clique of the graph adjacent to some vertex in both of the boundary cliques, it follows that in any ordering of stretch at most b , all item cliques and all delimiter cliques of the graph will be positioned in between the two boundary cliques.

As the size of the deletion set cannot depend on the number or values of the items, item cliques cannot be incident to individual deletion set vertices. This makes it tricky to enforce that every vertex of an item clique is contained in between the same two delimiter cliques as a majority of the item cliques would not be incident to any edge of maximum stretch and therefore allow them a lot of freedom of movement. In order to cope with this issue, we introduce a perfect copy of the delimiter and item cliques, as well as edges between the original cliques and their copies resulting in them becoming twice as big consisting of a left part, the original vertices, and a right part, the copy vertices. The left part of all cliques will be connected to the left boundary clique and will therefore appear to the left of the right parts. The right part will be connected to the right hand boundary cliques. The item cliques will now be kept in place by having maximum stretch between the vertices of the left part and the vertices of the right part.

► **Theorem 13.** *BANDWIDTH is W[1]-hard when parameterized by the cluster deletion number of the input graph.*

Construction. Let (A, k) be an instance of UNARY BIN PACKING, where $A = \{a_1, \dots, a_n\}$. Moreover, let $B = \sum_{j \in [n]} a_j/k$ be the capacity of every bin, where $B \in \mathbb{N}$, since otherwise this would have been a trivial instance. Set $b = 2kB + B - 1$. We will construct an equivalent instance (G, b) of BANDWIDTH as follows.



■ **Figure 2** Part of G , showing only the boundary and the delimiter cliques. Rectangles denote cliques, brackets denote number of vertices and black vertices compose a cluster deletion set.

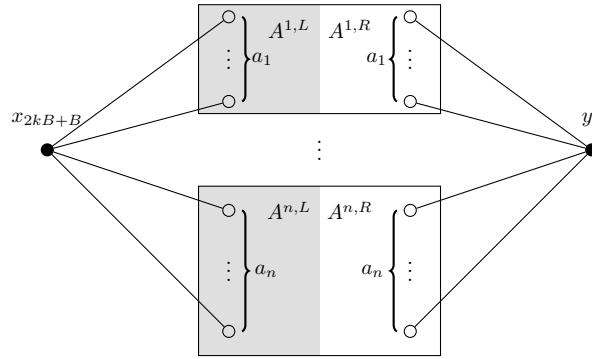
Boundary cliques. First, we create two cliques X and Y , referred to as *boundary cliques*, where $V(X) = \{x_1, \dots, x_{2kB+B}\}$ and $V(Y) = \{y_1, \dots, y_{2kB+B}\}$. We consider the following partition of the vertices of X : let $X^0 = \{x_1, \dots, x_{B+1}\}$ and for every $i \in [k-1]$ we denote the set $\{x_{2iB-B+2}, \dots, x_{2iB+B+1}\}$ by X^i , while $X^k = \{x_{2kB-B+2}, \dots, x_{2kB+B}\}$. Note that $|X^0| = B+1$, $|X^k| = 2B-1$ and $|X^i| = 2B$, for all $i \in [k-1]$. Moreover, we partition the vertices of Y in a similar but slightly asymmetric way: let $Y^1 = \{y_1, \dots, y_{3B-1}\}$ and for every $i \in [2, k]$ we denote the set $\{y_{2iB-B}, \dots, y_{2iB+B-1}\}$ by Y^i , while $Y^{k+1} = \{y_{2kB+B}\}$. Note that $|Y^1| = 3B-1$, $|Y^{k+1}| = 1$ and $|Y^i| = 2B$, for all $i \in [2, k]$.

Delimiter cliques. For every $i \in [k]$ we create a clique on vertex set $\{\ell_1^i, \dots, \ell_B^i, r_1^i, \dots, r_B^i\}$ of size $2B$. We denote the set $\{\ell_1^i, \dots, \ell_B^i\}$ by L^i and the set $\{r_1^i, \dots, r_B^i\}$ by R^i . Moreover, let $L = \bigcup_{i=1}^k L^i$ and $R = \bigcup_{i=1}^k R^i$. We add the following edges:

- For every $i \in [k]$, $x \in \bigcup_{j=i}^k X^j$, we add the edge $\{\ell_1^i, x\}$.
- For every $i \in [k-1]$, $\ell \in L^i$, we add the edge $\{x_{2iB+B+1}, \ell\}$. Moreover, we add an edge between x_{2kB+B} and every vertex of L^k .
- For every $i \in [k]$, $y \in \bigcup_{j=1}^i Y^j$, we add the edge $\{r_B^i, y\}$.
- For every $i \in [2, k]$, $r \in R^i$, we add the edge $\{y_{2iB-B}, r\}$. Moreover, we add an edge between y_1 and every vertex of R^1 .

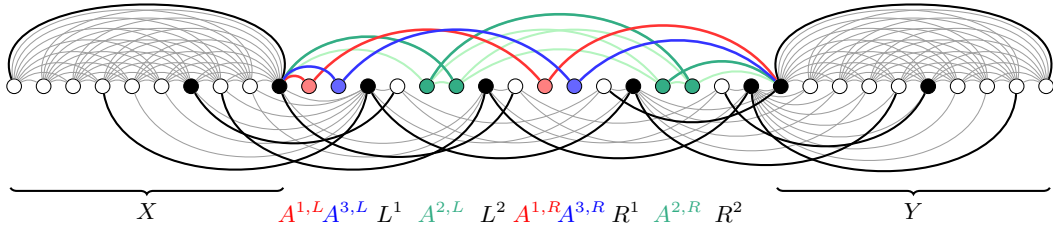
For an illustration of the boundary and delimiter cliques, see Figure 2.

Item cliques. For element $a_i \in A$, we construct a clique A^i on vertex set $\{a_j^{i,L}, a_j^{i,R} : j \in [a_i]\}$ of size $2a_i$. We denote the set of vertices $\{a_j^{i,L} : j \in [a_i]\}$ by $A^{i,L}$ and the set of vertices $\{a_j^{i,R} : j \in [a_i]\}$ by $A^{i,R}$. We add edges $\{x_{2kB+B}, a\}$ for every $a \in \bigcup_{i \in [k]} A^{i,L}$ and edges $\{y_1, a\}$ for every $a \in \bigcup_{i \in [k]} A^{i,R}$. For an illustration, see Figure 3.



■ **Figure 3** Rectangles denote cliques. Black vertices compose a cluster deletion set.

This concludes the construction of G . Figure 4 illustrates an example of an ordering of stretch b obtained by a YES-instance of UNARY BIN PACKING. In the following, we prove the equivalence of (G, b) to the initial instance of UNARY BIN PACKING.



■ **Figure 4** For the instance $(\{a_1, a_2, a_3\}, 2)$ of UNARY BIN PACKING with $a_1 = 1$, $a_2 = 2$ and $a_3 = 1$ the figure shows the graph G from the corresponding instance $(G, 9)$ of BANDWIDTH. Here the ordering of the vertices of G with stretch 9 corresponds to the solution of $(\{a_1, a_2, a_3\}, 2)$ in which a_1, a_3 are placed in the first bin and a_2 in the second.

► **Lemma 14** (★). *If (A, k) is a YES-instance of UNARY BIN PACKING, then (G, b) is a YES-instance of BANDWIDTH.*

► **Lemma 15** (★). *If (G, b) is a YES-instance of BANDWIDTH, then (A, k) is a YES-instance of UNARY BIN PACKING.*

► **Lemma 16** (★). *It holds that $\text{cvd}(G) = \mathcal{O}(k)$.*

5 Conclusion

In the current work, we extend our understanding of BANDWIDTH in the setting of parameterized complexity. In particular, we have shown that the problem is FPT when parameterized by the cluster vertex deletion number cvd plus the clique number ω of the input graph, although it becomes W[1]-hard when parameterized only by cvd .

The most natural research direction would be to explore the tractability of the problem when parameterized by twin cover, modular-width or vertex integrity, given the lack of any relevant FPT/XP algorithms or hardness results. As a matter of fact, it is not even known whether the problem is in XP when parameterized by cvd or treedepth.

Finally, most tractability results for the various structural parameters rely on some ILP formulation. This raises the question of whether any other kind of approach is applicable, as is the case for CUTWIDTH [10].

References

- 1 Olav Røthe Bakken. Arrangement problems parameterized by neighbourhood diversity. Master's thesis, University of Bergen, 2018.
- 2 Aritra Banik, Prahlad Narasimhan Kasthurirangan, and Venkatesh Raman. Dominator coloring and CD coloring in almost cluster graphs. In *Algorithms and Data Structures - 18th International Symposium, WADS 2023*, volume 14079 of *Lecture Notes in Computer Science*, pages 106–119. Springer, 2023. doi:10.1007/978-3-031-38906-1_8.
- 3 Hans L. Bodlaender. Parameterized complexity of bandwidth of caterpillars and weighted path emulation. In *Graph-Theoretic Concepts in Computer Science - 47th International Workshop, WG 2021*, volume 12911 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2021. doi:10.1007/978-3-030-86838-3_2.
- 4 Hans L. Bodlaender, Michael R. Fellows, and Michael T. Hallett. Beyond np-completeness for problems of bounded width: hardness for the W hierarchy. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 449–458. ACM, 1994. doi:10.1145/195058.195229.
- 5 Henning Bruhn, Morgan Chopin, Felix Joos, and Oliver Schaudt. Structural parameterizations for boxicity. *Algorithmica*, 74(4):1453–1472, 2016. doi:10.1007/s00453-015-0011-0.
- 6 Phyllis Z. Chinn, J. Chvatalova, A. K. Dewdney, and Norman E. Gibbs. The bandwidth problem for graphs and matrices - a survey. *J. Graph Theory*, 6(3):223–254, 1982. doi:10.1002/jgt.3190060302.
- 7 Janka Chlebíková and Morgan Chopin. The firefighter problem: A structural analysis. In *Parameterized and Exact Computation - 9th International Symposium, IPEC 2014*, volume 8894 of *Lecture Notes in Computer Science*, pages 172–183. Springer, 2014. doi:10.1007/978-3-319-13524-3_15.
- 8 Morgan Chopin, André Nichterlein, Rolf Niedermeier, and Mathias Weller. Constant thresholds can make target set selection tractable. *Theory Comput. Syst.*, 55(1):61–83, 2014. doi:10.1007/s00224-013-9499-3.
- 9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 10 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. On cutwidth parameterized by vertex cover. *Algorithmica*, 68(4):940–953, 2014. doi:10.1007/s00453-012-9707-6.
- 11 Marek Cygan and Marcin Pilipczuk. Exact and approximate bandwidth. *Theor. Comput. Sci.*, 411(40–42):3701–3713, 2010. doi:10.1016/j.tcs.2010.06.018.
- 12 Marek Cygan and Marcin Pilipczuk. Bandwidth and distortion revisited. *Discret. Appl. Math.*, 160(4-5):494–504, 2012. doi:10.1016/j.dam.2011.10.032.
- 13 Marek Cygan and Marcin Pilipczuk. Even faster exact bandwidth. *ACM Trans. Algorithms*, 8(1):8:1–8:14, 2012. doi:10.1145/2071379.2071387.
- 14 Josep Díaz, Jordi Petit, and Maria J. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002. doi:10.1145/568522.568523.
- 15 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate texts in mathematics*. Springer, 2017. doi:10.1007/978-3-662-53622-3.
- 16 Martin Doucha and Jan Kratochvíl. Cluster vertex deletion: A parameterization between vertex cover and clique-width. In *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012*, volume 7464 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2012. doi:10.1007/978-3-642-32589-2_32.
- 17 Markus Sortland Dregi and Daniel Lokshtanov. Parameterized complexity of bandwidth on trees. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014*, volume 8572 of *Lecture Notes in Computer Science*, pages 405–416. Springer, 2014. doi:10.1007/978-3-662-43948-7_34.

- 18 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Hadas Shachnai. Tractable parameterizations for the minimum linear arrangement problem. *ACM Trans. Comput. Theory*, 8(2):6:1–6:12, 2016. doi:10.1145/2898352.
- 19 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Matthias Mnich, Frances A. Rosamond, and Saket Saurabh. The complexity ecology of parameters: An illustration using bounded max leaf number. *Theory Comput. Syst.*, 45(4):822–848, 2009. doi:10.1007/s00224-009-9167-9.
- 20 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In *Algorithms and Computation, 19th International Symposium, ISAAC 2008*, volume 5369 of *Lecture Notes in Computer Science*, pages 294–305. Springer, 2008. doi:10.1007/978-3-540-92182-0_28.
- 21 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7:49–65, 1987. doi:10.1007/BF02579200.
- 22 Martin Fürer, Serge Gaspers, and Shiva Prasad Kasiviswanathan. An exponential time 2-approximation algorithm for bandwidth. *Theor. Comput. Sci.*, 511:23–31, 2013. doi:10.1016/j.tcs.2013.03.024.
- 23 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013. doi:10.1007/978-3-319-03898-8_15.
- 24 Robert Ganian. Twin-cover: Beyond vertex cover in parameterized algorithms. In *Parameterized and Exact Computation - 6th International Symposium, IPEC 2011*, volume 7112 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2011. doi:10.1007/978-3-642-28050-4_21.
- 25 Robert Ganian, Petr Hliněný, Jaroslav Nešetřil, Jan Obdržálek, Patrice Ossona de Mendez, and Reshma Ramadurai. When trees grow low: Shrubs and fast MSO1. In *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012*, volume 7464 of *Lecture Notes in Computer Science*, pages 419–430. Springer, 2012. doi:10.1007/978-3-642-32589-2_38.
- 26 Robert Ganian, Eun Jung Kim, and Stefan Szeider. Algorithmic applications of tree-cut width. *SIAM J. Discret. Math.*, 36(4):2635–2666, 2022. doi:10.1137/20m137478x.
- 27 M. R. Garey, R. L. Graham, D. S. Johnson, and D. E. Knuth. Complexity results for bandwidth minimization. *SIAM Journal on Applied Mathematics*, 34(3):477–495, 1978. doi:10.1137/0134037.
- 28 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theor. Comput. Sci.*, 918:60–76, 2022. doi:10.1016/j.tcs.2022.03.021.
- 29 Eitan M. Gurari and Ivan Hal Sudborough. Improved dynamic programming algorithms for bandwidth minimization and the mincut linear arrangement problem. *J. Algorithms*, 5(4):531–546, 1984. doi:10.1016/0196-6774(84)90006-3.
- 30 L. H. Harper. Optimal assignments of numbers to vertices. *Journal of the Society for Industrial and Applied Mathematics*, 12(1):131–135, 1964. doi:10.1137/0112012.
- 31 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *J. Comput. Syst. Sci.*, 79(1):39–49, 2013. doi:10.1016/j.jcss.2012.04.004.
- 32 Ravi Kannan. Minkowski’s convex body theorem and integer programming. *Math. Oper. Res.*, 12:415–440, 1987. doi:10.1287/moor.12.3.415.
- 33 Anjeneya Swami Kare and I. Vinod Reddy. Parameterized algorithms for graph burning problem. In *Combinatorial Algorithms - 30th International Workshop, IWOCA 2019*, volume 11638 of *Lecture Notes in Computer Science*, pages 304–314. Springer, 2019. doi:10.1007/978-3-030-25005-8_25.
- 34 Martin Kucera and Ondrej Suchý. Minimum eccentricity shortest path problem with respect to structural parameters. *Algorithmica*, 85(3):762–782, 2023. doi:10.1007/s00453-022-01006-x.

- 35 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. In *Algorithms - ESA 2010, 18th Annual European Symposium*, volume 6346 of *Lecture Notes in Computer Science*, pages 549–560. Springer, 2010. doi:10.1007/978-3-642-15775-2_47.
- 36 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.
- 37 Daniel Lokshtanov. Parameterized integer quadratic programming: Variables and coefficients. *CoRR*, abs/1511.00310, 2015. arXiv:1511.00310.
- 38 Diptapriyo Majumdar and Venkatesh Raman. FPT algorithms for FVS parameterized by split and cluster vertex deletion sets and other parameters. In *Frontiers in Algorithmics - 11th International Workshop, FAW 2017*, volume 10336 of *Lecture Notes in Computer Science*, pages 209–220. Springer, 2017. doi:10.1007/978-3-319-59605-1_19.
- 39 Neeldhara Misra and Harshil Mittal. Imbalance parameterized by twin cover revisited. *Theor. Comput. Sci.*, 895:1–15, 2021. doi:10.1016/j.tcs.2021.09.017.
- 40 Burkhard Monien. The bandwidth minimization problem for caterpillars with hair length 3 is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 7(4):505–512, 1986. doi:10.1137/0607057.
- 41 David Muradian. The bandwidth minimization problem for cyclic caterpillars with hair length 1 is np-complete. *Theor. Comput. Sci.*, 307(3):567–572, 2003. doi:10.1016/S0304-3975(03)00238-X.
- 42 Jaroslav Nešetřil and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.
- 43 Christos H. Papadimitriou. The np-completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270, 1976. doi:10.1007/BF02280884.
- 44 James B. Saxe. Dynamic-programming algorithms for recognizing small-bandwidth graphs in polynomial time. *SIAM J. Algebraic Discret. Methods*, 1(4):363–369, 1980. doi:10.1137/0601042.

Collective Graph Exploration Parameterized by Vertex Cover

Siddharth Gupta  

BITS Pilani, Goa Campus, India

Guy Sa'ar 

Ben Gurion University of the Negev, Beersheba, Israel

Meirav Zehavi  

Ben Gurion University of the Negev, Beersheba, Israel

Abstract

We initiate the study of the parameterized complexity of the COLLECTIVE GRAPH EXPLORATION (CGE) problem. In CGE, the input consists of an undirected connected graph G and a collection of k robots, initially placed at the same vertex r of G , and each one of them has an energy budget of B . The objective is to decide whether G can be *explored* by the k robots in B time steps, i.e., there exist k closed walks in G , one corresponding to each robot, such that every edge is covered by at least one walk, every walk starts and ends at the vertex r , and the maximum length of any walk is at most B . Unfortunately, this problem is NP-hard even on trees [Fraigniaud *et al.*, 2006]. Further, we prove that the problem remains W[1]-hard parameterized by k even for trees of treedepth 3. Due to the para-NP-hardness of the problem parameterized by treedepth, and motivated by real-world scenarios, we study the parameterized complexity of the problem parameterized by the vertex cover number (vc) of the graph, and prove that the problem is fixed-parameter tractable (FPT) parameterized by vc. Additionally, we study the optimization version of CGE, where we want to optimize B , and design an approximation algorithm with an additive approximation factor of $O(\text{vc})$.

2012 ACM Subject Classification Theory of computation \rightarrow Fixed parameter tractability; Theory of computation \rightarrow Approximation algorithms analysis

Keywords and phrases Collective Graph Exploration, Parameterized Complexity, Approximation Algorithm, Vertex Cover, Treedepth

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.22

Related Version *Full Version*: <https://arxiv.org/abs/2310.05480> [13]

Funding *Siddharth Gupta*: Supported by Engineering and Physical Sciences Research Council (EPSRC) grant EP/V007793/1.

Guy Sa'ar: Supported in part by the Israeli Smart Transportation Research Center and by the Lynne and William Frankel Center for Computing Science at Ben-Gurion University.

Meirav Zehavi: Supported by the European Research Council (ERC) grant titled PARAPATH.

1 Introduction

COLLECTIVE GRAPH EXPLORATION (CGE) is a well-studied problem in computer science and robotics, with various real-world applications such as network management and fault reporting, pickup and delivery services, searching a network, and so on. The problem is formulated as follows: given a set of robots (or agents) that are initially located at a vertex of an undirected graph, the objective is to explore the graph as quickly as possible and return to the initial vertex. A graph is *explored* if each of its edges is visited by at least one robot. In each time step, every robot may move along an edge that is incident to the vertex it is placed



© Siddharth Gupta, Guy Sa'ar, and Meirav Zehavi;
licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 22; pp. 22:1–22:18

Leibniz International Proceedings in Informatics



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

at. The total time taken by a robot is the number of edges it traverses. The exploration time is the maximum time taken by any robot. In many real-world scenarios, the robots have limited energy resources, which motivates the minimization of the exploration time [6].

The CGE problem can be studied in two settings: *offline* and *online*. In the offline setting, the graph is known to the robots beforehand, while in the online setting, the graph is unknown and revealed incrementally as the robots explore it. While CGE has received considerable attention in the online setting, much less is known in the offline setting (Section 1.1). Furthermore, most of the existing results in the offline setting are restricted to trees. Therefore, in this paper, we investigate the CGE problem in the offline setting for general graphs, and present some approximation and parameterized algorithms with respect to the vertex cover number of the graph.

1.1 Related Works

As previously mentioned, the CGE problem is extensively studied in the online setting, where the input graph is unknown. As we study the problem in the offline setting in this paper, we only give a brief overview of the results in the online setting, followed by the results in the offline setting.

Recall that, in the online setting, the graph is unknown to the robots and the edges are revealed to a robot once the robot reaches a vertex incident to the edge. The usual approach to analyze any online algorithm is to compute its *competitive ratio*, which is the worst-case ratio between the cost of the online and the optimal offline algorithm. Therefore, the first algorithms for CGE focused on the competitive ratios of the algorithms. In [11], an algorithm for CGE for trees with competitive ratio $O(\frac{k}{\log k})$ was given. Later in [14], it was shown that this competitive ratio is tight. Another line of work studied the competitive ratio as a function of the vertices and the depth of the input tree [4, 7, 8, 10, 14, 19]. We refer the interested readers to a recent paper by Cosson *et al.* [5] and the references within for an in-depth discussion about the results in the online setting.

We now discuss the results in the offline setting. In [1], it was shown that the CGE problem for edge-weighted trees is NP-hard even for two robots. In [2, 18], an $(2 - 2/(k + 1))$ -approximation was given for the optimization version of CGE for edge-weighted trees where we want to optimize B . In [11], the NP-hardness was shown for CGE for unweighted trees as well. In [9], a 2-approximation was given for the optimization version of CGE for unweighted trees where we want to optimize B . In the same paper, it was shown that the optimization version of the problem for unweighted trees is XP parameterized by the number of robots.

1.2 Our Contribution and Methods

In this paper, we initiate the study of the CGE problem for general unweighted graphs in the offline setting and obtain the following three results. We first prove that CGE is FPT parameterized by vc , where vc is the vertex cover number of the input graph. Specifically, we prove the following theorem.

► **Theorem 1.1.** *CGE is in FPT parameterized by $\text{vc}(G)$, where G is the input graph.*

We then study the optimization version of CGE where we want to optimize B and design an approximation algorithm with an additive approximation factor of $O(\text{vc})$. Specifically, we prove the following theorem.

► **Theorem 1.2.** *There exists an approximation algorithm for CGE that runs in time $\mathcal{O}((|V(G)| + |E(G)|) \cdot k)$, and returns a solution with an additive approximation of $8 \cdot \text{vc}(G)$, where G is the input graph and k is the number of robots.*

Finally, we show a border of (in-)tractability by proving that CGE is $W[1]$ -hard parameterized by k , even for trees of treedepth 3. Specifically, we prove the following theorem.

► **Theorem 1.3.** *CGE is $W[1]$ -hard with respect to k even on trees whose treedepth is bounded by 3.*

We first give an equivalent formulation of CGE based on Eulerian cycles (see Lemma 3.4). We obtain the FPT result by using Integer Linear Programming (ILP). By exploiting the properties of vertex cover and the conditions given by our formulation, we show that a potential solution can be encoded by a set of variables whose size is bounded by a function of vertex cover.

To design the approximation algorithm, we give a greedy algorithm that satisfies the conditions given by our formulation. Again, by exploiting the properties of vertex cover, we show that we can satisfy the conditions of our formulation by making optimal decisions at the independent set vertices and using approximation only at the vertex cover vertices.

To prove the W -hardness, we give a reduction from a variant of BIN PACKING, called EXACT BIN PACKING (defined in Section 2). We first prove that EXACT BIN PACKING is $W[1]$ -hard even when the input is given in unary. We then give a reduction from this problem to CGE to obtain our result. Due to lack of space, several concepts and proofs are deferred to the full version of this paper [13].

1.3 Choice of Parameter

As mentioned in the previous section, we proved that CGE is $W[1]$ -hard parameterized by k even on trees of treedepth 3. This implies that we cannot get an FPT algorithm parameterized by treedepth and k even on trees, unless $\text{FPT} = W[1]$. Thus, we study the problem parameterized by the vertex cover number of the input graph, a slightly weaker parameter than the treedepth.

Our choice of parameter is also inspired by several practical applications. For instance, consider a delivery network of a large company. The company has a few major distributors that receive the products from the company and can exchange them among themselves. There are also many minor distributors that obtain the products only from the major ones, as this is more cost-effective. The company employs k delivery persons who are responsible for delivering the products to all the distributors. The delivery persons have to start and end their routes at the company location. Since each delivery person has a maximum working time limit, the company wants to minimize the maximum delivery time among them. This problem can be modeled as an instance of CGE by constructing a graph G that has a vertex for the company and for each distributor and has an edge between every pair of vertices that correspond to locations that can be reached by a delivery person. The k robots represent the k delivery persons and are placed at the vertex corresponding to the company. Clearly, G has a small vertex cover, as the number of major distributors is much smaller than the total number of distributors.

For another real-world example where the vertex cover is small, suppose we want to cover all the streets of the city as fast as possible using k agents that start and end at a specific street. The city has a few long streets and many short streets that connect to them. This situation is common in many urban areas. We can represent this problem as an instance

of CGE by creating a graph G that has a vertex for each street and an edge between two vertices if the corresponding streets are adjacent. The k robots correspond to the k agents. Clearly, G has a small vertex cover, as the number of long streets is much smaller than the total number of streets.

2 Preliminaries

For $k \in \mathbb{N}$, let $[k]$ denote the set $\{1, 2, \dots, k\}$. For a multigraph G , we denote the set of vertices of G and the multiset of edges of G by $V(G)$ and $E(G)$, respectively. For $u \in V(G)$, the *set of neighbors* of u in G is $N_G(u) = \{v \in V \mid \{u, v\} \in E(G)\}$. When G is clear from the context, we refer to $N_G(u)$ as $N(u)$. The *multiset of neighbors* of u in G is the multiset $\hat{N}_G(u) = \{v \in V \mid \{u, v\} \in E(G)\}$ (with repetition). When G is clear from the context, we refer to $\hat{N}_G(u)$ as $\hat{N}(u)$. The *degree* of u in G is $|\hat{N}_G(u)|$ (including repetitions). Let \hat{E} be a multiset with elements from $E(G)$. Let $\text{Graph}(\hat{E})$ denote the multigraph (V', \hat{E}) , where $V' = \{u \mid \{u, v\} \in \hat{E}\}$. A multigraph H is a *submultigraph* of a multigraph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. Let $V' \subseteq V(G)$. We denote the submultigraph induced by V' by $G[V']$, that is, $V(G[V']) = V'$ and $E(G[V']) = \{\{u, v\} \in E(G) \mid u, v \in V'\}$. Let $U \subseteq V(G)$. Let $G \setminus U$ denote the subgraph $G[V(G) \setminus U]$ of G .

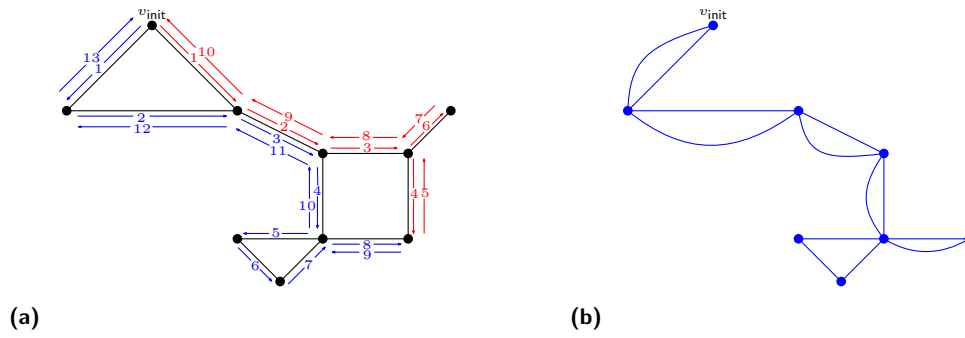
An *Eulerian cycle* in a multigraph \hat{G} is a cycle that visits every edge in $E(\hat{G})$ exactly once. A *vertex cover* of G is $V' \subseteq V(G)$ such that for every $\{u, v\} \in E(G)$, at least one among u and v is in V' . The *vertex cover number* of G is $\text{vc}(G) = \min\{|V'| \mid V' \text{ is a vertex cover of } G\}$. When G is clear from context, we refer to $\text{vc}(G)$ as vc . A *path* P in G is (v_0, \dots, v_ℓ) , where (i) for every $0 \leq i \leq \ell$, $v_i \in V(G)$, and (ii) for every $0 \leq i \leq \ell - 1$, $\{v_i, v_{i+1}\} \in E(G)$ (we allow repeating vertices). The *length* of a path $P = (v_0, \dots, v_\ell)$, denoted by $|P|$, is the number of edges in P (including repetitions), that is, ℓ . The set of vertices of P is $V(P) = \{v_0, \dots, v_{\ell-1}\}$. The multiset of edges of P is $E(P) = \{\{v_i, v_{i+1}\} \mid 0 \leq i \leq \ell - 1\}$ (including repetitions). A *cycle* C in G is a path (v_0, \dots, v_ℓ) such that $v_0 = v_\ell$. A *simple cycle* is a cycle $C = (v_0, \dots, v_\ell)$ such that for every $0 \leq i < j \leq \ell - 1$, $v_i \neq v_j$. An *isomorphism* of a multigraph G into a multigraph G' is a bijection $\alpha : V(G) \rightarrow V(G')$, such that $\{u, v\}$ appears in $E(G)$ ℓ times if and only if $\{\alpha(u), \alpha(v)\}$ appears in $E(G')$ ℓ times, for an $\ell \in \mathbb{N}$. For a multiset A , we denote by 2^A the *power set* of A , that is, $2^A = \{B \mid B \subseteq A\}$. Let A and B be two multisets. Let $A \setminus B$ be the multiset $D \subseteq A$ such that every $d \in A$ appears exactly $\max\{0, d_A - d_B\}$ times in D , where d_A and d_B are the numbers of times d appears in A and B , respectively. A *permutation* of a multiset A is a bijection $\text{Permut}_A : A \rightarrow [|A|]$.

► **Definition 2.1 (*v_{init}-Robot Cycle*).** Let G be a graph, let $v_{\text{init}} \in V(G)$. A *v_{init}-robot cycle* is a cycle $\text{RC} = (v_0 = v_{\text{init}}, v_1, v_2, \dots, v_\ell = v_{\text{init}})$ in G for some ℓ .

When v_{init} is clear from the context, we refer to a *v_{init}-robot cycle* as a *robot cycle*.

► **Definition 2.2 (*Solution*).** Let G be a graph, $v_{\text{init}} \in V(G)$ and $k \in \mathbb{N}$. A *solution* for (G, v_{init}, k) is a set of k *v_{init}-robot cycles* $\{\text{RC}_1, \dots, \text{RC}_k\}$ with $E(G) \subseteq E(\text{RC}_1) \cup E(\text{RC}_2) \cup \dots \cup E(\text{RC}_k)$. Its value is $\text{val}(\{\text{RC}_1, \dots, \text{RC}_k\}) = \max\{|E(\text{RC}_1)|, |E(\text{RC}_2)|, \dots, |E(\text{RC}_k)|\}$ (see Figure 1a for an illustration).

► **Definition 2.3 (*Collective Graph Exploration with k Agents*).** The *COLLECTIVE GRAPH EXPLORATION (CGE) problem with k agents* is: given a connected graph G , $v_{\text{init}} \in V(G)$ and $k \in \mathbb{N}$, find the minimum B such that there exists a solution $\{\text{RC}_1, \dots, \text{RC}_k\}$ where $\text{val}(\{\text{RC}_1, \dots, \text{RC}_k\}) = B$.



■ **Figure 1** (a) An illustration of a graph G (drawn in black) and a solution for $(G, v_{\text{init}}, k = 2)$. The 2 robot cycles are shown by red and blue edges where the edge labels show the order in which the edges were covered by the respective robots. (b) The Robot Cycle-Graph for the robot cycle drawn in blue.

► **Definition 2.4 (Collective Graph Exploration with k Agents and Budget B).** *The COLLECTIVE GRAPH EXPLORATION (CGE) problem with k agents and budget B is: given a connected graph G , $v_{\text{init}} \in V(G)$ and $k, B \in \mathbb{N}$, find a solution $\{\text{RC}_1, \dots, \text{RC}_k\}$ where $\text{val}(\{\text{RC}_1, \dots, \text{RC}_k\}) \leq B$, if such a solution exists; otherwise, return “no-instance”.*

► **Definition 2.5 (Bin Packing).** *The BIN PACKING problem is: given a finite set I of items, a size $s(i) \in \mathbb{N}$ for each $i \in I$, a positive integer B called bin capacity and a positive integer k , decide whether there is a partition of I into disjoint sets I_1, \dots, I_k such that for every $1 \leq j \leq k$, $\sum_{i \in I_j} s(i) \leq B$.*

► **Definition 2.6 (Exact Bin Packing).** *The EXACT BIN PACKING problem is: given a finite set I of items, a size $s(i) \in \mathbb{N}$ for each $i \in I$, a positive integer B called bin capacity and a positive integer k such that $\sum_{i \in I} s(i) = B \cdot k$, decide whether there is a partition of I into disjoint sets I_1, \dots, I_k such that for every $1 \leq j \leq k$, $\sum_{i \in I_j} s(i) = B$.*

► **Definition 2.7 (Integer Linear Programming).** *In the INTEGER LINEAR PROGRAMMING FEASIBILITY (ILP) problem, the input consists of t variables x_1, x_2, \dots, x_t and a set of m inequalities of the following form:*

$$\begin{array}{cccc} a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,p}x_t & \leq & b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \dots + a_{2,p}x_t & \leq & b_2 \\ \vdots & & \vdots \\ a_{m,1}x_1 + a_{m,2}x_2 + \dots + a_{m,p}x_t & \leq & b_m \end{array}$$

where all coefficients $a_{i,j}$ and b_i are required to integers. The task is to check whether there exist integer values for every variable x_i so that all inequalities are satisfiable.

► **Theorem 2.8** ([12, 16, 17]). *An ILP instance of size m with t variables can be solved in time $t^{\mathcal{O}(t)} \cdot m^{\mathcal{O}(1)}$.*

3 Reinterpretation Based on Eulerian Cycles

Our approach to CGE with k agents is as follows. Let G be a connected graph, let $v_{\text{init}} \in V(G)$ and let $k \in \mathbb{N}$. Let $\{\text{RC}_1, \dots, \text{RC}_k\}$ be a solution, let $1 \leq i \leq k$ and denote $\text{RC}_i = (v_0 = v_{\text{init}}, v_1, v_2, \dots, v_\ell = v_{\text{init}})$ for some $\ell \in \mathbb{N}$. If we define a multiset $\widehat{E}_{\text{RC}_i} = \{\{v_j, v_{j+1}\} \mid 0 \leq j \leq \ell - 1\}$, then, clearly, $\text{RC}_i = (v_0 = v_{\text{init}}, v_1, v_2, \dots, v_\ell = v_{\text{init}})$ is an Eulerian cycle in $\text{Graph}(\widehat{E}_{\text{RC}_i})$. We call this graph the RC_i -graph (see Figure 1b):

► **Definition 3.1 (Robot Cycle-Graph).** *Let G be a graph, let $v_{\text{init}} \in V(G)$ and let $\text{RC} = (v_0 = v_{\text{init}}, v_1, v_2, \dots, v_\ell = v_{\text{init}})$ be a robot cycle. The RC-graph, denoted by $\text{Graph}(\text{RC})$, is the multigraph $\text{Graph}(\widehat{E}_{\text{RC}})$, where $\widehat{E}_{\text{RC}} = \{\{v_i, v_{i+1}\} \mid 0 \leq i \leq \ell - 1\}$ is a multiset.*

► **Observation 3.2.** *Let G be a graph, let $v_{\text{init}} \in V(G)$ and let $\text{RC} = (v_0 = v_{\text{init}}, v_1, v_2, \dots, v_\ell = v_{\text{init}})$ be a robot cycle. Then RC is an Eulerian cycle in $\text{Graph}(\text{RC})$.*

On the opposite direction, let \widehat{E} be a multiset with elements from $E(G)$, and assume that $v_{\text{init}} \in V(\text{Graph}(\widehat{E}))$. Let $\text{RC} = (v_0 = v_{\text{init}}, v_1, v_2, \dots, v_\ell = v_0)$ be an Eulerian cycle in $\text{Graph}(\widehat{E})$ and assume, without loss of generality, that $v_0 = v_\ell = v_{\text{init}}$. It is easy to see that RC is a robot cycle in G :

► **Observation 3.3.** *Let G be a graph, let $v_{\text{init}} \in V(G)$, let \widehat{E} be a multiset with elements from $E(G)$ and assume that $v_{\text{init}} \in V(\text{Graph}(\widehat{E}))$. Let $\text{RC} = (v_0 = v_{\text{init}}, v_1, v_2, \dots, v_\ell = v_{\text{init}})$ be an Eulerian cycle in $\text{Graph}(\widehat{E})$. Then, RC is a robot cycle in G .*

From Observations 3.2 and 3.3, we get that finding a solution is equal to find k multisets $\widehat{E}_1, \dots, \widehat{E}_k$ such that: (i) for every $1 \leq i \leq k$, $v_{\text{init}} \in V(\text{Graph}(\widehat{E}_i))$ (ii) for every $1 \leq i \leq k$, there exists an Eulerian cycle in $\text{Graph}(\widehat{E}_i)$ and (iii) $E(G) \subseteq \widehat{E}_1 \cup \dots \cup \widehat{E}_k$, that is, each $e \in E$ appears at least once in at least one of $\widehat{E}_1, \dots, \widehat{E}_k$.

Recall that, in a multigraph \widehat{G} , there exists an Eulerian cycle if and only if \widehat{G} is connected and each $v \in V(\widehat{G})$ has even degree in \widehat{G} [3]. Thus, we have the following lemma:

► **Lemma 3.4.** *Let G be a connected graph, let $v_{\text{init}} \in V(G)$ and let $k, B \in \mathbb{N}$. Then, $(G, v_{\text{init}}, k, B)$ is a yes-instance of CGE if and only if there exist k multisets $\widehat{E}_1, \dots, \widehat{E}_k$ with elements from $E(G)$, such that the following conditions hold:*

1. *For every $1 \leq i \leq k$, $v_{\text{init}} \in V(\text{Graph}(\widehat{E}_i))$.*
2. *For every $1 \leq i \leq k$, $\text{Graph}(\widehat{E}_i)$ is connected, and every vertex in $\text{Graph}(\widehat{E}_i)$ has even degree.*
3. *$E(G) \subseteq \widehat{E}_1 \cup \dots \cup \widehat{E}_k$.*
4. *$\max\{|\widehat{E}_1|, \dots, |\widehat{E}_k|\} \leq B$.*

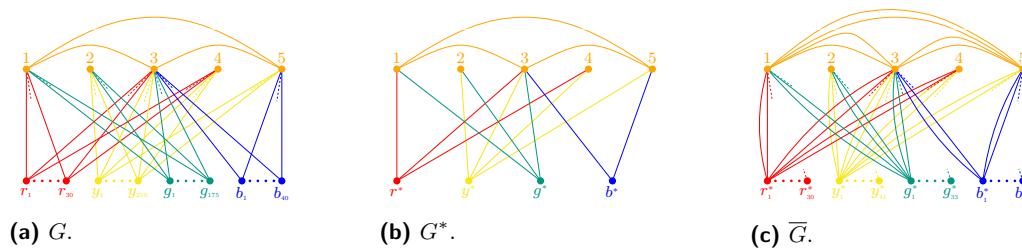
4 High-Level Overview

4.1 FPT Algorithm with Respect to Vertex Cover

Our algorithm is based on a reduction to the ILP problem. We aim to construct linear equations that verify the conditions in Lemma 3.4.

4.1.1 Encoding \widehat{E}_i by a Valid Pair

First, we aim to satisfy the “local” conditions of Lemma 3.4 for each robot, that is, Conditions 1 and 2. Let us focus on the “harder” condition of the two, that is, Condition 2. We aim to encode any potential \widehat{E}_i by smaller subsets whose union is \widehat{E}_i . In addition, we would like the



■ **Figure 2** An illustration of a graph G (in (a)), and its corresponding graphs G^* (in (b)) and \overline{G} (in (c)). The vertex cover vertices and their edges are shown in orange. The 4 equivalence classes and their vertices are shown by red, yellow, green, and blue.

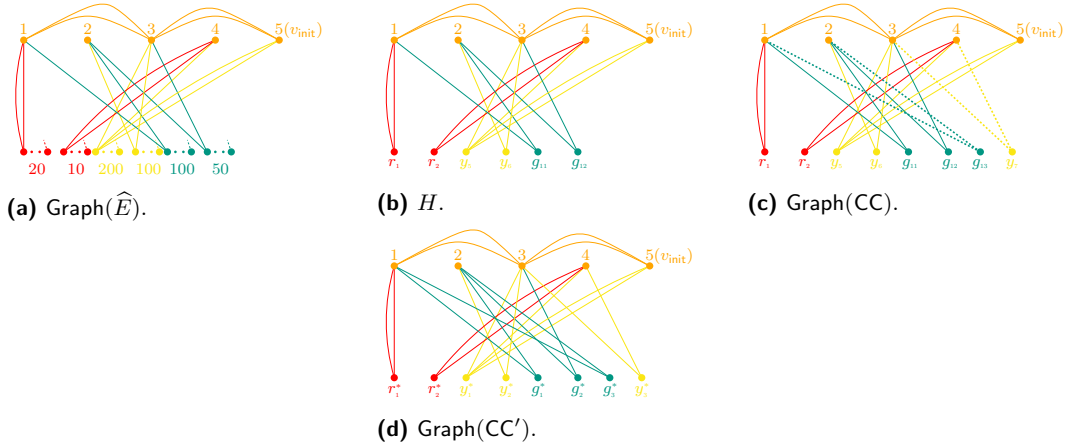
“reverse” direction as well: every collection of subsets that we will be able to unite must create some valid \widehat{E}_i . Note that we have two goals to achieve when uniting the subsets together: (i) derive a connected graph, where (ii) each vertex has even degree. In the light of this, the most natural encoding for the subsets are cycles, being the simplest graphs satisfying both aforementioned goals. Indeed, every cycle is connected, and a graph composed only of cycles is a graph where every vertex has even degree. Here, the difficulty is to maintain the connectivity of the composed graph. On the positive side, observe that every cycle in the input graph G has a non-empty intersection with any vertex cover VC of G . So, we deal with the connectivity requirement as follows. We seek for a graph \overline{G} that is essentially (but not precisely) a subgraph of G that is (i) “small” enough, and (ii) for every valid \widehat{E}_i , there exists $\text{CC} \subseteq E(\overline{G})$ such that $\text{Graph}(\text{CC})$ is a “submultigraph” of $\text{Graph}(\widehat{E}_i)$, $\text{Graph}(\text{CC})$ is connected, and $V(\text{Graph}(\text{CC})) \cap \text{VC} = V(\text{Graph}(\widehat{E}_i)) \cap \text{VC}$.

Equivalence Graph G^* . A first attempt to find such a graph is as follows. We define an equivalence relation on $V(G) \setminus \text{VC}$ based on the sets of neighbors of the vertices in $V(G) \setminus \text{VC}$ (see the 4 equivalence classes of the graph G in Figure 2a). We denote the set of equivalence classes induced by this equivalence relation by EQ . Then G^* is the graph defined as follows.

► **Definition 4.1 (Equivalence Graph G^*).** Let G^* be the graph that: (i) contains VC , and the edges having both endpoints in VC , and (ii) where every equivalence class $u^* \in \text{EQ}$ is represented by a single vertex adjacent to the neighbors of some $u \in u^*$ in G (which belong to VC). See Figure 2b.

Unfortunately, this attempt fails, as we might need to use more than one vertex from the same $u^* \in \text{EQ}$ in order to maintain the connectivity. E.g., see Figure 3b. If we delete r_2 and y_5 , which are in the same equivalence class (in G) as r_1 and y_6 , respectively, then the graph is no longer connected.

The Multigraph \overline{G} . So, consider the following second attempt. We use the aforementioned graph G^* , but instead of one vertex representing each $u^* \in \text{EQ}$, we have $\min\{|u^*|, 2^{|N_{G^*}(u^*)|}\}$ vertices. Observe that given a connected subgraph G' of G , and two vertices $u, u' \in u^*$ such that $N_{G'}(u) = N_{G'}(u')$, it holds that $G \setminus \{u\}$ remains connected (e.g., see Figure 3a and 3b). The connectivity is still maintained even after deleting all but one vertex in the same equivalence class (in G) having same neighbourhood). Therefore, we have enough vertices for each $u^* \in \text{EQ}$ in the graph, and its size is a function of $|\text{VC}|$; so, we obtained the sought graph \overline{G} . Now, we would like to have an additional property for CC , which is that every vertex in $\text{Graph}(\text{CC})$ has even degree in it. To this end, we add to \overline{G} more vertices for each



■ **Figure 3** The graphs shown here are with respect to the graph G shown in Figure 2a. An illustration of (a) a graph $\text{Graph}(\widehat{E})$, (b) the graph H obtained by deleting all but one vertex from the same equivalence class in G and have the same neighbours in $\text{Graph}(\widehat{E})$, (c) the graph $\text{Graph}(\text{CC})$ where CC is a skeleton of \widehat{E} obtained from the graph in (b) by adding four more edges from $\text{Graph}(\widehat{E}) \setminus H$, and (d) the graph $\text{Graph}(\text{CC}')$ where CC' is the skeleton in \overline{G} that is derived from the skeleton CC .

$u^* \in \text{EQ}$. See Figure 3c. The vertex g_{13} having the same neighbours as g_{11} in H and being in the same equivalence class (in G) as g_{11} is added to make the degrees of 1 and 2 even. We have the following definition for \overline{G} .

► **Definition 4.2 (The Multigraph \overline{G}).** Let \overline{G} be the graph that: (i) contains VC , and the edges having both endpoints in VC , (ii) for every equivalence class $u^* \in \text{EQ}$, there are exactly $\min\{|u^*|, 2^{|\text{NG}^*(u^*)|} + |\text{VC}|^2\}$ vertices, adjacent to the neighbors of some $u \in u^*$ in G (which belong to VC), (iii) each edge in \overline{G} appears exactly twice in $E(\overline{G})$ (for technical reasons). See Figure 2c.

A Skeleton of \widehat{E}_i . We think of $\text{Graph}(\text{CC})$ as a “skeleton” of a potential \widehat{E}_i . By adding cycles with a vertex from $V(\text{Graph}(\text{CC})) \cap \text{VC}$, we maintain the connectivity, and since every vertex in $\text{Graph}(\text{CC})$ has even degree, then by adding a cycle, this property is preserved as well. We have the following definition for a skeleton.

► **Definition 4.3 (A Skeleton CC).** A skeleton of \widehat{E}_i is $\text{CC} \subseteq \widehat{E}_i$ such that: (i) $\text{Graph}(\text{CC})$ is a “submultigraph” of \overline{G} , (ii) $\text{Graph}(\text{CC})$ is connected, $v_{\text{init}} \in V(\text{Graph}(\text{CC}))$ and every vertex in $\text{Graph}(\text{CC})$ has even degree, and (iii) $V(\text{Graph}(\text{CC})) \cap \text{VC} = V(\text{Graph}(\widehat{E}_i)) \cap \text{VC}$ (See Figure 3d).

An \widehat{E}_i -Valid Pair. We prove that we might assume that the \widehat{E}_i 's are *nice multisets*, that is, a multiset where every element appears at most twice. We prove that every \widehat{E}_i (assuming \widehat{E}_i is nice) can be encoded by a skeleton CC (See Figure 3c.) and a multiset \mathcal{C} of cycles (of length bounded by $2|\text{VC}|$). We say that (CC, \mathcal{C}) is an \widehat{E}_i -valid pair.

► **Definition 4.4 (A Valid Pair).** A pair (CC, \mathcal{C}) , where CC is a skeleton of \widehat{E}_i and \mathcal{C} is a multiset of cycles in $\text{Graph}(\widehat{E}_i)$, is an \widehat{E}_i -valid pair skeleton CC if:

1. The length of each cycle in \mathcal{C} is bounded by $2|\text{VC}|$.
2. At most $2|\text{VC}|^2$ cycles in \mathcal{C} have length other than 4.
3. $\text{CC} \cup \bigcup_{C \in \mathcal{C}} E(C) = \widehat{E}_i$ (being two multisets).

4.1.2 Robot and Cycle Types

Now, obviously, the number of different cycles in G (of length bounded by $2|\text{VC}|$) is potentially huge. Fortunately, it suffices to look at cycles in G^* in order to preserve Condition 2 of Lemma 3.4: assume that we have a connected $\text{Graph}(\text{CC})$ such that every vertex in $\text{Graph}(\text{CC})$ has even degree in it, and a multiset of cycles with a vertex from $V(\text{Graph}(\text{CC})) \cap \text{VC}$ in G^* . By replacing each vertex that represents $u^* \in \text{EQ}$ by any $u \in u^*$, the connectivity preserved, and the degree of each vertex is even.

Thus, each robot is associated with a *robot type* RobTyp , which includes a skeleton CC of the multiset \widehat{E}_i associated with the robot (along other information discussed later). In order to preserve Condition 1 of Lemma 3.4, we also demand that $v_{\text{init}} \in V(\text{Graph}(\text{CC}))$. Generally, for each type we define, we will have a variable that stands for the number of elements of that type. We are now ready to present our first equation of the ILP reduction:

Equation 1: Robot Type for Each Robot. In this equation, we ensure that the total sum of robots of the different robot types is exactly k , that is, there is exactly one robot type for each robot:

$$1. \quad \sum_{\text{RobTyp} \in \text{RobTypS}} x_{\text{RobTyp}} = k.$$

In addition, the other “pieces” of the “puzzle”, that is, the cycles, are also represented by types: Each cycle C of length at most $2|\text{VC}|$ in G^* is represented by a *cycle type*, of the form $\text{CycTyp} = (C, \text{RobTyp})$ (along other information discussed later), where RobTyp is a robot type that is “able to connect to C ”, that is, $V(\text{Graph}(\text{CC})) \cap \text{VC} \cap V(C) \neq \emptyset$ for $\text{RobTyp} = \text{CC}$. Similarly, we will have equations for our other types.

Satisfying the Budget Restriction. Now, we aim to satisfy the budget condition (Condition 2 of Lemma 3.4), that is, for every $i \in [k]$, $|\widehat{E}_i| \leq B$. Let $i \in [k]$ and let (CC, \mathcal{C}) be an \widehat{E}_i -valid pair. So, $\widehat{E}_i = \text{CC} \cup (\bigcup_{C \in \mathcal{C}} E(C))$ (being a union of two multisets). Now, we prove that “most” of the cycles in \mathcal{C} are of length 4, that is, for every $2 \leq j \leq 2|\text{VC}|$, $j \neq 4$, the number of cycles of length j in \mathcal{C} is bounded by $2|\text{VC}|^2$. Therefore, we add to the definition of a robot type also the number of cycles of length exactly j , encoded by a vector $\text{NumOfCyc} = (N_2, N_3, N_5, N_6, \dots, N_{2|\text{VC}|})$. So, for now, a robot type is $\text{RobTyp} = (\text{CC}, \text{NumOfCyc})$. Thus, in order to satisfy the budget condition, we verify that the budget used by all the robots of a robot type $\text{RobTyp} = (\text{CC}, \text{NumOfCyc})$, is as expected together. First, we ensure that the number of cycles of each length $2 \leq j \leq 2|\text{VC}|$, $j \neq 4$, is exactly as the robot type demands, times the number of robots associated with this type, that is, $N_j \cdot x_{\text{RobTyp}}$. So, we have the following equation:

Equation 5: Assigning the Exact Number of Cycles of Length Other Than 4 to Each Robot Type. We have the following notation: $\text{CycTypS}(\text{RobTyp}, j)$ is the set of cycle types for cycles of length j assigned to a robot of robot type RobTyp .

$$5. \quad \text{For every robot type } \text{RobTyp} = (\text{CC}, \text{NumOfCyc}) \text{ and for every } 2 \leq j \leq 2|\text{VC}|, j \neq 4, \\ \sum_{\text{CycTyp} \in \text{CycTypS}(\text{RobTyp}, j)} x_{\text{CycTyp}} = N_j \cdot x_{\text{RobTyp}}, \text{ where } \text{NumOfCyc} = (N_2, N_3, N_5, N_6, \dots, N_{2|\text{VC}|}).$$

Observe that once this equation is satisfied, we are able to arbitrary allocate N_j cycles of length j to each robot of type RobTyp . So, in order to verify the budget limitation, we only need to deal with the cycles of length 4. Now, notice that the budget left for a robot of type

22:10 Collective Graph Exploration Parameterized by Vertex Cover

$\text{RobTyp} = (\text{CC}, \text{NumOfCyc})$ for the cycles of length 4 is $B - (|\text{CC}| + \sum_{2 \leq j \leq 2|\text{VC}|, j \neq 4} N_j \cdot j)$, where $\text{NumOfCyc} = (N_2, N_3, N_5, N_6, \dots, N_{2|\text{VC}|})$. Now, the maximum number of cycles we can add to a single robot of type RobTyp is the largest number which is a multiple of 4, that is less or equal to $B - \text{Bud}(\text{RobTyp})$. So, for every robot type RobTyp let $\text{CycBud}(\text{RobTyp}) = \lfloor (B - (|\text{CC}| + \sum_{2 \leq j \leq 2|\text{VC}|, j \neq 4} N_j \cdot j)) \cdot \frac{1}{4} \rfloor \cdot 4$. Notice that $\text{CycBud}(\text{RobTyp})$ is the budget left for the cycles of length 4. Thus, we have the following equation:

Equation 6: Verifying the Budget Limitation. This equation is defined as follows.

6. For every $\text{RobTyp} \in \text{RobTypS}$,

$$\sum_{\text{CycTyp} \in \text{CycTypS}(\text{RobTyp}, 4)} 4 \cdot x_{\text{CycTyp}} \leq x_{\text{RobTyp}} \cdot \text{CycBud}(\text{RobTyp}).$$

By now, we have that there exist $\widehat{E}_1, \dots, \widehat{E}_k$ that satisfy Conditions 1, 2 and 4 of Lemma 3.4 if and only if Equations 1, 5 and 6 can be satisfied.

Covering Edges with Both Endpoints in VC. Now, we aim to satisfy Condition 3 of Lemma 3.4, that is, we need to verify that every edge is covered by at least one robot. First, we deal with edges with both endpoints in VC. Here, for every $\{u, v\}$ such that $u, v \in \text{VC}$, we just need to verify that at least one cycle or one of the CC's contains $\{u, v\}$. This we can easily solve by the following equation:

Equation 4: Covering Each Edge With Both Endpoints in VC. We have the following notations: For every $\{u, v\} \in E$ such that $u, v \in \text{VC}$, (i) let $\text{CycTypS}(\{u, v\})$ be the set of cycle types $\text{CycTyp} = (C, \text{RobTyp})$ where C covers $\{u, v\}$, and (ii) let $\text{RobTypS}(\{u, v\})$ be the set of robot types $\text{RobTyp} = (\text{CC}, \text{NumOfCyc})$ where CC covers $\{u, v\}$. In this equation, we ensure that each $\{u, v\} \in E(G)$ with both endpoints in VC is covered at least once:

4. For every $\{u, v\} \in E$ such that $u, v \in \text{VC}$,

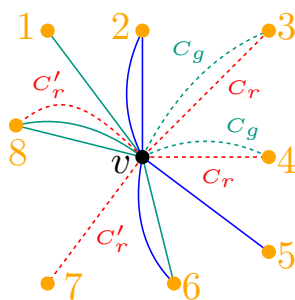
$$\sum_{\text{CycTyp} \in \text{CycTypS}(\{u, v\})} x_{\text{CycTyp}} + \sum_{\text{RobTyp} \in \text{RobTypS}(\{u, v\})} x_{\text{RobTyp}} \geq 1.$$

Let RobTypS be the set of the robot types, and let CycTypS be the set of cycle types.

Covering Edges with an Endpoint in $V(G) \setminus \text{VC}$. Now, we aim to cover the edges from $E(G)$ with (exactly) one endpoint in $V(G) \setminus \text{VC}$. Here, we need to work harder. Let x_z , for every $z \in \text{RobTypS} \cup \text{CycTypS}$, be values that satisfy Equations 1 and 4–6. As for now, we will arbitrarily allocate cycles to robots according to their types. Then, we will replace every $u^* \in V(\text{Graph}(\text{CC}_i))$ and $u^* \in V(C)$, for every cycle C allocated to the i -th robot, by an arbitrary $u \in u^*$. Then, we will define \widehat{E}_i as the union of edge set of the cycles and CC_i we obtained. We saw that due to Equations 1, 5 and 6, Conditions 1, 2 and 4 of Lemma 3.4 are satisfied. In addition, due to Equations 4, we ensure that each $\{u, v\} \in E(G)$ with both endpoints in VC is covered. The change we need to do in order to cover edges with an endpoint in $V(G) \setminus \text{VC}$ is to make a smarter choices for the replacements of u^* vertices.

4.1.3 Vertex Type

Allocation of Multisets with Elements from $\mathbf{N}_{G^*}(u^*)$. Observe that each $u_j^* \in V(\text{Graph}(\text{CC}_i))$ that is replaced by some $u \in u^*$, covers the multiset of edges $\{\{u, v\} \mid v \in \widehat{\mathbf{N}}_{\text{Graph}(\text{CC}_i)}(u_j^*)\}$. In addition, every $u^* \in V(C)$ that is replaced by $u \in u^*$, covers the multiset of edges $\{\{u, v\}, \{u, v'\}\}$, where v and v' are the vertices right before and right after u in



■ **Figure 4** An illustration of the parts of a solution around an independent set vertex v . The three colors represent the parts of the multisets corresponding to three robots. The solid edges belong to the skeleton of the specific robot. The dashed edges belong to a cycle, labelled in the figure, of the multiset of the cycles corresponding to the specific robot. The vertex type of v derived from the solution shown in the figure is $(v^*, \{\{1, 6, 8, 8\}, \{3, 4\}, \{7, 8\}, \{2, 2, 5, 6\}\})$, where $v \in v^* \in \text{EQ}$.

C , respectively. Now, in order to cover every edge with an endpoint in $V(G) \setminus \text{VC}$, we need to cover the set $\{\{u, v\} \mid v \in \text{N}_{G^*}(u^*)\}$ for every $u \in u^* \in \text{EQ}$. Therefore, we would like to ensure that the union of multisets of neighbors “allocated” for each u , when we replace some u^* by u , contains $\{\{u, v\} \mid v \in \text{N}_{G^*}(u^*)\}$.

The Set NeiSubsets of Multisets Needed to Allocate to a Vertex. Now, the reverse direction holds as well: let $\widehat{E}_1, \dots, \widehat{E}_k$ be multisets satisfying the conditions of Lemma 3.4, for every $i \in [k]$, let $(\text{CC}_i, \mathcal{C}_i)$ be an \widehat{E}_i -valid pair, and let $u \in u^* \in \text{EQ}$. Consider the following multisets (*): (i) for every $i \in [k]$ such that $u \in V(\text{Graph}(\text{CC}_i))$, the multiset $\widehat{\text{N}}_{\text{Graph}(\text{CC}_i)}(u)$; (ii) for every $i \in [k]$ and $C \in \mathcal{C}_i$ and every appearance of u in C , the multiset $\{v, v'\}$, where v and v' are the vertices in C right before and right after the appearance of u . By Condition 3 of Lemma 3.4, every edge appears in at least one among $\widehat{E}_1, \dots, \widehat{E}_k$. So, as for every $i \in [k]$, $\widehat{E}_i = \text{CC}_i \cup_{C \in \mathcal{C}_i} E(C)$, the union of the multisets in (*) obviously contains $\text{N}_{G^*}(u^*)$, e.g. see Figure 4. We would like to store the information of these potential multisets that ensures we covered $\text{N}_{G^*}(u^*)$. The issue is that there might be a lot of multisets, as u might appear in many \widehat{E}_i 's. Clearly, it is sufficient to store one copy of each such multiset, as we only care that the union of the multisets contains $\text{N}_{G^*}(u^*)$. Now, as we assume that $\widehat{E}_1, \dots, \widehat{E}_k$ are nice multisets, each element in every multiset we derived appears at most twice in that multiset. In addition, since every edge in $E(\overline{G})$ appears at most twice, for each skeleton $\text{CC} \subseteq E(\overline{G})$, each edge appears at most twice in $E(\text{Graph}(\text{CC}))$. So, for each $u_j^* \in V(\text{Graph}(\text{CC}))$ we replace by some $u \in u^*$, in the multiset of neighbors that are covered, every element appears at most twice. Moreover, since the degree is even, we have that the number of element in each multiset is even.

For a set A we define the multiset $A \times 2 = \{a, a \mid a \in A\}$. That is, each element in A appears exactly twice in $A \times 2$. Thus, we have the following definition for a vertex type.

► **Definition 4.5 (Vertex Type).** *Let G be a connected graph and let VC be a vertex cover of G . Let $u^* \in \text{EQ}$ and let $\text{NeiSubsets} \subseteq 2^{\text{N}_{G^*}(u^*) \times 2}$. Then, $\text{VerTyp} = (u^*, \text{NeiSubsets})$ is a vertex type if for every $\text{NeiSub} \in \text{NeiSubsets}$, $|\text{NeiSub}|$ is even, and $\text{N}_{G^*}(u^*) \subseteq \bigcup \text{NeiSubsets}$.*

Now, given $\widehat{E}_1, \dots, \widehat{E}_k$ satisfying the conditions of Lemma 3.4, for every $i \in [k]$, an \widehat{E}_i -valid pair $(\text{CC}_i, \mathcal{C}_i)$, and $u \in u^* \in \text{EQ}$, we derive the vertex type of u as follows. We take the set NeiSubsets of multisets as described in (*). Clearly, $(u^*, \text{NeiSubsets})$ is a vertex type.

22:12 Collective Graph Exploration Parameterized by Vertex Cover

For the reverse direction, we will use vertex type in order to cover the edges incident to each $u \in u^* \in \text{EQ}$. Let VerTypS be the set of vertex types. We have a variable x_z for every $z \in \text{VerTypS}$. First, each $u \in u^* \in \text{EQ}$ is associated with exactly one vertex type $\text{VerTyp} = (u^*, \text{NeiSubsets})$, for some NeiSubsets . To achieve this, we first ensure that for every $u^* \in \text{EQ}$, the total sum of x_z for $z \in \text{VerTypS}_{u^*}$, is exactly $|u^*|$, where $\text{VerTypS}_{u^*} = \{(u^*, \text{NeiSubsets}) \in \text{VerTypS}\}$.

Equation 2: Vertex Type for Each Vertex. This equation is defined as follows.

$$2. \text{ For every } u^* \in \text{EQ}, \quad \sum_{\text{VerTyp} \in \text{VerTypS}_{u^*}} x_{\text{VerTyp}} = |u^*|$$

Given values for the variables that satisfy the equation, we arbitrary determine a vertex type $(u^*, \text{NeiSubsets})$ for each $u \in u^*$, such that there are exactly x_{VerTyp} vertices of type VerTyp .

Allocation Functions of Multisets to Vertex Types. Now, let $u \in u^* \in \text{EQ}$ of a vertex type $(u^*, \text{NeiSubsets})$. We aim that when we do the replacements of u^* 's by vertices from u^* , each u gets an allocation of at least one of any of the multisets in NeiSubsets . This ensures that we covered all of the edges adjacent to u . Instead of doing this for each $u \in u^* \in \text{EQ}$, we will ensure that each $\text{NeiSub} \in \text{NeiSubsets}$ is allocated for vertices of type $(u^*, \text{NeiSubsets})$ at least x_{VerTyp} times. To this end, we add more information for the robot types. For a robot type with a skeleton CC , recall that we replace each $u_j^* \in V(\text{Graph}(\text{CC}))$ by some $u \in u^*$. The robot type also determines what is the vertex type of u that replaces u_j^* . In particular, we add to the robot type an allocation for each of $\{(u_j^*, \widehat{N}_{\text{Graph}(\text{CC})}(u_j^*)) \mid u_j^* \in V(\text{Graph}(\text{CC}))\}$, that is, a function $\text{Alloc}_{\text{Graph}(\text{CC})}$ from this set into VerTypS (e.g., a robot of a robot type associated with the skeleton illustrated by Figure 3d, needs to allocate the pair $(r_1^*, \{1, 1\})$, along with the other pairs shown in the figure). Observe that u_j^* is the vertex being replaced, and $\widehat{N}_{\text{Graph}(\text{CC})}(u_j^*)$ is the multiset of neighbors that are covered. So, we demand that each $(u_j^*, \widehat{N}_{\text{Graph}(\text{CC})}(u_j^*))$ is allocated to a vertex type $(u^*, \text{NeiSubsets})$ that “wants” to get $\widehat{N}_{\text{Graph}(\text{CC})}(u_j^*)$, that is, $\widehat{N}_{\text{Graph}(\text{CC})}(u_j^*) \in \text{NeiSubsets}$ (e.g, a robot of a robot type associated with the skeleton illustrated by Figure 3d, might allocate $(r_1^*, \{1, 1\})$ to a vertex type $(r^*, \{\{1, 1\}, \{3, 4\}\})$). Now, we are ready to define a robot type as follows.

► **Definition 4.6 (Robot Type).** A robot type is $\text{RobTyp} = (\text{CC}, \text{Alloc}_{\text{Graph}(\text{CC})}, \text{NumOfCyc})$ such that:

1. $\text{CC} \subseteq E(\overline{G})$.
2. $\text{Graph}(\text{CC})$ is connected, every vertex in $\text{Graph}(\text{CC})$ has even degree and $v_{\text{init}} \in V(\text{Graph}(\text{CC}))$.
3. $\text{Alloc}_{\text{Graph}(\text{CC})}$ is an allocation of $\{(u_j^*, \widehat{N}_{\text{Graph}(\text{CC})}(u_j^*)) \mid u_j^* \in V(\text{Graph}(\text{CC}))\}$ to vertex types.
4. $\text{NumOfCyc} = (N_2, N_3, N_5, N_6, \dots, N_{2|\text{VC}|})$, where $0 \leq N_i \leq 2|\text{VC}|^2$ for every $2 \leq i \leq 2|\text{VC}|$, $i \neq 4$.

Similarly, we add to a cycle type with a cycle C in G^* an allocation of the multiset $\{\{v, v'\} \mid u^* \in V(C), v \text{ and } v' \text{ are the vertices appears right before and right after } u^*\}$ to vertex types (given by a function PaAlloc_C). Now, we are ready to define a cycle type as follows.

► **Definition 4.7 (Cycle Type).** Let $C \in \text{Cyc}_{G^*}$, let PaAlloc_C be an allocation of $\{\{v, v'\} \mid u^* \in V(C), v \text{ and } v' \text{ are the vertices appears right before and right after } u^*\}$ to vertex types, and let $\text{RobTyp} = (\text{CC}, \text{Alloc}_{\text{Graph}(\text{CC})}, \text{NumOfCyc})$ be a robot type. Then, $\text{CycTyp} = (C, \text{PaAlloc}_C, \text{RobTyp})$ is a cycle type if $V(\text{Graph}(\text{CC})) \cap V(C) \cap \text{VC} \neq \emptyset$.

We have the following notations.

For every $\text{VerTyp} = (u^*, \text{NeiSubsets}) \in \text{VerTypS}$, every $\text{NeiSub} = \{v, v'\} \in \text{NeiSubsets}$ and $1 \leq j \leq 2|\text{VC}|$, $\text{CycTypS}(\text{VerTyp}, \text{NeiSub}, j)$ is the set of cycle types that assign NeiSub to VerTyp exactly j times. For every $\text{VerTyp} = (u^*, \text{NeiSubsets}) \in \text{VerTypS}$, every $\text{NeiSub} \in \text{NeiSubsets}$ and $1 \leq j \leq 2^{|\text{VC}|} + |\text{VC}|^2$, $\text{RobTypS}(\text{VerTyp}, \text{NeiSub}, j)$ is the set of robot types that assign NeiSub to VerTyp exactly j times. Finally, we have the following equation:

Equation 3: Assigning Enough Subsets for Each Vertex Type. The equation is defined as follows.

3. For every $\text{VerTyp} = (u^*, \text{NeiSubsets}) \in \text{VerTypS}$, and every $\text{NeiSub} \in \text{NeiSubsets}$,

$$\sum_{j=1}^{2|\text{VC}|} \sum_{\text{CycTyp} \in \text{CycTypS}(\text{VerTyp}, \text{NeiSub}, j)} j \cdot x_{\text{CycTyp}} + \sum_{j=1}^{2^{|\text{VC}|} + |\text{VC}|^2} \sum_{\text{RobTyp} \in \text{RobTypS}(\text{VerTyp}, \text{NeiSub}, j)} j \cdot x_{\text{RobTyp}} \geq x_{\text{VerTyp}}.$$

4.1.4 The Correctness of The Reduction

We denote the ILP instance associated with Equations 1–6 by $\text{Reduction}(G, v_{\text{init}}, k, B)$. Now, we give a proof sketch for the correctness of the reduction:

► **Lemma 4.8.** Let G be a connected graph, let $v_{\text{init}} \in V(G)$ and let $k, B \in \mathbb{N}$. Then, $(G, v_{\text{init}}, k, B)$ is a yes-instance of CGE, if and only if $\text{Reduction}(G, v_{\text{init}}, k, B)$ is a yes-instance of the INTEGER LINEAR PROGRAMMING.

Proof. Let x_z , for every $z \in \text{VerTypS} \cup \text{RobTypS} \cup \text{CycTypS}$, be values satisfying Equations 1–6. For every vertex type $\text{VerTyp} = (u^*, \text{NeiSubsets})$ and each $\text{NeiSub} \in \text{NeiSubsets}$, let $\text{Alloc}(\text{VerTyp}, \text{NeiSub})$ be the set of every allocation of NeiSub to VerTyp by cycles or robots. We arbitrary allocate each element in $\text{Alloc}(\text{VerTyp}, \text{NeiSub})$ to a vertex in u^* , such that every vertex $u \in u^*$ of type VerTyp gets at least one allocation. Due to Equation 3, we ensure we can do that. Then, we replace every $u^* \in V(\text{Graph}(\text{CC}_i))$ and every $u^* \in V(C)$ (for every $C \in \mathcal{C}_i$) by the $u \in u^*$ derived by the allocation. This ensures we covered every edge adjacent to a vertex in $V(G) \setminus \text{VC}$. As seen in this overview, the other conditions of Lemma 3.4 hold.

For the reverse direction, let $\widehat{E}_1, \dots, \widehat{E}_k$ be multisets satisfying the conditions of Lemma 3.4. For every $1 \leq i \leq k$ let $(\text{CC}_i, \mathcal{C}_i)$ be an \widehat{E}_i -valid pair. Then, we first derive the vertex type of each $u \in V(G) \setminus \text{VC}$, according to its equivalence class in EQ, and the set of multisets derived from $((\text{CC}_i, \mathcal{C}_i))_{1 \leq i \leq k}$ (e.g. see Figure 4). Then, we derive the robot type $\text{RobTyp} = (\text{CC}, \text{Alloc}_{\text{Graph}(\text{CC})}, \text{NumOfCyc})$ for each $i \in [k]$: (i) the skeleton CC is determined by CC_i (e.g., see Figure 3d), (ii) NumOfCyc is determined by the number of cycle of each length in \mathcal{C}_i and (iii) the allocation of the multisets of $\text{Graph}(\text{CC}_i)$ is determined by the vertex types of $u \in V(\text{Graph}(\text{CC}_i)) \cap (V(G) \setminus \text{VC})$ we have already computed. Then, for every $i \in [k]$ and every $C' \in \mathcal{C}_i$, we determine the cycle type $\text{CycTyp} = (C, \text{PaAlloc}_C, \text{RobTyp})$ of C' : (i) C is determined by C' (we replace each $u \in u^* \in \text{EQ}$ in C' by u^*), (ii) RobTyp is the robot type of i we have already computed, and (iii) PaAlloc_C is determined by the

vertex types of $u \in V(C') \cap (V(G) \setminus \text{VC})$ we have already computed. Then, for every $z \in \text{VerTypS} \cup \text{RobTypS} \cup \text{CycTypS}$, we define x_z to be the number of elements of type z . As seen in this overview, the values of the variables satisfy Equations 1–6. ◀

Observe that the number of variables is bounded by a function of $|\text{VC}|$, so we will get an FPT runtime with respect to vc . Thus, we conclude the correction of Theorem 1.1.

4.2 Approximation Algorithm with Additive Error of $\mathcal{O}(\text{vc})$

Our algorithm is based on a greedy approach. Recall that, our new goal (from Lemma 3.4) is to find k multisets $\widehat{E}_1, \dots, \widehat{E}_k$ such that for every $1 \leq i \leq k$, $v_{\text{init}} \in V(\text{Graph}(\widehat{E}_i))$, $\text{Graph}(\widehat{E}_i)$ is connected and each $u \in V(\text{Graph}(\widehat{E}_i))$ has even degree in $\text{Graph}(\widehat{E}_i)$. Now, assume that we have a vertex cover VC of G such that $G[\text{VC}]$ is connected and $v_{\text{init}} \in \text{VC}$, and let $I = V \setminus \text{VC}$. We first make the degree of every vertex in I even in G , by duplicating an arbitrary edge for vertices having odd degree. Observe that, after these operations, G may be a multigraph.

We initialize $\widehat{E}_1, \dots, \widehat{E}_k$ with k empty sets. We partition the set of edges of G with one endpoint in I in the following manner. We choose the next multiset from $\widehat{E}_1, \dots, \widehat{E}_k$ in a round-robin fashion and put a pair of edges, not considered so far, incident to some vertex $v \in I$, in the multiset. This ensures that the degree of every vertex in I is even in each multiset. Let $\widehat{E}'_1, \dots, \widehat{E}'_k$ be multisets satisfying the conditions of Lemma 3.4. Then, due to Condition 2 of Lemma 3.4, the degree of every vertex is even in every multiset \widehat{E}'_i . Thus, the total number of edges (with repetition) incident to any vertex in $\text{Graph}(\widehat{E}'_1 \cup \dots \cup \widehat{E}'_k)$ is even. Therefore, there must be at least one additional repetition for at least one edge of every vertex with odd degree in G . So, adding an additional edge to each vertex with odd degree is “must” and it does not “exceed” the optimal budget. Then, we partition the edges with both endpoints in VC , in a balanced fashion, as follows. We choose an edge, not considered so far, and add it to a multiset with minimum size.

Observe that, after this step, we have that: i) every edge of the input graph belongs to at least one of the multisets \widehat{E}_i , ii) the degree of each vertex of I in each multiset is even, and iii) we have not exceeded the optimal budget. We still need to ensure that i) $\text{Graph}(\widehat{E}_i)$ is connected, for every $i \in [k]$, ii) the degree of each vertex of VC in each multiset is even, and iii) $v_{\text{init}} \in V(\text{Graph}(\widehat{E}_i))$ for every $i \in [k]$. Next, we add a spanning tree of $G[\text{VC}]$ to each of the \widehat{E}_i , in order to make $\text{Graph}(\widehat{E}_i)$ connected and to ensure that $v_{\text{init}} \in V(\text{Graph}(\widehat{E}_i))$. Lastly, we add at most $|\text{VC}|$ edges, with both endpoints in VC , to every \widehat{E}_i in order to make the degree of each $u \in \text{VC}$ even in each of the multiset. Observe that the multisets $\widehat{E}_1, \dots, \widehat{E}_k$ satisfy the conditions of Lemma 3.4. Moreover, we added at most $\mathcal{O}(|\text{VC}|)$ additional edges to each \widehat{E}_i , comparing to an optimal solution.

References

- 1 Igor Averbakh and Oded Berman. A heuristic with worst-case analysis for minimax routing of two travelling salesmen on a tree. *Discret. Appl. Math.*, 68(1-2):17–32, 1996. doi:10.1016/0166-218X(95)00054-U.
- 2 Igor Averbakh and Oded Berman. $(p - 1)/(p + 1)$ -approximate algorithms for p -traveling salesmen problems on a tree with minmax objective. *Discret. Appl. Math.*, 75(3):201–216, 1997. doi:10.1016/S0166-218X(97)89161-5.
- 3 Béla Bollobás. *Modern graph theory*, volume 184. Springer Science & Business Media, 1998.
- 4 Peter Brass, Flavio Cabrera-Mora, Andrea Gasparri, and Jizhong Xiao. Multirobot tree and graph exploration. *IEEE Trans. Robotics*, 27(4):707–717, 2011. doi:10.1109/TR0.2011.2121170.

- 5 Romain Cosson, Laurent Massoulié, and Laurent Viennot. Breadth-first depth-next: Optimal collaborative exploration of trees with low diameter. *CoRR*, abs/2301.13307, 2023. doi:10.48550/arXiv.2301.13307.
- 6 Shantanu Das, Dariusz Dereniowski, and Christina Karousatou. Collaborative exploration of trees by energy-constrained mobile robots. *Theory Comput. Syst.*, 62(5):1223–1240, 2018. doi:10.1007/s00224-017-9816-3.
- 7 Dariusz Dereniowski, Yann Disser, Adrian Kosowski, Dominik Pajak, and Przemyslaw Uznanski. Fast collaborative graph exploration. *Inf. Comput.*, 243:37–49, 2015. doi:10.1016/j.ic.2014.12.005.
- 8 Yann Disser, Frank Mousset, Andreas Noever, Nemanja Skoric, and Angelika Steger. A general lower bound for collaborative tree exploration. *Theor. Comput. Sci.*, 811:70–78, 2020. doi:10.1016/j.tcs.2018.03.006.
- 9 Miroslaw Dynia, Miroslaw Korzeniowski, and Christian Schindelhauer. Power-aware collective tree exploration. In Werner Grass, Bernhard Sick, and Klaus Waldschmidt, editors, *Architecture of Computing Systems - ARCS 2006, 19th International Conference, Frankfurt/Main, Germany, March 13-16, 2006, Proceedings*, volume 3894 of *Lecture Notes in Computer Science*, pages 341–351. Springer, 2006. doi:10.1007/11682127_24.
- 10 Miroslaw Dynia, Jaroslaw Kutylowski, Friedhelm Meyer auf der Heide, and Christian Schindelhauer. Smart robot teams exploring sparse trees. In Rastislav Kralovic and Pawel Urzyczyn, editors, *Mathematical Foundations of Computer Science 2006, 31st International Symposium, MFCS 2006, Stará Lesná, Slovakia, August 28-September 1, 2006, Proceedings*, volume 4162 of *Lecture Notes in Computer Science*, pages 327–338. Springer, 2006. doi:10.1007/11821069_29.
- 11 Pierre Fraigniaud, Leszek Gasieniec, Dariusz R. Kowalski, and Andrzej Pelc. Collective tree exploration. *Networks*, 48(3):166–177, 2006. doi:10.1002/net.20127.
- 12 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987. doi:10.1007/BF02579200.
- 13 Siddharth Gupta, Guy Sa'ar, and Meirav Zehavi. Collective graph exploration parameterized by vertex cover, 2023. arXiv:2310.05480.
- 14 Yuya Higashikawa, Naoki Katoh, Stefan Langerman, and Shin-ichi Tanigawa. Online graph exploration algorithms for cycles and trees by multiple searchers. *J. Comb. Optim.*, 28(2):480–495, 2014. doi:10.1007/s10878-012-9571-y.
- 15 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *J. Comput. Syst. Sci.*, 79(1):39–49, 2013. doi:10.1016/j.jcss.2012.04.004.
- 16 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.
- 17 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/moor.12.3.415.
- 18 Hiroshi Nagamochi and Kohei Okada. A faster 2-approximation algorithm for the minmax p-traveling salesmen problem on a tree. *Discret. Appl. Math.*, 140(1-3):103–114, 2004. doi:10.1016/j.dam.2003.06.001.
- 19 Christian Ortolf and Christian Schindelhauer. A recursive approach to multi-robot exploration of trees. In Magnús M. Halldórsson, editor, *Structural Information and Communication Complexity - 21st International Colloquium, SIROCCO 2014, Takayama, Japan, July 23-25, 2014. Proceedings*, volume 8576 of *Lecture Notes in Computer Science*, pages 343–354. Springer, 2014. doi:10.1007/978-3-319-09620-9_26.

A W[1]-Hardness for CGE

In this section, we aim to prove the following theorem:

► **Theorem 1.3.** *CGE is W[1]-hard with respect to k even on trees whose treedepth is bounded by 3.*

We prove Theorem 1.3 by showing a reduction from EXACT BIN PACKING (see Definition 2.6).

First, we show that unary EXACT BIN PACKING is $W[1]$ -hard with respect to k . It is known that unary BIN PACKING is $W[1]$ -hard with respect to k [15]. So, we give a reduction from BIN PACKING to EXACT BIN PACKING in order to prove the following lemma:

► **Lemma A.1.** *Unary EXACT BIN PACKING is $W[1]$ -hard with respect to k .*

Proof. Let (I, s, B, k) be an instance of BIN PACKING problem. Let $t = B \cdot k - \sum_{i \in I} s(i)$ and let $s' : I \cup \{i_1, \dots, i_t\} \rightarrow \mathbb{N}$ be a function defined as follows. For every $i \in I$, $s'(i) = s(i)$, and for every $i_\ell \in \{i_1, \dots, i_t\}$, $s'(i_\ell) = 1$. Observe that $(I \cup \{i_1, \dots, i_t\}, s', B, k)$ is an instance of EXACT BIN PACKING. We show that (I, s, B, k) is a yes-instance of BIN PACKING if and only if $(I \cup \{i_1, \dots, i_t\}, s', B, k)$ is a yes-instance of EXACT BIN PACKING.

Assume that (I, s, B, k) is a yes-instance of BIN PACKING. Let I_1, \dots, I_k be a partition of I into disjoint sets such that for every $1 \leq j \leq k$, $\sum_{i \in I_j} s(i) \leq B$. For every $1 \leq j \leq k$, let $t_j = B - \sum_{i \in I_j} s(i)$. Let I'_1, \dots, I'_k be a partition of $\{i_1, \dots, i_t\}$ into k disjoint sets such that for every $1 \leq j \leq k$, $|I'_j| = t_j$. Observe that there exists such a partition since $\sum_{1 \leq j \leq k} t_j = t$. Clearly, $I_1 \cup I'_1, \dots, I_k \cup I'_k$ is a partition of $I \cup \{i_1, \dots, i_t\}$ into disjoint sets such that for every $1 \leq j \leq k$, $\sum_{i \in I_j \cup I'_j} s(i) = B$. Therefore, $(I \cup \{i_1, \dots, i_t\}, s', B, k)$ is a yes-instance of EXACT BIN PACKING.

Now, assume that $(I \cup \{i_1, \dots, i_t\}, s', B, k)$ is a yes-instance of EXACT BIN PACKING. Let I_1, \dots, I_k be a partition of $I \cup \{i_1, \dots, i_t\}$ into disjoint sets such that for every $1 \leq j \leq k$, $\sum_{i \in I_j} s(i) = B$. Observe that $I_1 \setminus \{i_1, \dots, i_t\}, \dots, I_k \setminus \{i_1, \dots, i_t\}$ is a partition of I into disjoint sets such that for every $1 \leq j \leq k$, $\sum_{i \in I_j} s(i) \leq B$. Therefore, (I, s, B, k) is a yes-instance of BIN PACKING.

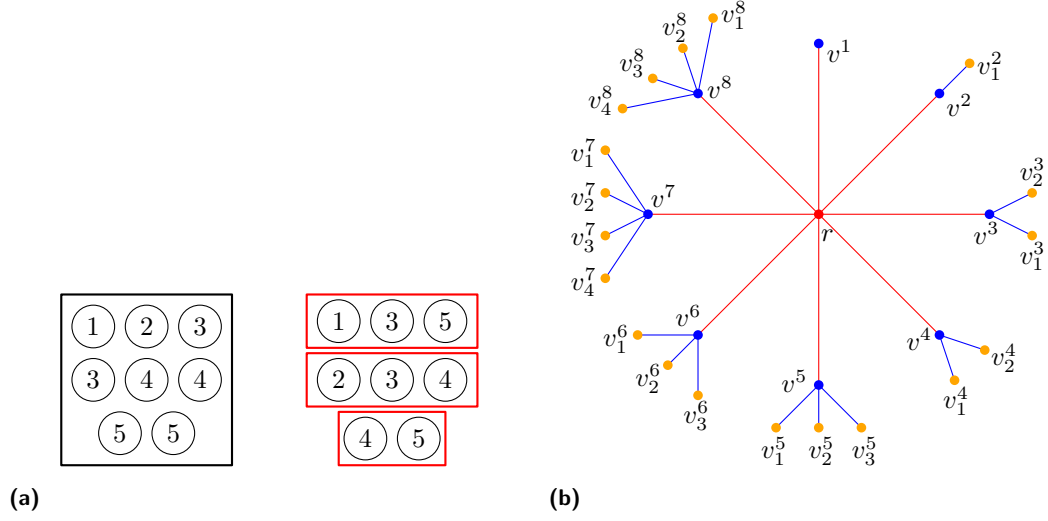
Clearly, the reduction works in polynomial time when the input is in unary. Thus, since unary BIN PACKING is $W[1]$ -hard with respect to k [15], unary EXACT BIN PACKING is $W[1]$ -hard with respect to k . ◀

A.1 Reduction From EXACT BIN PACKING to CGE

Given an instance (I, s, B, k) of EXACT BIN PACKING problem, denote by $\text{BinToRob}(I, s, B, k)$ the instance of CGE defined as follows. First, we construct the graph T as follows. For each $i \in I$ we create a star with $s(i) - 1$ leaves. We connect each such star with an edge to a vertex r . Formally, $V(T) = \{v^i, v_1^i, \dots, v_{s(i)-1}^i \mid i \in I\} \cup \{r\}$ and $E(T) = \{\{v^i, v_j^i\} \mid i \in I, 1 \leq j \leq s(i) - 1\} \cup \{\{r, v_i\} \mid i \in I\}$. Now, we define $\text{BinToRob}(I, s, B, k) = (T, r, k, 2B)$. See Figure 5 for an example. Next, we prove the correctness of the reduction:

► **Lemma A.2.** *Let (I, s, B, k) be an instance of EXACT BIN PACKING. Then, (I, s, B, k) is a yes-instance if and only if $\text{BinToRob}(I, s, B, k)$ is a yes-instance of CGE.*

Proof. First, assume that (I, s, B, k) is a yes-instance. Let I_1, \dots, I_k be a partition of I into disjoint sets such that for every $1 \leq j \leq k$, $\sum_{i \in I_j} s(i) = B$. We prove that $\text{BinToRob}(I, s, B, k) = (T, r, k, 2B)$ is a yes-instance of CGE, by showing that there exist k multisets $\widehat{E}_1, \dots, \widehat{E}_k$ such that the conditions of Lemma 3.4 are satisfied. For every $1 \leq j \leq k$, let $\widehat{E}_j = \{\{v^i, v_t^i\}, \{v^i, v_t^i\} \mid i \in I_j, 1 \leq t \leq s(i) - 1\} \cup \{\{v^i, r\}, \{v^i, r\}\}$. Clearly, $r \in V(\text{Graph}(\widehat{E}_j))$, $\text{Graph}(\widehat{E}_j)$ is connected, and every vertex in $\text{Graph}(\widehat{E}_j)$ has even degree. Therefore, Conditions 1 and 2 are satisfied. In addition, since $I = I_1 \cup \dots \cup I_k$, we have that $E \subseteq \widehat{E}_1 \cup \dots \cup \widehat{E}_k$, so Condition 3 is satisfied. Now, for every $1 \leq j \leq k$,



■ **Figure 5** An illustration of a EXACT BIN PACKING instance, a solution (in sub-figure (a)) and the equivalent instance of CGE constructed by the BinToRob function (in sub-figure (b)).

$|\widehat{E}_j| = |\{\{v^i, v_t^i\}, \{v^i, v_t^i\} \mid i \in I_j, 1 \leq t \leq s(i) - 1\} \cup \{\{v^i, r\}, \{v^i, r\}\}| = \sum_{i \in I_j} 2(s(i) - 1) + \sum_{i \in I_k} 2 = 2 \sum_{i \in I_k} s(i) = 2B$. Thus, Condition 4 is satisfied. Therefore, all the conditions of Lemma 3.4 are satisfied, so BinToRob(I, s, B, k) is a yes-instance of CGE.

Now, we prove the reverse direction. Assume that BinToRob(I, s, B, k) = ($T, r, k, 2B$) is a yes-instance of CGE. From Lemma 3.4, there exist k multisets $\widehat{E}_1, \dots, \widehat{E}_k$ such that the conditions of Lemma 3.4 hold. Let $1 \leq j \leq k$. We first show that every $\{u, v\} \in \widehat{E}_j$ appears at least twice in \widehat{E}_j . Let $\{u, v\} \in \widehat{E}_j$. We the following two cases:

Case 1: $\{u, v\} = \{v^i, v_t^i\}$ for some $i \in I$ and $1 \leq t \leq s(i) - 1$. From Condition 2 of Lemma 3.4, v_t^i has even degree in $\text{Graph}(\widehat{E}_j)$. Since $\{v^i, v_t^i\}$ is the only edge having v_t^i as an endpoint in T , $\{v^i, v_t^i\}$ appears an even number of times in \widehat{E}_j , and so it appears at least twice in \widehat{E}_j .

Case 2: $\{u, v\} = \{v^i, r\}$ for some $i \in I$. From Condition 2 of Lemma 3.4, v^i has even degree in $\text{Graph}(\widehat{E}_j)$. From Case 1, each $\{v^i, v_t^i\} \in \widehat{E}_j$ appears an even number of times in \widehat{E}_j . Therefore, since r is the only neighbor of v^i other than v_t^i , $1 \leq t \leq s(i) - 1$, $\{v^i, r\}$ appears an even number of times, which is greater or equal to 2, in \widehat{E}_j .

Now, observe that $|E(T)| = \sum_{i \in I} s(i) = B \cdot k$, and from Condition 4 of Lemma 3.4, $\sum_{1 \leq j \leq k} |\widehat{E}_j| \leq 2B \cdot k$. In addition, from Condition 3 of Lemma 3.4, (1): $E(T) \subseteq \widehat{E}_1 \cup \dots \cup \widehat{E}_k$. So, since we have already proved that for every $1 \leq j \leq k$, each $\{u, v\} \in \widehat{E}_j$ appears at least twice in \widehat{E}_j , we get that for every $1 \leq j < j' \leq k$, $\widehat{E}_j \cap \widehat{E}_{j'} = \emptyset$, and $\sum_{1 \leq \ell \leq k} |\widehat{E}_\ell| = 2B \cdot k$; in turn, for every $1 \leq j \leq k$, $|\widehat{E}_j| = 2B$, and each $\{u, v\} \in \widehat{E}_j$ appears exactly twice in \widehat{E}_j . Moreover, from Conditions 1 and 2 of Lemma 3.4, for every $1 \leq j \leq k$, $r \in V(\text{Graph}(\widehat{E}_j))$ and $\text{Graph}(\widehat{E}_j)$ is connected. Therefore, for every $1 \leq j \leq k$ and $i \in I$, if $v^i \in V(\text{Graph}(\widehat{E}_j))$ then $\{\{v^i, v_t^i\} \mid 1 \leq t \leq s(i) - 1\} \cup \{r, v^i\} \subseteq \widehat{E}_j$. Thus, for every $1 \leq j < j' \leq k$, (2): $V(\text{Graph}(\widehat{E}_j)) \cap V(\text{Graph}(\widehat{E}_{j'})) = \{r\}$.

22:18 Collective Graph Exploration Parameterized by Vertex Cover

Now, for every $1 \leq j \leq k$, let $I_j = \{i \in I \mid v^i \in V(\text{Graph}(\widehat{E}_j))\}$. By (1) and (2), I_1, \dots, I_k is a partition of I into disjoint sets. We show, that for every $1 \leq j \leq k$, $\sum_{i \in I_j} s(i) = B$. Let $1 \leq j \leq k$. Then, $\sum_{i \in I_j} s(i) = \sum_{i \in I_j} |\{\{v_i, v_{i_t}\} \mid 1 \leq t \leq s(i) - 1\} \cup \{r, v_i\}| = \frac{1}{2} |\widehat{E}'_j| = \frac{1}{2} \cdot 2 \cdot B = B$. Therefore I_1, \dots, I_k is a solution for (I, s, B, k) , so (I, s, B, k) is a yes-instance of EXACT BIN PACKING problem. This ends the proof. \blacktriangleleft

Clearly, the reduction works in polynomial time when the input is in unary. In addition, observe that the treedepth of the tree, obtained by the reduction, is bounded by 3. Now, recall that, by Lemma A.1, unary EXACT BIN PACKING is $\mathsf{W}[1]$ -hard with respect to k . Thus, we conclude from Lemma A.2 the correctness of Theorem 1.3.

Drawn Tree Decomposition: New Approach for Graph Drawing Problems

Siddharth Gupta ✉ 

BITS Pilani, Goa Campus, India

Guy Sa'ar ✉

Ben Gurion University of the Negev, Beersheba, Israel

Meirav Zehavi ✉ 

Ben Gurion University of the Negev, Beersheba, Israel

Abstract

Over the past decade, we witness an increasing amount of interest in the design of exact exponential-time and parameterized algorithms for problems in Graph Drawing. Unfortunately, we still lack knowledge of general methods to develop such algorithms. An even more serious issue is that, here, “standard” parameters very often yield intractability. In particular, for the most common structural parameter, namely, treewidth, we frequently observe NP-hardness already when the input graphs are restricted to have constant (often, being just 1 or 2) treewidth.

Our work deals with both drawbacks simultaneously. We introduce a novel form of tree decomposition that, roughly speaking, does not decompose (only) a graph, but an entire drawing. As such, its bags and separators are of geometric (rather than only combinatorial) nature. While the corresponding parameter – like treewidth – can be arbitrarily smaller than the height (and width) of the drawing, we show that – unlike treewidth – it gives rise to efficient algorithms. Specifically, we get slice-wise polynomial (XP) time algorithms parameterized by our parameter. We present a general scheme for the design of such algorithms, and apply it to several central problems in Graph Drawing, including the recognition of grid graphs, minimization of crossings and bends, and compaction. Other than for the class of problems we discussed in the paper, we believe that our decomposition and scheme are of independent interest and can be further extended or generalized to suit even a wider class of problems. Additionally, we discuss classes of drawings where our parameter is bounded by $\mathcal{O}(\sqrt{n})$ (where n is the number of vertices of the graph), yielding subexponential-time algorithms. Lastly, we prove which relations exist between drawn treewidth and other width measures, including treewidth, pathwidth, (dual) carving-width and embedded-width.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Human-centered computing → Graph drawings; Theory of computation → Computational geometry

Keywords and phrases Graph Drawing, Parameterized Complexity, Tree decomposition

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.23

Related Version *Full Version:* <https://arxiv.org/abs/2310.05471> [33]

Funding *Siddharth Gupta:* Supported by Engineering and Physical Sciences Research Council (EPSRC) grant EP/V007793/1.

Guy Sa'ar: Supported in part by the Israeli Smart Transportation Research Center and by the Lynne and William Frankel Center for Computing Science at Ben-Gurion University.

Meirav Zehavi: Supported by the European Research Council (ERC) grant titled PARAPATH.



© Siddharth Gupta, Guy Sa'ar, and Meirav Zehavi;
licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 23; pp. 23:1–23:22

Leibniz International Proceedings in Informatics



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

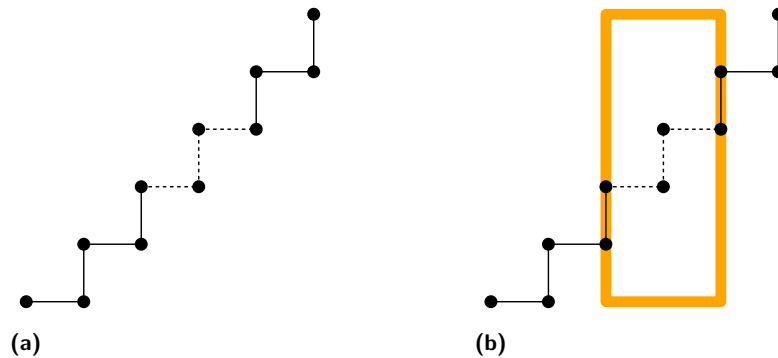
1 Introduction

Over the past decade, we witness an increasing amount of interest in the design of exact exponential-time and parameterized algorithms for problems in Graph Drawing. For a few illustrative examples, let us mention that this includes studies of crossing minimization [32, 37, 38], recognition of planar graph families such as upward planarity testing [16, 35] and grid graph recognition [34], as well as recognition of beyond planar graph families [4], turn-minimization [27], linear layouts such as books embeddings [5, 8], clustered planarity and hybrid planarity [19, 40, 20], and bend minimization [24, 22]. For more information on recent progress on these and other topics, we refer to the report [28] and surveys such as [48]. Unfortunately, still, we have very limited knowledge of general methods to develop exact exponential-time and parameterized algorithms for problems in Graph Drawing.

An even more serious issue is that, for Graph Drawing problems, “standard” parameters very often yield intractability. In particular, for the most common structural parameter, namely, treewidth,¹ we frequently observe NP-hardness already when the input graphs are restricted to have constant (often, being just 1 or 2) treewidth. The same result holds even for the larger parameter pathwidth. For example, GRID RECOGNITION is NP-hard on graphs of treewidth 1 (being trees) [7] or pathwidth 2 [34], ORTHOGONAL COMPACTION is NP-hard even on cycles [26] and hence on graphs of pathwidth (and treewidth) 2, MIN-AREA PLANAR STRAIGHT-LINE DRAWING is NP-complete on outerplanar graphs and hence on graphs of treewidth 2 [9, 39], and GRID UPWARD DRAWING is NP-complete on graphs of treewidth 1 (being trees) [1, 10]. In light of this, we must seek parameterizations that are larger (or incomparable) to treewidth. Due to the nature of the problems at hand, it is natural to seek parameters of geometric flavors. Here, perhaps, the first choice that comes to mind is the *height* (or, rather, the minimum among the height and width) of the sought (or given) drawing. In particular, we can easily observe that this parameter for planar orthogonal grid drawings is bounded by $\Omega(\text{tw})$, where tw is the treewidth of the drawn graph, and that it gives rise to the use of dynamic programming. However, denoting the number of vertices by n , we can also easily observe that this parameter can be as large as $\Omega(n)$ for ridiculously simple planar orthogonal grid drawings (and graphs)! For example, consider the path drawn in Figure 1a – here, already, both height and width are equal to (roughly) $n/2$.

Our work deals with both drawbacks mentioned above simultaneously. We introduce a novel form of tree decomposition that, roughly speaking, does not decompose (only) a graph, but an entire drawing. As such, its bags and separators are of geometric (rather than only combinatorial) nature. We further discuss this concept (still informally but in more detail) in Section 1.1 ahead. While the corresponding parameter – like treewidth – can be arbitrarily smaller than the height (and width) of the drawing (e.g., for the aforementioned example in Figure 1a, our parameter is a fixed constant), we show that – unlike treewidth – it gives rise to efficient (that is, XP) algorithms. Specifically, we present a general scheme for the design of such algorithms (described in Section 1.3), and apply it to several central problems in Graph Drawing, including the recognition of grid graphs, minimization of crossings and bends, and compaction (see Section 1.4). We believe that our new concept of geometric tree decomposition is interesting on its own, and exploring the connections between it and notions concerning (classical) tree decompositions is a promising research direction. Furthermore, we believe that this concept and our scheme can be further extended or generalized to be applicable to problems other than those discussed in this paper. Due to lack of space, several concepts and proofs are deferred to the full version of this paper [33].

¹ Definitions of standard terms and notations used in the Introduction can be found in Section A.



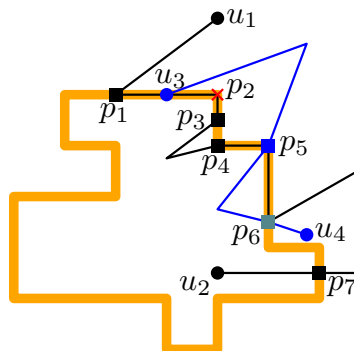
■ **Figure 1** (a) A drawing of a path on n vertices with height and width $(n - 1)/2$. However, the drawn treewidth is 16. (b) An illustration of a frame shown in orange with width 16.

1.1 The Concept of Drawn Tree Decomposition

Here, we discuss (informally) our main conceptual contribution: the introduction and study of the concepts of *drawn tree decomposition* and *drawn treewidth*, which we believe to be of independent interest. Then, in Section 1.2, we compare our parameter with several seemingly related graph parameters. Later, in Sections 1.3 and 1.4, we discuss our main technical contribution (which has been our initial motivation for these concepts): our general algorithmic scheme and its applications to problems in Graph Drawing. Our focus is on a class of rather general drawings of graphs on the Euclidean plane (allowing drawings of edges to have both crossings and bends, as well as to consist of segments that are not necessarily parallel to the axes), called *polyline grid drawings*. Roughly speaking, a polyline grid drawing d of a graph G is a mapping of the vertices of G to distinct grid points (being points of the form (i, j) where $i, j \in \mathbb{Z}$) and edges to *straight-line paths* between their endpoints. That is, the drawing of an edge is a simple curve that is the concatenation of straight-line segments (e.g, see Figure 11e in Section A.2). Towards the (informal) definition of a drawn tree decomposition ahead, we first introduce three critical terms: *frame*, *cutter*, and *rectangular*.

Frame. A *frame* is, simply, a straight-line cycle (defined analogously to a straight-line path above) whose segments are axis-parallel (see the orange polygon in Figure 2). In other words, it is a simple rectilinear polygon whose vertices lie on grid points.

For the definition of the width of our decomposition (presented later), we define the *width of a frame*. Roughly speaking, the width of a frame f , denoted by $\text{width}(f)$, is the sum of measures of the complexities of (i) the frame itself, and (ii) the “way” in which the drawing “traverses” the frame. For (i), we simply count the number of vertices of the frame (ignoring “superfluous” vertices, being those where the angle between incident edges is of 180 degrees). For (ii), we regard the drawings of vertices and edges separately (and sum up the two corresponding numbers). Specifically, for vertices, we simply count the number of vertices drawn on the frame. However, for edges, the measure is somewhat more complex, based on the notion of *turning points* (defined immediately); for each edge, we count the number of its turning points on the frame, and, then, the measure is the sum (over all edges) of these counters. We remark that some points on the plane might be counted multiple times – at the extreme case, the same point might be (a) a vertex of the frame, (b) a point on which a vertex of the graph is drawn, and (c) a turning point for one (or more) edges. We find this multi-count to be justified: the more complicated the frame and the drawing are at a certain point, the more that point “contributes” to the complexity of the measure.



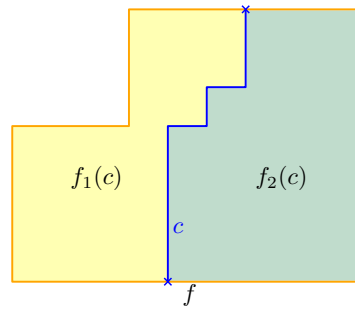
■ **Figure 2** The turning points of the black edge $e = \{u_1, u_2\}$ in the orange frame f are (p_1, e) , (p_3, e) , (p_4, e) , (p_6, e) and (p_7, e) . Note that, (p_2, e) and (p_5, e) are not turning points in f . Similarly, the turning points of the blue edge $e' = \{u_3, u_4\}$ in the frame f are $(d(u_3), e')$, (p_5, e') and (p_6, e') . Note that (p_5, e') is a turning point in f , but (p_5, e) is not a turning point in f .

Now, let us define the notion of a turning point. For this purpose, consider some edge $e = \{u, v\}$ of the graph and some point p on the frame. Then, roughly speaking, we refer to (p, e) as a turning point if, when we traverse the drawing of e from u to v or from v to u , we encounter p , and “immediately” before this encounter, we were in the strict interior or exterior of the frame. Additionally, we refer to (p, e) as a turning point if u or v themselves are drawn on p . An illustrative example is given in Figure 2.

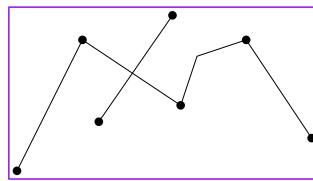
For an example of the definition of the width of a frame, we refer to Figure 1b. Here, the frame itself (being a rectangle) consists of exactly 4 vertices. Second, the path contains exactly 4 vertices that are drawn on the frame. Third, every point on which one of these vertices, say, v , is drawn is a turning point of 2 edges, being the two edges incident to v . So, the width of the frame is $4 + 4 + 4 \cdot 2 = 16$.

Cutter. A *cutter of a frame* is, simply, a straight-line path whose segments are axis-parallel and which intersects the frame in exactly two points, which are the endpoints of the cutter. Later, we discuss the “futility” of two simpler definitions for a cutter. The utility of a cutter of a frame f is, as its name suggests, in “cutting” f into (exactly) two frames f_1 and f_2 . Roughly speaking, we obtain one of f_1 and f_2 by the concatenation of the cutter with one path among the two subpaths of f between the endpoints of the cutter, and we obtain the other of f_1 and f_2 by the concatenation of the cutter with the other path among the two subpaths of f between the endpoints of the cutter. For more intuition, we refer the reader to Figure 3.

Rectangular. For the sake of intuition, the construction of a drawn tree decomposition may be thought of as a recursive process where, for a given frame, we compute a cutter that cuts it into two, and then proceed (recursively) with each of these two resulting frames. Then, two questions arise: What is the initial frame, and when does this process terminate? For the first question, the answer is simply the *rectangular* of the drawing (defined immediately). For the second question, the answer is even simpler – we stop when the current frame does not contain any grid point in its strict interior. Roughly speaking, the rectangular of a drawing is the (unique) frame whose interior is minimized among all frames whose “shape” is a rectangle and which contain the given drawing in their strict interior (see Figure 4).



■ **Figure 3** An illustration for a cutter c , shown in blue, of a frame f , shown in orange, and its associated frames $f_1(c)$ and $f_2(c)$.

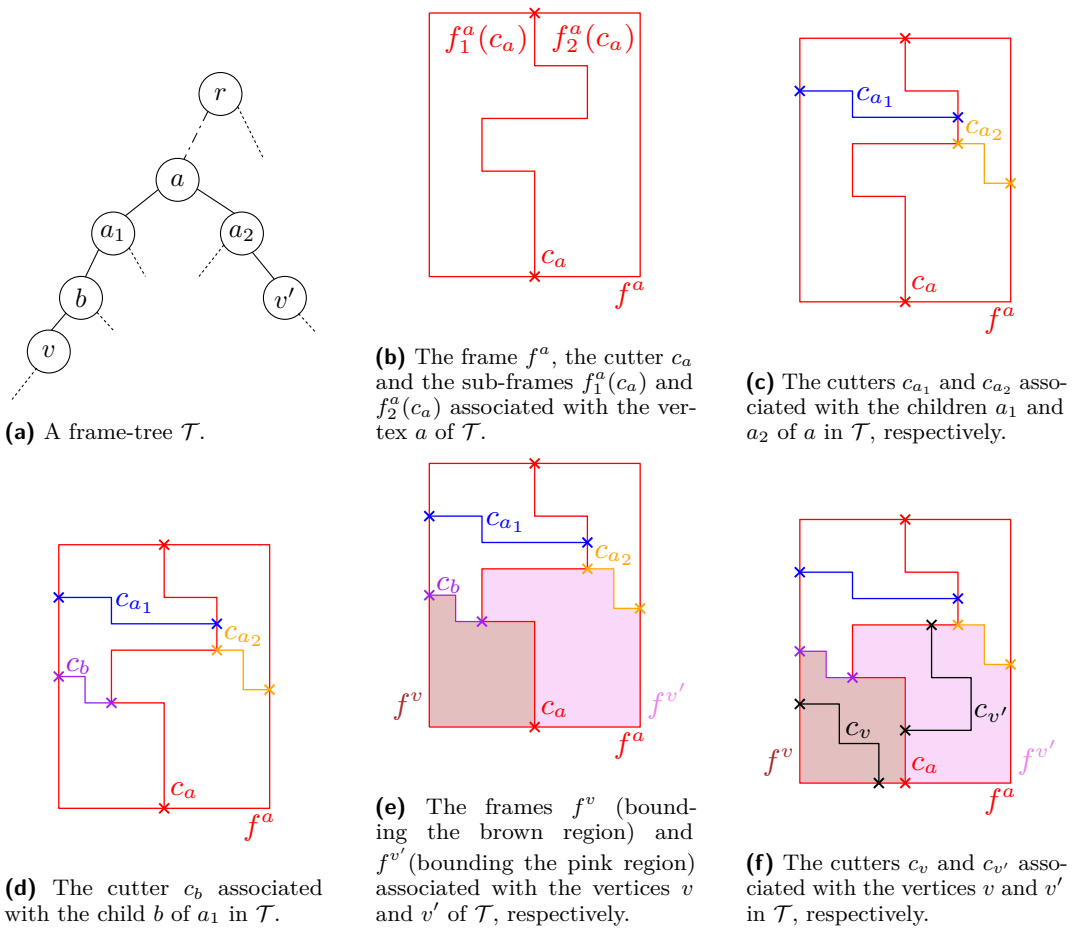


■ **Figure 4** The frame shown in purple is R_d where d is the drawing inside the frame.

Drawn Tree Decomposition. At the heart of the concept of a drawn tree decomposition, lies our definition of a *frame-tree* (abbreviation for *tree of frames*). Informally, for a graph G and a polyline grid drawing d of G , a frame-tree is a pair (\mathcal{T}, α) where \mathcal{T} is a binary rooted tree and α maps each vertex of \mathcal{T} to a frame, such that: (i) the root is mapped to the rectangular of d ; (ii) for every internal vertex v of \mathcal{T} , there exists a (unique) cutter c_v of $\alpha(v)$ so that the frames mapped to the children of v are those obtained by cutting $\alpha(v)$ by c_v ; (iii) the leaves of \mathcal{T} (and none of the internal vertices of \mathcal{T}) are mapped to frames whose strict interior does not contain any grid point. For an illustrative example, see Figure 5.

Now, for the definition of a drawn tree decomposition, we consider a frame-tree (\mathcal{T}, α) . Then, we “enrich” the frame-tree by the introduction of an additional mapping, β , from the vertex set of \mathcal{T} to subsets of vertices of G . In particular, we define β so that we can: (P1) prove that (\mathcal{T}, β) is a tree decomposition (this proof is slightly technical, based on case analysis); (P2) prove that, for every vertex v of \mathcal{T} , $|\beta(v)|$ is at most twice the sum of the widths of the frames of v and its two children (if they exist). For the definition of β , we (next) define the *set of vertices associated with a frame*, and the *set of vertices associated with a cutter* of a frame. Then, for a vertex v of \mathcal{T} , $\beta(v)$ is simply the union of the set of vertices associated with $\alpha(v)$, and the set of vertices associated with the cutter c_v of $\alpha(v)$. Correspondingly, the triple $(\mathcal{T}, \alpha, \beta)$ is a drawn tree decomposition.

So, consider a graph G , a polyline grid drawing d of G , a frame f and a cutter c of f . Then, the *set of vertices associated with f* is the union of the set of vertices of G that d draws on f and the set of endpoints of edges of G whose drawing (by d) is *separated* by f – that is, edges having one endpoint in the strict interior of f and the other endpoint in the strict exterior of f (see Figure 6a). Similarly, the *set of vertices associated with c* is the union of the set of vertices of G that d draws on c and the set of endpoints of edges of G whose drawing (by d) is *separated* by c – that is, edges having one endpoint in the strict interior of one of the frames obtained by cutting f by c , and the other endpoint in the strict interior of the other frame obtained by cutting f by c (see Figure 6b).

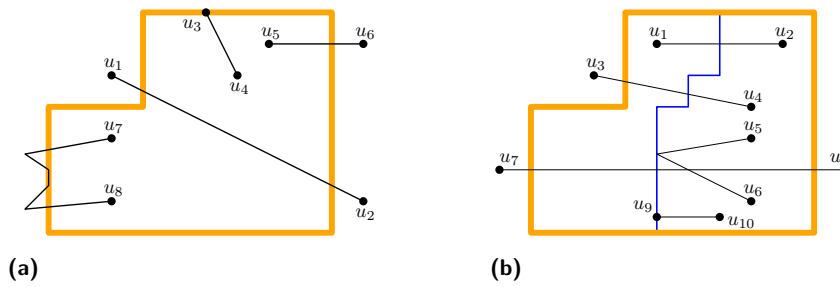


■ **Figure 5** Example of frames and cutters of a frame-tree. For clarity, the polyline grid drawing is not shown.

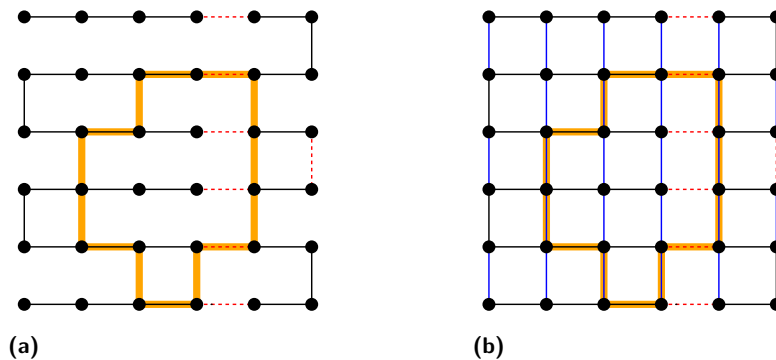
Drawn Treewidth. The *width* of a drawn tree decomposition $(\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}}), \alpha, \beta)$ is the maximum width of its frames, that is, $\max_{v \in V_{\mathcal{T}}} \text{width}(\alpha(v))$. Accordingly, the drawn treewidth of a polyline grid drawing d of a graph G is the minimum width of a drawn tree decomposition of d . Notably, due to (P1) and (P2) mentioned above, we can easily conclude that the treewidth of G is at most 6 times its drawn treewidth.

We remark that the usage of frames bears similarity to that of *cycle separators of planar graphs* (being a central player in proofs of the planar separator theorem; see, e.g., [3, 42]). However, the corresponding widths (drawn treewidth versus treewidth) can be critically different: While treewidth is bounded from above by the order of drawn treewidth, we have already pointed out that for various problems where treewidth yields intractability, drawn treewidth does not – this, of course, implies that treewidth can, often, be arbitrarily smaller than drawn treewidth; for a concrete example, see Figure 7. Further, treewidth depends only on the graph, while drawn treewidth depends (as desired) on the drawing; for example, notice that Figures 1a and 7a depict the same graph, but the corresponding drawings have radically different drawn treewidths.

Besides its above-mentioned relation to treewidth, drawn treewidth for planar orthogonal grid drawings can also be related to height (and width). On the one hand, we prove the desirable property that – like treewidth – drawn treewidth is bounded from above by the order



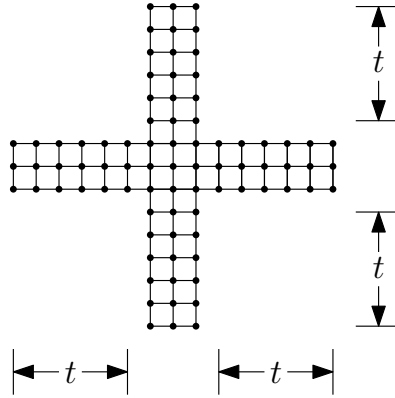
■ **Figure 6** Example of vertices associated with a frame and a cutter. (a) The edge $\{u_5, u_6\}$ is the only edge separated by the orange frame. The vertices associated with the orange frame are u_3, u_5 and u_6 . (b) The vertices associated with the blue cutter of the orange frame are u_1, u_2 and u_9 .



■ **Figure 7** (a) A path P on n vertices and a frame f shown in orange. (b) A grid graph G on the same set of vertices and the frame f shown in orange. Consider a frame, say f , in P with width w . Observe that f is also a frame in G . Moreover, the width of f in G is at most $3w$ as every vertex has exactly 2 more edges in G compared to P so the vertex may be counted 2 more times in the width of f in G as the turning points of those 2 extra edges. As treewidth is a lower bound for drawn treewidth and the treewidth of a grid graph is \sqrt{n} , the drawn treewidth of P is $\Omega(\sqrt{n})$ (while its treewidth is 1).

of the minimum among the height and width of the drawing. Notably, various central graph width measures do *not* have this property. For example, one of the most commonly used relaxations of pathwidth is *treedepth* (see, e.g., [18] for information on treedepth); however, the treedepth of an n -vertex path is $\lceil \log_2(n + 1) \rceil$, while it can be easily drawn so that the height (or, symmetrically, width) of the drawing is 1. On the other hand, we have already observed that the drawn treewidth can be arbitrarily smaller than the minimum among the height and width of a drawing (see Figure 1a).

Bounds for Specific Types of Drawings. For some classes of drawings (being subclasses of polyline grid drawings), we are able to prove that drawn treewidth is bounded by a sublinear function of n (the number of vertices of the graph). For example, for grid drawings – which are mappings of vertices to distinct grid points and of edges to unit-length straight lines between their endpoints (see Figure 11b in Section A.2) – we prove that the drawn treewidth (and even the *straight-line drawn treewidth*, defined ahead) is bounded by $\mathcal{O}(\sqrt{n})$. More generally, we prove that given a graph G and an orthogonal grid drawing d of G , drawn treewidth of d is $\mathcal{O}(\Delta \cdot \sqrt{\Delta} \cdot \ell \cdot n \cdot \max \text{Int})$, where (i) Δ is the maximum degree in G , (ii) ℓ is the average length of the edges of G in d , and (iii) $\max \text{Int}$ is the maximum number of edges and vertices intersected in a grid point in d .

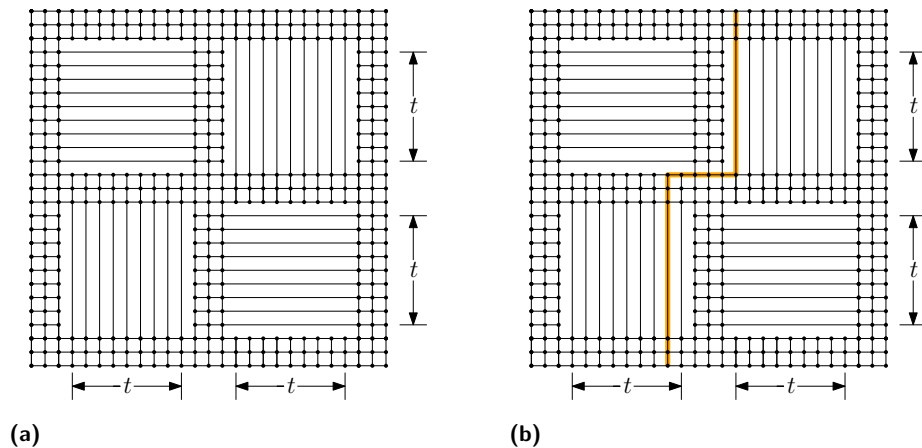


■ **Figure 8** Example of a rectilinear drawing of a graph on n vertices with $t = \Omega(n)$. The horizontal/vertical drawn treewidth of this drawing is $\Omega(n)$.

At this point, a short discussion is in order. One of the most well-known results in graph theory about planar graphs is that every n -vertex planar graph has pathwidth (and hence treewidth) bounded by $\mathcal{O}(\sqrt{n})$ [13]. In particular, this result and generalizations thereof have found impactful applications in algorithm design, particularly of parameterized and approximation algorithms. In fact, (almost) all subexponential-time algorithms for problems on planar graphs rely on it. Here, a central component in several proofs is the planar separator theorem [3, 41, 42] (briefly mentioned earlier), which states that every n -vertex planar graph contains an $\mathcal{O}(\sqrt{n})$ -sized subset of vertices (called separator) whose removal from the graph yields connected components that are each of size at most $2n/3$. Thus, due to the above-mentioned sub-quadratic bound on drawn treewidth for grid drawings, the following *conjecture* seems tempting: the drawn treewidth of any *planar* polyline grid drawing is $\mathcal{O}(\sqrt{n})$. However, we observe that the statement analogous to the planar separator theorem does not hold in our case, where our notion of a separator is that of a cutter and their sizes is, in particular, bounded from below by the size of the set of vertices associated with the cutter.

Drawbacks of Simpler Definitions for a Cutter. Lastly, we present and discuss two alternative restricted forms of cutters: *horizontal (or vertical) cutters* and *straight-line cutters*. A horizontal cutter (resp., vertical cutter) of a frame is a cutter of that frame where all vertices have the same y -coordinate (resp., x -coordinate). Then, a straight-line cutter is a cutter that is either horizontal or vertical. The replacement of cutters by horizontal/vertical cutters or straight-line cutters yields corresponding definitions of horizontal/vertical drawn tree decompositions and straight-line drawn tree decompositions, and, accordingly, of horizontal/vertical drawn treewidth and straight-line drawn treewidth. In particular, when we use these restricted forms of cutters, every frame has the shape of a rectangle. In turn, this significantly simplifies the visualization (and, possibly, also the use) of these concepts.

Unfortunately, horizontal/vertical drawn treewidth and even straight-line drawn treewidth can be arbitrarily larger than drawn treewidth. To see this, let us first consider horizontal cutters (or, symmetrically, vertical cutters), and the graph depicted in Figure 8. Notably, this graph, in fact, admits *exactly one* grid drawing (up to isomorphism) – the one depicted in the figure. Now, notice that the horizontal drawn treewidth of this drawing is $\Omega(n)$. To see



■ **Figure 9** (a) Example of a rectilinear drawing of a graph on n vertices with $t = \Omega(n)$. The straight-line drawn treewidth of this drawing is $\Omega(n)$. (b) Example of a cutter used in the drawn tree decomposition of width $O(1)$.

this, notice that, for any horizontal tree decomposition and for each of the three horizontal straight lines in the “middle” of the drawing, the rooted tree will have to contain a vertex whose associate cutter “coincides” with that line. However, the drawn treewidth of this drawing is only $O(1)$, and, more generally, recall that we prove that for any grid drawing, the straight-line drawn treewidth (and hence also the drawn treewidth) is $O(\sqrt{n})$. So, for example, by using only horizontal (or vertical) cutters, we will not be able to attain the subexponential-time algorithm for GRID RECOGNITION mentioned in Section 1.3.

Nevertheless, the straight-line treewidth of the drawing in Figure 8 can be seen to be bounded by $O(1)$ as well. However, regarding straight-line cutters, we consider the graph depicted in Figure 9a. Notably, every *rectilinear grid drawing* of this graph (being a generalization of a grid drawing, where edges are straight-lines of arbitrary lengths) can be obtained from the one depicted in the figure by “stretching” the drawings of some of its edges (and up to isomorphism). Now, notice that the straight-line drawn treewidth of this drawing is $\Omega(n)$. To see this, notice that every axis-parallel straight-line that intersects this graph, intersects the drawings of at least $\Omega(n)$ distinct vertices and edges of this graph. However, the drawn treewidth of this drawing is only $O(1)$. To see this, consider the usage of cutters as the one depicted in Figure 9b.

1.2 Comparison with Other Graph Width Parameters

Recall that, drawn tree decomposition is based on decomposing a given polyline grid drawing of a graph. Therefore, the drawn treewidth is dependent on the polyline grid drawing of the graph. For e.g., Figures 1a and 7a depicts two different drawings of the same path which have different drawn treewidth. As path has a unique embedding, this also shows that **different drawings of the same embedded graph may have different drawn treewidth. To the best of our knowledge, our parameter is the only one that depends on the drawing (rather than the embedding or just the graph)**. Thus, we compare and discuss the differences between the drawn treewidth of a given polyline drawing of the graph and some seemingly related graph width parameters, namely: treewidth, pathwidth, carving-width, dual carving-width and embedded-width. Note that, the dual carving-width and the embedded-width is only defined when the given graph is a plane graph. Specifically, we prove the following theorem.

- **Theorem 1.1.** *Given a graph G and a polyline drawing d of G , we have the following.*
- (a) *The treewidth of G is at most 6 times the drawn treewidth of d . Moreover, the drawn treewidth of d might be arbitrary larger than the treewidth of G .*
 - (b) *The pathwidth of G and the drawn treewidth of d are incomparable.*
 - (c) *The drawn treewidth of d might be arbitrary larger than the carving-width of G .*
 - (d) *If G is a plane graph, the dual carving-width and the embedded-width of G might be arbitrary larger than the drawn treewidth of d .*

We now give the proof of the above theorem. Let Δ , tw , pw , and cw be the maximum degree, treewidth, pathwidth and the carving-width of G , respectively. Further, if G is a plane graph, let ℓ , dcw and emw be the maximum face size, the dual carving-width (the carving width of the dual graph), and the embedded-width of G , respectively.

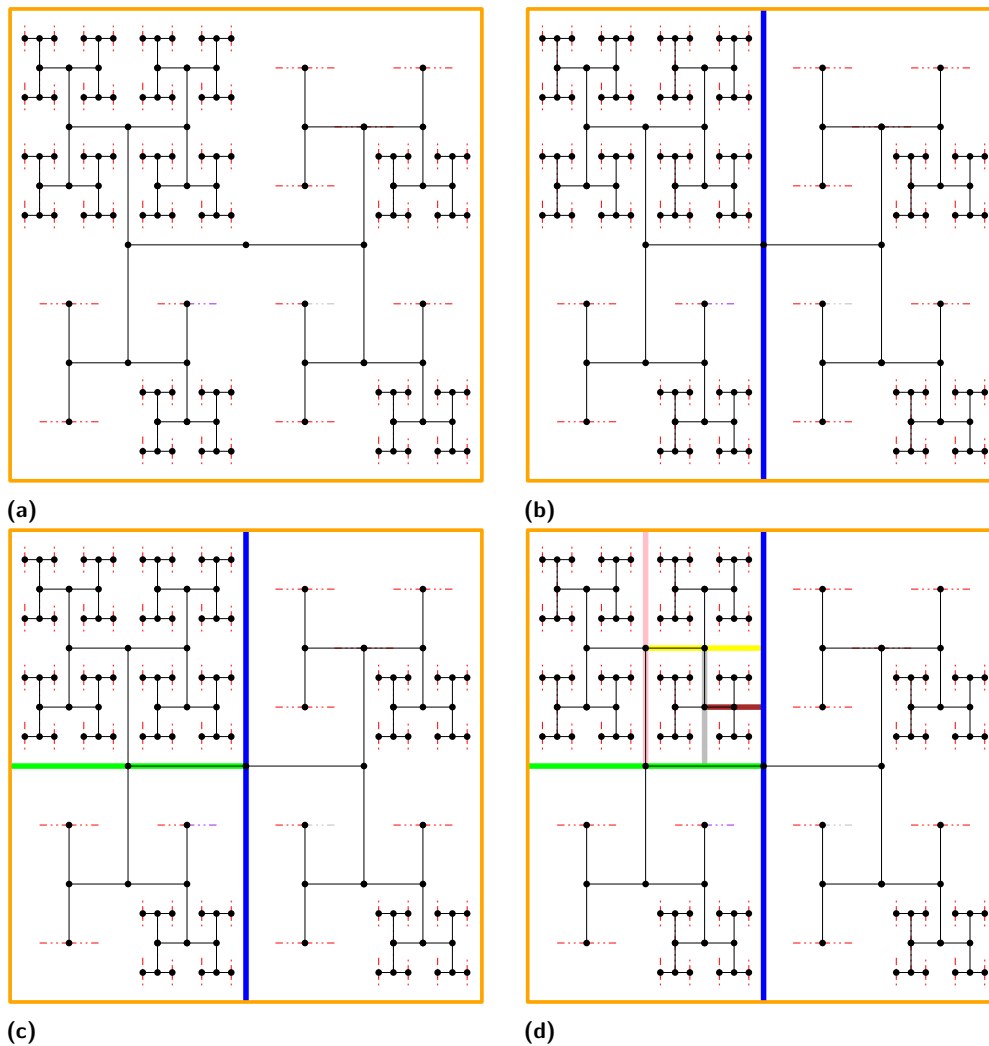
Comparison with Treewidth. As mentioned earlier in Section 1.1, we prove that given a graph and a polyline drawing of it, tw is at most 6 times the drawn treewidth. Moreover, we also show that given a graph and a polyline drawing of it, the drawn treewidth of the drawing might be arbitrary larger than the treewidth of the graph (see Figure 7).

Comparison with Pathwidth. In Figure 10, we have a rectilinear grid drawing of a binary tree. By using cutters as illustrated in the figure (in orange), we can get a drawn tree decomposition of constant width. In particular, one can see that each cutter intersects a constant number of edges and vertices. Since we use only straight cutters, and the maximum degree of the graph is 3, we conclude that the width of each frame in such a drawn tree frame is bounded by a constant. Therefore, we get that the drawn treewidth of the drawing is bounded by a constant. Observe that this example can be expanded to a binary tree of any size. Furthermore, the pathwidth of a binary tree with n vertices is $\Omega(\log_2(n))$. So, given a graph and a polyline drawing of it, the pathwidth of the graph might be arbitrary larger than the drawn treewidth of the drawing.

On the other hand, GRID RECOGNITION is NP-hard on graphs of pathwidth 2, and we show in this paper that the problem is XP with respect to drawn treewidth. So, given a graph and a polyline drawing of it, the drawn treewidth of the drawing might be arbitrary larger than the pathwidth of the graph. Thus, we conclude that the two parameters, pathwidth and drawn treewidth, are incomparable.

Comparison with Carving-width, Dual Carving-width and Embedded-width. It is known that $\text{cw} \leq \Delta(\text{tw} + 1)$ [11]. As the GRID RECOGNITION problem is NP-hard even for binary trees, we get that it is NP-hard even for graphs of carving-width at most 6. In this paper, we show that the problem is XP with respect to drawn treewidth. So, given a graph and a polyline drawing of it, the drawn treewidth of the drawing might be arbitrary larger than the carving-width of the graph.

If the given graph is plane, it is known that $\ell \leq \text{dcw}$ and $\ell \leq \text{emw}$ [20]. Therefore, we get that both the dual carving-width and the embedded-width of a path are at least the size of its vertex set. In this paper, we show that there exists a drawing of any path with drawn treewidth at most 16 (see Figure 1b). So, given a plane graph and a polyline drawing of it, the dual carving-width and the embedded-width of the graph might be arbitrary larger than the drawn treewidth of the drawing. Thus, we conclude that drawn treewidth differs from carving-width, dual carving-width and embedded-width.



■ **Figure 10** Example of a rectilinear drawing (in black) of a binary tree on n vertices. The rectangular is shown in orange. Examples of cutters are shown in blue, green, pink, yellow, grey and brown. Each one of them intersects $O(1)$ vertices and edges. Overall, the pathwidth of the tree is $\Omega(\log n)$, while the drawn treewidth of this drawing is $O(1)$.

1.3 Our Scheme

Here, we present (informally) our general scheme for the design of algorithms for problems in Graph Drawing parameterized by the drawn treewidth of the sought drawing (that should be, in particular, a polyline grid drawing), based on dynamic programming. For the clarity of the discussion, we first introduce the four main definitions required for the scheme and its proof of correctness. Then, we discuss the usage of our scheme – specifically, which two procedures the user should design in order to apply the scheme as a black box. Afterwards, we specify the properties that a problem should satisfy so that our scheme will solve it correctly, and the running time that will be attained. Lastly, we present some technical details concerning the scheme itself, that is, how it is executed.

Key Players: Info-Frames, Info-Cutters, Splitting and Glueing.

Info-Frames. The most basic definition required for our scheme is that of an *info-frame*. Briefly, an info-frame encodes information about the “behaviour” of the restriction of some drawing d to the interior of some particular frame f . For that purpose, the info-frame consists of five components, where the first one is, simply, the frame f . The second component is a drawing d_f that specifies the drawings of the vertices and edges (by d) of the graph on f itself. More precisely, d_f specifies which vertices of the graph are drawn on f and where are they drawn on f . Additionally, for every edge e of the graph, it specifies which are the turning points of e on f and where are these turning points drawn on f . Moreover, for the aforementioned turning points, it specifies the order in which they are encountered (when we “walk” along the drawing of e from one end to the other), and for each maximal subcurve of the drawing of e that does not contain a turning point internally, it specifies whether this subcurve is drawn on f (i.e., being a subcurve of f as well), and if yes, then it specifies the drawing of this subcurve (for which, knowing the drawings of its endpoints, we have only two options).

The third and fourth components, denoted by U_f and E_f , concern the strict interior of f . Specifically, U_f specifies which vertices of the graph are drawn strictly inside f . As for E_f , for every edge e of the graph and for each maximal subcurve of the drawing of e that does not contain a turning point internally, it specifies whether this subcurve is drawn in the strict interior of f (except for, possibly, the endpoints of the curve). We remark that the number of “sensible” choices for U_f and E_f is much smaller than it might appear to be at first glance, supposing that the graph at hand is connected. The fifth component, roughly speaking, describes the “angles” in which drawings of edges cross f using straight line segments attached to turning points. Such information is necessary, for example, to ensure that some subcurves corresponding to the drawings of the same edge lie in a single straight line, so that no bend – if forbidden by the problem at hand – occurs.

Importantly, the definition of an info-frame is independent of a specific drawing, being an “abstract” tuple of five components. Every drawing that can be described by the tuple (as discussed above) is said to be a drawing of the info-frame. So, one info-frame may describe multiple drawings, or none at all. We note that for an “abstract” five-component tuple to be an info-frame, it should satisfy various (considerably technical) properties, which, in particular, any info-frame that does describe at least one drawing must satisfy. On the one hand, these properties bound the number of possible info-frames, and, on the other hand, they are also used in the proof of correctness of our scheme.

Lastly, observe that the restriction of some drawing d to the interior of some particular frame f is *not* a drawing of a *graph*. Indeed, some edges are drawn (by d) partially in the interior of f and partially in the strict exterior of f . However, if we “enrich” the graph by placing “virtual” vertices on turning points, then the restriction of d to the interior of f will be a drawing of a graph (being a subgraph of the enriched graph). So, for technical reasons, this is exactly what we do. For this purpose, we define and work with so-called G^* -drawings; however, to keep the overview short and simple, we will not discuss G^* -drawings and related technical terms in this overview.

Info-Cutters. Just as we use an info-frame to encode information about the “behaviour” of a drawing d with respect to a frame f , we use an *info-cutter* of an info-frame to encode information about the “behaviour” of d with respect to a cutter c of f . Rather than directly describing how d is drawn on c and how d is “split” by c inside f , we find it easier to indirectly describe this information by defining an info-cutter based on two info-frames corresponding to the frames obtained by cutting f with c (later, for the dynamic programming implementation,

we can thus immediately know to which already computed entries to refer). Observe that, in particular, the two frames being part of these two info-frames contain c , and, thus, these two info-frames capture the aforementioned information.

To be more precise, an info-cutter C of an info-frame F , where the first component of F is some frame f , is a triple (c, F_1, F_2) , where, in particular, c is a cutter of f , and F_1 and F_2 are info-frames for the two frames obtained by cutting f with c . Additionally, for such a triple to be an info-cutter, it should satisfy (considerably technical) properties, which, in particular, any info-cutter that does describe at least one drawing must satisfy. Very briefly, these properties validate consistency between the information described by F , F_1 and F_2 . This is more complicated than it might appear to be at first glance, since, even on the cutter c , F_1 and F_2 might describe the existence of different virtual vertices (having different turning points). For the sake of simplicity, we do not discuss these details in the overview.

Splitting and Glueing. First, let us consider the *splitter function*, which, for our scheme, is used only for the proof of correctness (where its input is assumed to contain a subdrawing of a hypothetical solution drawing). Given an info-frame F whose first component (being a frame) is f , a drawing d restricted to the interior of f that is compatible with the description encoded by F , and a cutter c of f , the splitter function returns an info-cutter $C = (c, F_1, F_2)$ and two drawings, d_1 and d_2 . Let f_1 (f_2) be the first component of F_1 (F_2). Briefly, we define the output such that d_1 and d_2 would be the subdrawings of d restricted to the interiors of f_1 and f_2 , respectively, and F_1 and F_2 would be the info-frames that describe d_1 and d_2 , respectively.

The *glue function* is, intuitively, the “inverse” of the split function, and it is used algorithmically in our scheme. Its input consists of an info-frame F , an info-cutter $C = (c, F_1, F_2)$ of F , a drawing d_1 of F_1 and a drawing d_2 of F_2 . Roughly speaking, this function aims to “glue” d_1 and d_2 into a single drawing d that is restricted to the interior of f , being the first component of F , and that should be compatible with the description encoded by F ; of course, this operation might be impossible, and then the function simply announces that. Among other proofs concerning these functions, we show, in particular, that the specific way in which we define the splitter and glue functions (not described in the overview) ensures that, if we apply the glue function on an output of the splitter function, we are able to reconstruct the drawing given as input to the splitter function.

The User’s Point of View. For the execution of the scheme, we expect the user to provide four components: some universe denoted by INF, and three algorithmic procedures (that will be defined immediately). All of these components are problem-dependent.

- The first procedure, termed *classifier* and denoted by **Classifier**, is given an info-frame F and a corresponding drawing d , and it returns an element from INF. Intuitively, this element describes the equivalence class of d . So, we say that two drawings corresponding to the same info-frame are *equivalent* if the classifier associates them with the same element.
- The second procedure, termed *classifier algorithm*, is given an info-frame F , an info-cutter $C = (c, F_1, F_2)$ of F and $I_1, I_2 \in \text{INF}$, and it returns $I' \in \text{INF}$ such that: For any two drawings d_1 and d_2 corresponding to F_1 and F_2 , respectively, such that **Classifier**(F_1, d_1) = I_1 and **Classifier**(F_2, d_2) = I_2 , we have **Classifier**(F, d) = I' where $d = \text{Glue}(F, C, d_1, d_2)$. In particular, notice that any two drawings of the same two equivalence classes always yield (when being glued) a drawing of the same equivalence class – this justifies our usage of the term *equivalence* in this context.

- The third procedure, termed *leaf solver*, is given an info-frame F whose frame does not contain any grid point in its strict interior, and for every $I' \in \text{INF}$, it returns “yes” if and only if there exists a drawing d corresponding to F such that $\text{Classifier}(F, d) = I'$. Practically, we require this procedure to solve the basis of our dynamic programming computation, corresponding to info-frames whose frames do not contain any grid point in their strict interiors.

The scheme, once given these components, can be executed in a black box fashion. For the sake of simplicity of the overview, we do not discuss the technical details of the execution itself (as a white box) here.

To Which Type of Problems Does Our Scheme Apply? Roughly speaking, we prove that our scheme can be applied to any graph drawing problem Π such that:

1. Every instance of Π contains, in particular, a connected graph G , dimensions h and w for the sought drawing (which are, usually, bounded from above by the number of vertices n of G), and the parameter k (being any non-negative integer).
2. The objective is to determine whether G admits a polyline grid drawing bounded by rectangle of dimensions $h \times w$, whose drawn treewidth is at most k , and that satisfies various problem-specific properties (for some examples, see Section 1.4).
3. The user can design the three algorithmic procedures discussed above.

For any such problem Π , we prove that the runtime of the scheme is bounded by

$$\mathcal{O}(k \cdot h \cdot w \cdot n)^{\mathcal{O}(k)} \cdot |\text{INF}|^{\mathcal{O}(1)} \cdot \left(2^{\mathcal{O}(\Delta \cdot k)} \cdot \text{T2} + \text{T3}\right),$$

where T2 and T3 bound the runtimes of the second and third procedures provided by the user, and Δ is the maximum degree of G . In particular, if $h, w, |\text{INF}|, \text{T2}$ and T3 can be bounded by $n^{\mathcal{O}(1)}$ (which is the case for many applications, such as grid recognition and orthogonal compaction), then the runtime above simplifies to $n^{\mathcal{O}(k)}$, that is, we obtain an XP-algorithm.

1.4 Applications of Our Scheme to Problems in Graph Drawing

For most of the problems considered in this paper, the time complexity of our scheme can be bounded by $n^{\mathcal{O}(k)}$, where k is the input parameter that upper bounds the drawn treewidth of the output drawing. We remark that the formal definitions of these problems are relegated to Section A.3.

Grid Recognition. We first consider the relatively simple GRID RECOGNITION problem in order to demonstrate the application of our scheme. Here, given a (connected) graph G , the objective is to determine whether G is a grid graph, that is, whether it admits a grid drawing. The GRID RECOGNITION problem was first proved to be NP-hard in 1987, on ternary trees of pathwidth 3 [7]. Two years later in 1989, the problem was proved to be NP-hard even on binary trees [31]. Recently in 2021, the problem was proved to be NP-hard even on trees of pathwidth 2 [34]. In the same paper, it was also proved that the problem is polynomial time solvable on graphs of pathwidth 1. A year later in 2022, it was proved that even if we require all the internal faces of the drawing to be rectangles, the problem is still NP-hard even for biconnected graphs [2]. In the same paper, it was also proved that if we require all the faces of the drawing to be rectangles (including outer face), the problem is cubic time solvable.

As we deal with the parameterized version of this problem where the parameter is the drawn treewidth of the sought drawing (or, more precisely, an upper bound on it), we are also given k as input. We prove the following result.

► **Theorem 1.2.** *There exists an algorithm that solves the GRID RECOGNITION problem in time $n^{\mathcal{O}(k)}$.*

Since for grid drawings, we also prove that $k \leq \mathcal{O}(\sqrt{n})$, we get the following corollary.

► **Corollary 1.3.** *There exists an algorithm that solves the GRID RECOGNITION problem in time $n^{\mathcal{O}(\sqrt{n})}$.*

Thus, we obtain a subexponential-time algorithm for GRID RECOGNITION, matching the running time of the current best known algorithm for this problem [21].

Crossing and Bend Minimization. For our second application, we study a variant of the CROSSING MINIMIZATION problem. The CROSSING MINIMIZATION problem is one of the most fundamental graph layout problems. It was shown to be NP-complete by Garey and Johnson in 1983 [29]. Later, it was proved to be NP-complete even on graph of maximum degree 3 [36] and also on *almost planar graphs* which are graphs that can be made planar by removing a single edge [15]. It was also shown that the problem remains NP-hard even if the cyclic order of the neighbours around each vertex is fixed and to be respected by the resulting drawing [44]. On the positive side, it is known the problem is FPT with respect to the number of crossings [32, 38] and also with respect to the vertex cover [37]. There are many other variants of this problem which are studied in the literature. One of them concerns with minimizing the number of pairwise crossing edges in any straight-line drawing of the graph. This problem is known to be NP-hard [12] (and even $\exists\mathbb{R}$ -complete [46]). For more information about the crossing minimization and its variants, we refer to the survey [48].

A related problem is the BEND MINIMIZATION problem. Given a graph G , the BEND MINIMIZATION problem asks for an orthogonal grid drawing of G with minimum number of total bends. The problem was proved to be NP-complete in 2001, even when there are no bends [30]. On the positive side, if the input graph is plane, the problem can be solved in polynomial time [47]. When the input graph is not planar, there are polynomial time algorithms for subclasses of planar graphs, namely planar graphs with maximum degree 3 [17, 6, 25, 45] and series-parallel graphs [49].

We study the STRAIGHT-LINE GRID CROSSING MINIMIZATION problem where the sought drawing should be a straight-line grid drawing. Here, given a (connected) graph G and $h, w \in \mathbb{N}$, the objective is to determine a straight-line grid drawing of G bounded by a rectangle of dimension $h \times w$ with minimum number of crossings, if one exists. Similar to the previous example, as we study the parameterized version of this problem, we are also given k as input. We prove the following result.

► **Theorem 1.4.** *There exists an algorithm that solves STRAIGHT-LINE GRID CROSSING MINIMIZATION problem in time $\mathcal{O}((k \cdot h \cdot w \cdot n)^{\mathcal{O}(k)} \cdot 2^{\mathcal{O}(\Delta \cdot k)})$, where Δ is the maximum degree of the input graph.*

More generally, our scheme can be applied to a very wide class of problems of such flavor; in particular, every problem where:

- The input consists of (some or all of) the following: a graph G ; $\text{cross} : E(G) \rightarrow \mathbb{N}_0 \cup \infty$; $\text{bend} : E(G) \rightarrow \mathbb{N}_0 \cup \infty$, and $C, B, k \in \mathbb{N}_0 \cup \{\infty\}$. Here, $E(G)$ is the edge set of G , and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

- The objective is to determine whether G admits a drawing that is (i) a grid drawing, or (ii) a rectilinear drawing, or (iii) an orthogonal grid drawing, or (iv) a straight-line grid drawing, or (v) a polyline grid drawing, such that:
 - For every edge $e \in E(G)$, the drawing of e has at most $\text{cross}(e)$ crossings and at most $\text{bend}(e)$ bends.
 - In total, we have at most C crossings and at most B bends.

Further, the scheme can be applied to various variants of the above generic problem that were studied in the literature. For example, we can specify, for every edge, whether it should be crossed an even or odd number of times. Similarly, we can also consider the weighted crossing number.

Orthogonal Compaction. Lastly, we note that our scheme can also be applied to problems of flavors quite different than the above. As an example, we consider the ORTHOGONAL COMPACTION problem. Here, given a planar orthogonal representation H of a connected planar graph G , the objective is to compute a minimum-area drawing of H . The ORTHOGONAL COMPACTION problem was first proved to be NP-hard on general graphs in 2001 [43]. Later, it was shown that the problem is NP-hard even on cycles [26], ruling out an FPT algorithm with respect to treewidth, unless $P=NP$. On the positive side, it was proved that the problem is linear time solvable for a restricted class of planar orthogonal representation [14]. Recently, it was also shown that the problem is FPT with respect to number of “kitty corner vertices”, a parameter central to the problem [23].

Similar to the previous examples, as we study the parameterized version of this problem, we are also given k as input. We prove the following result.

► **Theorem 1.5.** *There exists an algorithm that solves the ORTHOGONAL COMPACTION problem in time $n^{\mathcal{O}(k)}$.*

References

- 1 Hugo A. Akitaya, Maarten Löffler, and Irene Parada. How to fit a tree in a box. *Graphs Comb.*, 38(5):155, 2022. doi:10.1007/s00373-022-02558-z.
- 2 Carlos Alegria, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Fabrizio Grosso, and Maurizio Patrignani. Unit-length rectangular drawings of graphs. In Patrizio Angelini and Reinhard von Hanxleden, editors, *Graph Drawing and Network Visualization - 30th International Symposium, GD 2022, Tokyo, Japan, September 13-16, 2022, Revised Selected Papers*, volume 13764 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 2022. doi:10.1007/978-3-031-22203-0_10.
- 3 Noga Alon, Paul D. Seymour, and Robin Thomas. Planar separators. *SIAM J. Discret. Math.*, 7(2):184–193, 1994. doi:10.1137/S0895480191198768.
- 4 Michael J. Bannister, Sergio Cabello, and David Eppstein. Parameterized complexity of 1-planarity. *Journal of Graph Algorithms and Applications*, 22(1):23–49, 2018.
- 5 Michael J. Bannister and David Eppstein. Crossing minimization for 1-page and 2-page drawings of graphs with bounded treewidth. *Journal of Graph Algorithms and Applications*, 22(4):577–606, 2018.
- 6 Giuseppe Di Battista, Giuseppe Liotta, and Francesco Vargiu. Spirality and optimal orthogonal drawings. *SIAM J. Comput.*, 27(6):1764–1811, 1998. doi:10.1137/S0097539794262847.
- 7 Sandeep N. Bhatt and Stavros S. Cosmadakis. The complexity of minimizing wire lengths in VLSI layouts. *Inf. Process. Lett.*, 25(4):263–267, 1987. doi:10.1016/0020-0190(87)90173-6.
- 8 Sujoy Bhore, Robert Ganian, Fabrizio Montecchiani, and Martin Nöllenburg. Parameterized algorithms for book embedding problems. *Journal of Graph Algorithms and Applications*, 24(4):603–620, 2020.

- 9 Therese Biedl. On area-optimal planar graph drawings. 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 198–210. Springer, 2014. doi:10.1007/978-3-662-43948-7_17.
- 10 Therese Biedl and Debajyoti Mondal. On upward drawings of trees on a given grid. In Fabrizio Frati and Kwan-Liu Ma, editors, *Proc. 25th International Symposium on Graph Drawing and Network Visualization (GD)*, volume 10692 of *LNCS*, pages 318–325. Springer, 2017. doi:10.1007/978-3-319-73915-1_25.
- 11 Therese Biedl and Martin Vatshelle. The point-set embeddability problem for plane graphs. *Int. J. Comput. Geom. Appl.*, 23(4-5):357–396, 2013. doi:10.1142/S0218195913600091.
- 12 Daniel Bienstock. Some provably hard crossing number problems. *Discrete & Computational Geometry*, 6(3):443–459, 1991.
- 13 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.
- 14 Stina S Bridgeman, Giuseppe Di Battista, Walter Didimo, Giuseppe Liotta, Roberto Tamassia, and Luca Vismara. Turn-regularity and optimal area drawings of orthogonal representations. *Computational Geometry*, 16(1):53–93, 2000.
- 15 Sergio Cabello and Bojan Mohar. Adding one edge to planar graphs makes crossing number and 1-planarity hard. *SIAM Journal on Computing*, 42(5):1803–1829, 2013.
- 16 Hubert Chan. A parameterized algorithm for upward planarity testing. In *European Symposium on Algorithms, ESA*, pages 157–168. Springer, 2004.
- 17 Yi-Jun Chang and Hsu-Chun Yen. On bend-minimized orthogonal drawings of planar 3-graphs. 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 29:1–29:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.SoCG.2017.29.
- 18 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 19 Giordano Da Lozzo, David Eppstein, Michael T. Goodrich, and Siddharth Gupta. Subexponential-time and FPT algorithms for embedded flat clustered planarity. In *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, pages 111–124, 2018. doi:10.1007/978-3-030-00256-5_10.
- 20 Giordano Da Lozzo, David Eppstein, Michael T. Goodrich, and Siddharth Gupta. C-planarity testing of embedded clustered graphs with bounded dual carving-width. *Algorithmica*, 83(8):2471–2502, 2021. doi:10.1007/s00453-021-00839-2.
- 21 Peter Damaschke. Enumerating grid layouts of graphs. *J. Graph Algorithms Appl.*, 24(3):433–460, 2020.
- 22 Emilio Di Giacomo, Giuseppe Liotta, and Fabrizio Montecchiani. Orthogonal planarity testing of bounded treewidth graphs. *Journal of Computer and System Sciences*, 125:129–148, 2022. doi:10.1016/j.jcss.2021.11.004.
- 23 Walter Didimo, Siddharth Gupta, Philipp Kindermann, Giuseppe Liotta, Alexander Wolff, and Meirav Zehavi. Parameterized approaches to orthogonal compaction. In Leszek Gasiencic, editor, *SOFSEM 2023: Theory and Practice of Computer Science - 48th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2023, Nový Smokovec, Slovakia, January 15-18, 2023, Proceedings*, volume 13878 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 2023. doi:10.1007/978-3-031-23101-8_8.
- 24 Walter Didimo and Giuseppe Liotta. Computing orthogonal drawings in a variable embedding setting. In *Proceedings of the 9th International Symposium on Algorithms and Computation, ISAAC*, pages 80–89. Springer, 1998.

- 25 Walter Didimo, Giuseppe Liotta, and Maurizio Patrignani. Bend-minimum orthogonal drawings in quadratic time. In Therese Biedl and Andreas Kerren, editors, *Graph Drawing and Network Visualization - 26th International Symposium, GD 2018, Barcelona, Spain, September 26-28, 2018, Proceedings*, volume 11282 of *Lecture Notes in Computer Science*, pages 481–494. Springer, 2018. doi:10.1007/978-3-030-04414-5_34.
- 26 William S. Evans, Krzysztof Fleszar, Philipp Kindermann, Noushin Saeedi, Chan-Su Shin, and Alexander Wolff. Minimum rectilinear polygons for given angle sequences. *Comput. Geom.*, 100:101820, 2022. doi:10.1016/j.comgeo.2021.101820.
- 27 Mike Fellows, Panos Giannopoulos, Christian Knauer, Christophe Paul, Frances A. Rosamond, Sue Whitesides, and Nathan Yu. Milling a graph with turn costs: A parameterized complexity perspective. In *Proceedings of the 36th International Workshop on Graph Theoretic Concepts in Computer Science, WG*, pages 123–134, 2010.
- 28 Robert Ganian, Fabrizio Montecchiani, Martin Nöllenburg, and Meirav Zehavi. Parameterized complexity in graph drawing (dagstuhl seminar 21293). *Dagstuhl Reports*, 11(6):82–123, 2021.
- 29 Michael R Garey and David S Johnson. Crossing number is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983.
- 30 Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001. doi:10.1137/S0097539794277123.
- 31 Angelo Gregori. Unit-length embedding of binary trees on a square grid. *Information Processing Letters*, 31(4):167–173, 1989.
- 32 Martin Grohe. Computing crossing numbers in quadratic time. *Journal of Computer and System Sciences*, 68(2):285–302, 2004.
- 33 Siddharth Gupta, Guy Sa’ar, and Meirav Zehavi. Drawn tree decomposition: New approach for graph drawing problems, 2023. arXiv:2310.05471.
- 34 Siddharth Gupta, Guy Sa’ar, and Meirav Zehavi. Grid recognition: Classical and parameterized computational perspectives. *Journal of Computer and System Sciences*, 136:17–62, 2023. doi:10.1016/j.jcss.2023.02.008.
- 35 Patrick Healy and Karol Lynch. Two fixed-parameter tractable algorithms for testing upward planarity. *International Journal of Foundations of Computer Science*, 17(05):1095–1114, 2006.
- 36 Petr Hliněný. Crossing number is hard for cubic graphs. *Journal of Combinatorial Theory, Series B*, 96(4):455–471, 2006.
- 37 Petr Hliněný and Abhisekh Sankaran. Exact crossing number parameterized by vertex cover. In *Proceedings of the 27th International Symposium on Graph Drawing and Network Visualization, GD*, pages 307–319, 2019.
- 38 Ken-ichi Kawarabayashi and Bruce Reed. Computing crossing number in linear time. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, STOC*, pages 382–390, 2007.
- 39 Marcus Krug and Dorothea Wagner. Minimizing the area for planar straight-line grid drawings. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Graph Drawing, 15th International Symposium, GD 2007, Sydney, Australia, September 24-26, 2007. Revised Papers*, volume 4875 of *Lecture Notes in Computer Science*, pages 207–212. Springer, 2007. doi:10.1007/978-3-540-77537-9_21.
- 40 Giuseppe Liotta, Ignaz Rutter, and Alessandra Tappini. Parameterized complexity of graph planarity with restricted cyclic orders. *J. Comput. Syst. Sci.*, 135:125–144, 2023. doi:10.1016/j.jcss.2023.02.007.
- 41 Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.
- 42 Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32(3):265–279, 1986. doi:10.1016/0022-0000(86)90030-9.
- 43 Maurizio Patrignani. On the complexity of orthogonal compaction. *Computational Geometry*, 19(1):47–67, 2001.

- 44 Michael J. Pelsmajer, Marcus Schaefer, and Daniel Stefankovic. Crossing numbers of graphs with rotation systems. *Algorithmica*, 60(3):679–702, 2011.
- 45 Md. Saidur Rahman, Noritsugu Egi, and Takao Nishizeki. No-bend orthogonal drawings of subdivisions of planar triconnected cubic graphs. *IEICE Trans. Inf. Syst.*, 88-D(1):23–30, 2005. URL: http://search.ieice.org/bin/summary.php?id=e88-d_1_23&category=D&year=2005&lang=E&abst=.
- 46 Marcus Schaefer. Complexity of some geometric and topological problems. In *Proceedings of the 18th International Symposium on Graph Drawing and Network Visualization, GD*, pages 334–344. Springer, 2009.
- 47 Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987. doi:10.1137/0216030.
- 48 Meirav Zehavi. Parameterized analysis and crossing minimization problems. *Computer Science Review*, 45:100490, 2022. doi:10.1016/j.cosrev.2022.100490.
- 49 Xiao Zhou and Takao Nishizeki. Orthogonal drawings of series-parallel graphs with minimum bends. *SIAM J. Discret. Math.*, 22(4):1570–1604, 2008. doi:10.1137/060667621.

A Preliminaries

In this paper, we only consider finite simple undirected graphs, unless stated otherwise. Moreover, we refer to straight line segments as line segments, unless stated otherwise. Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. For $k, i, j \in \mathbb{N}$, we denote $[k] = \{1, 2, \dots, k\}$ and $[i, j] = \{i, i + 1, \dots, j\}$.

A.1 Graph Notation and Decompositions

For a graph $G = (V, E)$ and a subset of vertices $U \subseteq V$, we denote by $G[U]$ the subgraph of G induced by U . For a given subset $V' \subseteq V$ of vertices, we define the *boundary* of V' as the set of vertices in V' that are adjacent to a vertex in $V \setminus V'$:

► **Definition A.1 (Boundary).** Let $G = (V, E)$ be a graph. Let $V' \subseteq V$. Then the boundary of V' in G , denoted by $B_G(V')$, is the set of vertices of V' that have a neighbor in $V \setminus V'$, i.e., $B_G(V') = \{v' \in V' \mid \text{there exists } v \in V \setminus V' \text{ such that } \{v, v'\} \in E\}$.

When the graph G is clear from the context, we drop it from the subscript. Given a path P , we represent P as a sequence of vertices v_1, v_2, \dots, v_k , such that $\{v_i, v_{i+1}\}$ is an edge in P for every $1 \leq i \leq k - 1$. Similarly, given a cycle C , we represent C as a sequence of vertices v_1, v_2, \dots, v_k , such that $v_1 = v_k$ and $\{v_i, v_{i+1}\}$ is an edge in C for every $1 \leq i \leq k - 1$. Note that we use the terms path and cycle to refer to simple path and cycles. We now define the concepts of a *tree decomposition* and a *path decomposition*.

► **Definition A.2 (Tree Decomposition).** A tree decomposition of a graph $G = (V, E)$ is a pair $(\mathcal{T} = (V_T, E_T), \beta : V_T \rightarrow 2^V)$ where \mathcal{T} is a tree such that:

1. For every $v \in V$, the subgraph of \mathcal{T} induced by $\{x \in V_T \mid v \in \beta(x)\}$ is non-empty and connected.
2. For every $\{u, v\} \in E$, there exists $x \in V_T$ such that $\{u, v\} \subseteq \beta(x)$.

The width of (\mathcal{T}, β) is defined to be $\max_{x \in V_T} |\beta(x)| - 1$. For every $x \in V_T$, $\beta(x)$ is called a bag. The treewidth of a graph G is the minimum width of any tree decomposition of G .

► **Definition A.3 (Path Decomposition).** A path decomposition of a graph $G = (V, E)$ is a pair $(\mathcal{P} = (V_P, E_P), \beta : V_P \rightarrow 2^V)$ where \mathcal{P} is a path such that:

1. For every $v \in V$, the subgraph of \mathcal{P} induced by $\{x \in V_P \mid v \in \beta(x)\}$ is non-empty and connected.
2. For every $\{u, v\} \in E$, there exists $x \in V_P$ such that $\{u, v\} \subseteq \beta(x)$.

The width of (\mathcal{P}, β) is defined to be $\max_{x \in V_P} |\beta(x)| - 1$. For every $x \in V_P$, $\beta(x)$ is called a bag. The pathwidth of a graph G is the minimum width of any path decomposition of G .

A.2 Graph Drawing

For a given graph G , a *drawing* of G on the plane is a mapping of the vertices to distinct points of \mathbb{R}^2 and of the edges to simple curves in \mathbb{R}^2 , connecting the images of their endpoints. A drawing of a graph is *planar* if no pair of edges, or an edge and a vertex, cross except at a common endpoint. Two planar drawings of the same graph are *equivalent* if they determine the same *rotation* at each vertex, that is, the same circular ordering for the edges around each vertex. An *embedding* is an equivalence class of planar drawings.

Given a drawing d of G , we represent d as a pair of functions (d_V, d_E) as follows. The function $d_V : V \rightarrow \mathbb{R} \times \mathbb{R}$ is an injection, which maps each vertex v of G to a point (i, j) in the plane; then, i and j are also denoted as $d_x(v)$ and $d_y(v)$, respectively, that is, $d_V(v) = (d_x(v), d_y(v))$. The function $d_E : E \rightarrow \mathcal{C}$, where \mathcal{C} is the set of all simple curves in the plane, maps each edge $\{u, v\} \in E$ to a simple curve $c \in \mathcal{C}$ between $d_V(u)$ and $d_V(v)$. For simplicity, we refer to (d_V, d_E) as one function, $d : V \cup E \rightarrow \{\mathbb{R} \times \mathbb{R}\} \cup \mathcal{C}$, such that $d(v) = d_V(v)$ for every $v \in V$, and $d(\{u, v\}) = d_E(\{u, v\})$ for every $\{u, v\} \in E$. We call V and E the *vertex set* and the *edge set associated with d* , respectively. Let d be a drawing of a graph G , and let $p \in \mathbb{R}^2$ be a point. We say that p is *on d* if p is on the image of an edge of G in d or p is the image of a vertex of G in d . We denote by $\text{PlanePoints}(d)$ the set of points on d .

For two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ in the plane, we denote the line segment joining the points by $\ell(p_1, p_2)$. For four points, $p_i = (x_i, y_i) \in \mathbb{R}^2$ for every $1 \leq i \leq 4$, we say that $\ell(p_1, p_2)$ *crosses* $\ell(p_3, p_4)$ if the line segments $\ell(p_1, p_2)$ and $\ell(p_3, p_4)$ cross except at $p_i = (x_i, y_i)$ for every $1 \leq i \leq 4$. Let a and b be two points in \mathbb{R}^2 and let $\epsilon > 0$. We denote $\ell(a, a_\epsilon)$ by $\text{line}_\epsilon(a, b)$, where a_ϵ is the point on the line $\ell(a, b)$ at distance ϵ from a if it exists. For a pair of points (p_1, p_2) , and a point p' , where $p_1, p_2, p' \in \mathbb{R}^2$, we say that $\ell(p_1, p_2)$ *intersects* p' if p' is on the line $\ell(p_1, p_2)$, including its endpoints. We use the term *grid points* to refer to the infinite set of points $(x, y) \in \mathbb{R}^2$ where $x, y \in \mathbb{N}_0$. Given two distinct grid points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, we say that $p_1 < p_2$ if $x_1 < x_2$ or $x_1 = x_2$ and $y_1 < y_2$.

A *drawn graph* is a graph with a prescribed drawing. A *plane graph* is a drawn graph whose prescribed drawing is planar. A drawing of a graph is called a *straight-line* drawing if the edges are mapped to line segments, connecting the images of their endpoints. We define a *straight-line path (cycle)* as a plane path (cycle), where the vertices are mapped to grid points and edges are mapped to line segments connecting the images of their endpoints. We denote by $\mathcal{P} \subset \mathcal{C}$ the (infinite) set of straight-line paths in \mathbb{R}^2 . Moreover, we alternatively denote any path $P = (v_1, \dots, v_k) \in \mathcal{P}$ by the sequence (p_1, \dots, p_k) , where $p_i \in \mathbb{R}^2$ is the image of the vertex v_i in P , for every $1 \leq i \leq k$. We define an *axis-parallel path (cycle)* as a straight-line path (cycle), where every edge of the path is parallel to the X - or Y - axis. For an axis-parallel path $P = (p_1, \dots, p_k)$, we denote by $|P|$ the (Euclidean) *length* of P , that is, $|P| = |p_2 - p_1| + |p_3 - p_2| + \dots + |p_k - p_{k-1}|$. Next, we define a *grid drawing* of a graph G as a straight-line drawing of G where the vertices are mapped to grid points and the edges are mapped to (axis-parallel) unit length line segments (e.g., see Figure 11b):

► **Definition A.4 (Straight-Line Grid Drawing).** Let G be a graph. A straight-line grid drawing d of G is a straight-line drawing d of G such that (i) for every $u \in V$, $d(u)$ is a grid point (ii) For every $\{u, v\}, \{u', v'\} \in E$, $d(\{u, v\})$ and $d(\{u', v'\})$ are intersected in at most one point.

► **Definition A.5 (Grid Drawing).** Let $G = (V, E)$ be a graph. A grid drawing d of G is a drawing $d : V \cup E \rightarrow \mathbb{N}_0 \times \mathbb{N}_0 \cup \mathcal{P}$ such that if $\{u, v\} \in E$ then $|d_x(u) - d_x(v)| + |d_y(u) - d_y(v)| = 1$.

We now extend the concept of a grid drawing to a *rectilinear grid drawing*, where the edges are mapped to variable length line segments parallel to the axes (e.g., see Figure 11c):

► **Definition A.6 (Rectilinear Grid Drawing).** Let $G = (V, E)$ be a graph. A rectilinear grid drawing d of G is a drawing $d : V \cup E \rightarrow \mathbb{N}_0 \times \mathbb{N}_0 \cup \mathcal{P}$ of G , such that for every edge $\{u, v\} \in E$, $d(\{u, v\})$ is a line segment between $d(u)$ and $d(v)$ such that $d_x(u) = d_x(v)$ or $d_y(u) = d_y(v)$.

Further, we extend the concept of a rectilinear grid drawing to an *orthogonal grid drawing*, where the edges are mapped to straight-line paths, such that the edges of these paths are mapped to line segments parallel to the axes (e.g., see Figure 11d):

► **Definition A.7 (Orthogonal Grid Drawing).** Let $G = (V, E)$ be a graph. An orthogonal grid drawing d of G is a drawing $d : V \cup E \rightarrow \mathbb{N}_0 \times \mathbb{N}_0 \cup \mathcal{P}$ of G , such that for every edge $\{u, v\} \in E$, $d(\{u, v\})$ is an axis-parallel path between $d(u)$ and $d(v)$.

Finally, we extend the concept of an orthogonal grid drawing to a *polyline grid drawing*, where the edges are mapped to straight-line paths instead of axis-parallel paths (e.g., see Figure 11e).

► **Definition A.8 (Polyline Grid Drawing).** Let $G = (V, E)$ be a graph. A polyline grid drawing d of G is a drawing $d : V \cup E \rightarrow \mathbb{N}_0 \times \mathbb{N}_0 \cup \mathcal{P}$ of G .

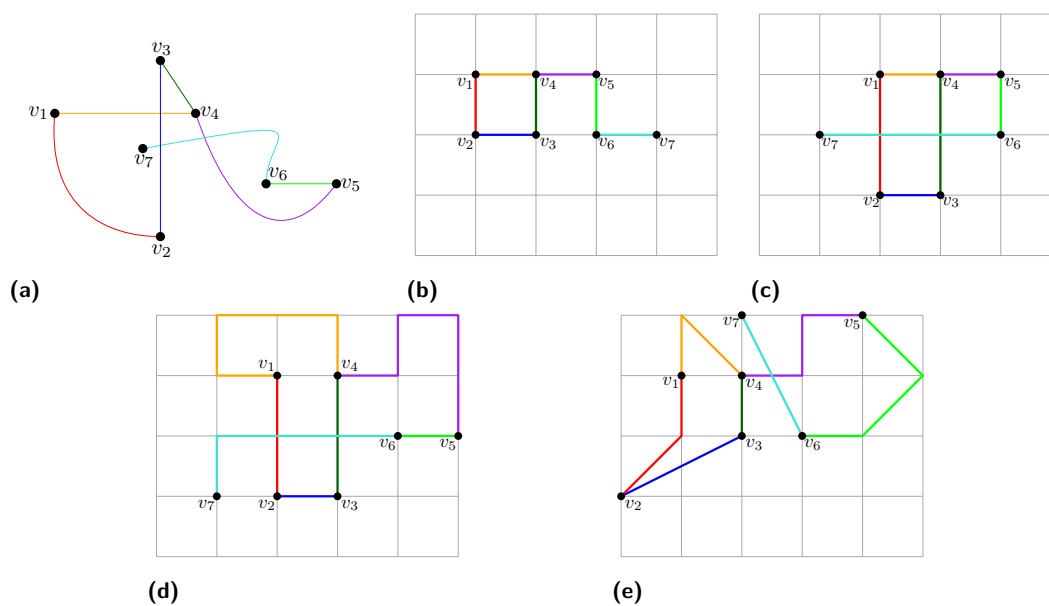
A.3 Problem Definitions

In this subsection, we give the definitions for the problems we will solve using our new concept.

► **Definition A.9 (Grid Recognition Problem).** The GRID RECOGNITION problem is, given a graph G , to determine whether G has a grid drawing.

► **Definition A.10 (Crossing Minimization Problem on Straight-Line Grid Drawings).** The STRAIGHT-LINE GRID CROSSING MINIMIZATION problem is, given a graph G and $h, w \in \mathbb{N}$, to construct a straight-line grid drawing d of G (if one exists) such that: (i) d is strictly bounded by $R_{h,w}$, (ii) d has minimum number of crossings out of all the straight-line grid drawings of G which are strictly bounded by $R_{h,w}$. If such a drawing does not exist, return “no-instance”.

In the ORTHOGONAL COMPACTION problem we get a connected graph G . We assume to have an order on the vertices, that is, for every $u, v \in V$ such that $u \neq v$, either $u > v$ or $v < u$. In addition to G , we have, for every $\{u, v\} \in E$ where $u > v$, the relative position of v compared to u , that is, the *direction* of the $\{u, v\}$ from u to v . We denote these directions by U, D, L and R; this stands for “up”, “down”, “left” and “right”, respectively. We assume that there exists a planar rectilinear grid drawing of G such that for every $\{u, v\} \in E$, the relative position of v compared to u is as given as input. Our goal is to find such a drawing of minimum area. We start by defining the problem formally. For this purpose, we first have the following definition:



■ **Figure 11** Different drawings (defined in Definitions A.5-A.8) of the graph G shown in (a). A grid, a rectilinear grid, an orthogonal grid and a polyline grid drawings of G are shown in (b), (c), (d) and (e), respectively.

► Definition A.11 (Drawing Respects an Edge Direction). Let G be a connected graph, let $\{u, v\} \in E$ such that $u > v$, and let $\text{dir}_{\{u,v\}} \in \{U, D, L, R\}$. Let d be a rectilinear grid drawing of G . We say that d respects $\text{dir}_{\{u,v\}}$ if the following conditions are satisfied

1. If $\text{dir}_{\{u,v\}} = U$, then $d_x(v) = d_x(u)$ and $d_y(v) > d_y(u)$.
2. If $\text{dir}_{\{u,v\}} = D$, then $d_x(v) = d_x(u)$ and $d_y(v) < d_y(u)$.
3. If $\text{dir}_{\{u,v\}} = L$, then $d_y(v) = d_y(u)$ and $d_x(v) < d_x(u)$.
4. If $\text{dir}_{\{u,v\}} = R$, then $d_y(v) = d_y(u)$ and $d_x(v) > d_x(u)$.

Now, we define the problem **ORTHOGONAL COMPACTION** as follows:

► Definition A.12 (Orthogonal Compaction Problem). Let G be a connected graph. For every $\{u, v\} \in E$ let $\text{dir}_{\{u,v\}} \in \{U, D, L, R\}$. The **ORTHOGONAL COMPACTION** problem is to find a planar rectilinear grid drawing d of G such that (i) for every $\{u, v\} \in E$, d respects $\text{dir}_{\{u,v\}}$, and (ii) d is strictly bounded by $R_{h,w}$ such that $(h-1) \cdot (w-1)$ is minimum.



Single Machine Scheduling with Few Deadlines

Klaus Heeger  

Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Danny Hermelin  

Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Dvir Shabtay  

Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

We study single-machine scheduling problems with few deadlines. We focus on two classical objectives, namely minimizing the weighted number of tardy jobs and the total weighted completion time. For both problems, we give a pseudopolynomial-time algorithm for a constant number of different deadlines. This algorithm is complemented with an ETH-based, almost tight lower bound. Furthermore, we study the case where the number of jobs with a nontrivial deadline is taken as parameter. For this case, the complexity of our two problems differ: Minimizing the total number of tardy jobs becomes fixed-parameter tractable, while minimizing the total weighted completion time is W[1]-hard.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Theory of computation → W hierarchy; Theory of computation → Dynamic programming; Theory of computation → Scheduling algorithms

Keywords and phrases Single-machine scheduling, weighted completion time, tardy jobs, pseudopolynomial algorithms, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.24

Funding Supported by the ISF, grant No. 1070/20.

1 Introduction

Already since the 1950s, scheduling has been an important area of combinatorial optimization, with various applications coming from a broad range of areas such as manufacturing, management, and healthcare [2, 19]. This led to a wide variety of different scheduling problems, depending on the targeted applications. What almost all scheduling problems have in common is that there is a set of jobs $\{1, \dots, n\}$ with different characteristics that need to be processed on one or several machines, subject to some feasibility constraints, and with the objective of optimizing a predefined objective function. Usually, the objective function is based on the completion times of the jobs in the proposed solution schedule.

One very basic scheduling setting is the following: We are given a set of n jobs, all available to be non-preemptively processed on a single machine at time zero. Each job j has a processing time p_j , a weight w_j (corresponding to its importance), and a due date d_j . The jobs must be processed one after another on a single machine. In this setting, the *completion time* C_j of a job j is simply the sum of processing times of all jobs scheduled before j (including j itself). A job is *tardy* if its completion time is larger than its due date. The tardiness of a job can also be expressed using the *unit-penalty* function U_j which is one if job j is tardy and zero otherwise.



© Klaus Heeger, Danny Hermelin, and Dvir Shabtay;
licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 24; pp. 24:1–24:15

Leibniz International Proceedings in Informatics



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

Two of the most-common objective functions in single-machine scheduling are the total weighted completion time $\sum w_j C_j$, and the total weighted number of tardy jobs $\sum w_j U_j$. In the classic 3-field notation by Graham *et al.* [8], these two problems are denoted by $1||\sum w_j C_j$ and $1||\sum w_j U_j$.

A less well-studied generalization of these two problems is when the jobs also have deadlines $\bar{d}_1, \dots, \bar{d}_n$. Deadlines differ from due dates in that they must be met, and so scheduling problems with deadlines require solution schedules to have all jobs complete before their deadline. Adding job deadlines to $1||\sum w_j C_j$ and $1||\sum w_j U_j$ results in problems $1|\bar{d}_j|\sum w_j C_j$ and $1|\bar{d}_j|\sum w_j U_j$. Both these problems have natural applications in practice: In the $1|\bar{d}_j|\sum w_j C_j$ problem we wish to minimize the total completion time (or equivalently average completion time) under the additional deadline constraints, while in $1|\bar{d}_j|\sum w_j U_j$ we can interpret the weight of a job as a premium for early completion of the job (before its due date instead of before its deadline) [9].

Note that the two problems above have quite efficient solutions when the jobs have no deadlines. The $1||\sum w_j C_j$ problem can be solved in $O(n \log n)$ time [22] by scheduling all jobs according to the weighted shortest processing time (WSPT) order. The $1||\sum w_j U_j$ problem is weakly NP-hard [14], but it admits an $O(Pn)$ pseudo-polynomial time algorithm [16], where $P = \sum p_j$ is the total sum of processing times of all input jobs, and an $O(n \log n)$ time algorithm [18] when all jobs have unit weight (*i.e.* $w_j = 1$ for any job j). Thus, it is natural to ask whether these results can be generalized to the case where there are only a few different deadlines among all n jobs, or when only a few jobs have nontrivial deadlines (*i.e.*, jobs j with deadline $\bar{d}_j < P$). This question is the starting point of this paper.

Our Contribution. We investigate the parameterized complexity of $1|\bar{d}_j|\sum w_j U_j$ and $1|\bar{d}_j|\sum w_j C_j$ with respect to two different parameters: The first is the number of different deadlines among all n jobs, and the second is the number of jobs with a nontrivial deadline, *i.e.* the number of jobs j with $\bar{d}_j < P$. Note that the latter parameter upper bounds the former. For the number of different deadlines parameter, both problems behave similarly: They are weakly NP-hard already for two different deadlines (one of which is a “trivial” deadline equaling the total processing time). When considering the case where all numbers (processing times and weights) are encoded in unary, both problems are W[1]-hard but admit XP-time algorithms.

For the number of jobs with nontrivial deadline parameter, the complexity of both problems diverge: For $1|\bar{d}_j|\sum w_j C_j$, the hardness results for the number of different deadlines carry over. For $1|\bar{d}_j|\sum w_j U_j$, however, there is an $O(2^k \cdot P \cdot n)$ -time algorithm. For the special case of unit weights (*i.e.* $w_j = 1$ for all j), this algorithm can be improved to an $O(2^k \cdot n \log n)$ -time algorithm. We refer to Table 1 for an overview of our results.

■ **Table 1** Overview of our results. We use n to denote the number of jobs, and P to denote the total processing time of all jobs.

	$k = \#$ different deadlines	$k = \#$ jobs with deadline
$1 \bar{d}_j \sum w_j U_j$	weakly NP-hard even if $k = 2$ and $w_j = 1$ (3.1) W[1]-hard with unary input even if $w_j = 1$ (3.1) $P^{O(k)}$ -time algorithm (3.4)	weakly NP-hard if $k = 0$ [14] $O(2^k n \log n)$ if $w_j = 1$ (3.6) $O(2^k Pn)$ (3.5)
$1 \bar{d}_j \sum w_j C_j$	weakly NP-hard even if $k = 2$ [17] W[1]-hard even with unary input (4.2) $P^{O(k)}$ -time algorithm (4.4)	

Related Work. While there are numerous papers on single-machine scheduling, rather few consider deadlines. Regarding the first problem we study, $1|\bar{d}_j|\sum w_j U_j$, Lawler [15] showed that $1|\bar{d}_j|\sum U_j$ (the unit weight special case of $1|\bar{d}_j|\sum w_j U_j$) is weakly NP-hard. This result was strengthened by Yuan [24] who showed that $1|\bar{d}_j|\sum U_j$ is strongly NP-hard. Huo *et al.* [12] showed that $1|\bar{d}_j|\sum U_j$ can be solved in polynomial time if either $d_i \leq d_j$ implies $\bar{d}_i \leq \bar{d}_j$ and $p_i \leq p_j$, or if $p_i \leq p_j$ implies $d_i - p_i \geq d_j - p_j$. The running times of some of these algorithms were improved later [11]. The problem $1|\bar{d}_j|\sum w_j U_j$ was also studied from a practical point of view: Hariri and Potts [9] designed a branch-and-bound based algorithm for it, while Baptiste *et al.* [1] designed an ILP-formulation with n variables and $2n$ constraints for the problem.

Concerning the second problem we study, $1|\bar{d}_j|\sum w_j C_j$, Lenstra *et al.* [17] proved that the problem is weakly NP-hard even if only one job has a nontrivial deadline. A related problem is $1|r_j, \bar{d}_j, \text{pmtn}|\sum C_j$. Here, the jobs additionally have release dates (that is, a job cannot be scheduled before its release date) and preemption is allowed (that is, it is possible to process a job j partially, then schedule other jobs, and later continue processing job j). Further, all jobs have weight 1. Wan *et al.* [23] showed weak NP-hardness of $1|r_j, \bar{d}_j, \text{pmtn}|\sum C_j$. Recently, strong NP-hardness was shown [4]. In case of “agreeable” processing times and deadlines (that is, whenever the deadline of job j is larger than the one of job j' , then also the processing time of j is larger than the processing time of j'), the problem is solvable in polynomial time [10].

Chen and Yuan [3] studied the problem of minimizing the total tardiness $\sum T_j$ on a single machine with deadlines (the tardiness of a job j is $T_j = \max\{0, C_j - d_j\}$). Chen and Yuan [3] showed that this problem (denoted $1|\bar{d}_j|\sum T_j$) is strongly NP-hard. A similar problem is that of minimizing the total late work $\sum Y_j$ of all jobs, where the late work of a job j is $Y_j = \min\{p_j, T_j\}$. Chen *et al.* [5] showed $1|\bar{d}_j|\sum Y_j$ is strongly NP-hard. However, if all jobs share a common due date, then the problem is weakly NP-hard and admits a pseudopolynomial-time algorithm. Chen *et al.* also showed that a few special cases of the problem become polynomial-time solvable.

2 Preliminaries

We consider non-preemptive scheduling problems on a single machine. Here, the input consists of n jobs $\{1, \dots, n\}$ which are all available to be processed at time zero. We denote by $[n] := \{1, 2, \dots, n\}$. Each job j is characterized by its *processing time* $p_j \in \mathbb{N}$, its *weight* $w_j \in \mathbb{N}$, and its *deadline* $\bar{d}_j \in \mathbb{N}$. In one of the problems we consider below, each job j will also have a *due date* $d_j \in \mathbb{N}$. We assume without loss of generality that $d_j \leq \bar{d}_j$ holds for every job $j \in [n]$. We denote the total processing time of all jobs by $P = \sum_{j \in [n]} p_j$, and their total weight by $W = \sum_{j \in [n]} w_j$.

A *schedule* is a permutation $\sigma : [n] \rightarrow [n]$ of the jobs. Given a schedule σ , the *completion time* $C_j(\sigma)$ of job j is $C_j(\sigma) := \sum_{i \in [n]: \sigma(i) \leq \sigma(j)} p_i$; that is, it is the total processing times of all jobs preceding j in the schedule (including j itself). A schedule σ is *feasible* if $C_j(\sigma) \leq \bar{d}_j$ for all $j \in [n]$. A job j is *early* in σ if $C_j(\sigma) \leq d_j$, and otherwise it is *tardy* in σ . We use $U(\sigma)$ to denote the set of all jobs that are tardy in σ . We call a deadline \bar{d}_j *trivial* if $\bar{d}_j \geq P$ (that is, any schedule fulfills this deadline). We assume without loss of generality that $\bar{d}_j = P$ is the only occurring trivial deadline.

24:4 Single Machine Scheduling with Few Deadlines

We focus on instances with few different deadlines. Thus, we set $\bar{d}^{(1)}$ to be the smallest appearing deadline, $\bar{d}^{(2)}$ to be the second-smallest appearing deadline (different from $\bar{d}^{(1)}$), and so on. Note that the largest deadline $\bar{d}^{(k)}$ will always be P . For each $i \in [k]$, we denote by $J^{(i)}$ the set of jobs with deadline $\bar{d}^{(i)}$, and by $P^{(i)}$ the total processing time of all jobs in $J^{(i)}$.

We study two problems in this paper that differ according to the objective function used to evaluate feasible schedules:

$$1|\bar{d}_j|\sum w_j U_j$$

Input: A set of n jobs with due dates, a number b .

Question: Is there a feasible schedule σ such that $\sum_{j \in U(\sigma)} w_j \leq b$?

$$1|\bar{d}_j|\sum w_j C_j$$

Input: A set of n jobs, a number b .

Question: Is there a feasible schedule σ such that $\sum_{j \in [n]} w_j C_j(\sigma) \leq b$?

The problem names are derived from the classical 3-field notation by Graham *et al.* [8], where the first field encodes the machine setting (in our case, “1” represents a single machine), the second field contains constraints (in our setting, \bar{d}_j indicates the existence of deadlines), and the third field containing the objective function (in our case either the weighted completion time $\sum w_j C_j$ or the weighted number of late jobs $\sum w_j U_j$).

3 The $1|\bar{d}_j|\sum w_j U_j$ problem

In this section, we study the problem of minimizing the weighted number of tardy jobs. We start by showing some hardness results for $1|\bar{d}_j|\sum w_j U_j$ in Section 3.1. Afterwards, we give a pseudopolynomial-time algorithm for constant number of different deadlines in Section 3.2. Finally, in Section 3.3, we design efficient algorithms for the case of few jobs having a nontrivial deadline.

3.1 Hardness results

Recently, Yuan [24] showed that $1|\bar{d}_j|\sum U_j$ is strongly NP-hard via a reduction from 3-PARTITION. This NP-hard problem is a special case of the following MULTIWAY NUMBER PARTITIONING partition problem:

MULTIWAY NUMBER PARTITIONING

Input: Integers m and k , and $t := m \cdot k$ numbers a_1, \dots, a_t .

Question: Is there a partition (A_1, \dots, A_k) of $\{a_1, \dots, a_t\}$ such that $|A_i| = m$ for each $i \in [k]$ and $\sum_{a \in A_i} a = \sum_{a \in A_j} a$ for all $i, j \in [k]$?

Observe that MULTIWAY NUMBER PARTITIONING with $k = 2$ is known as the weakly NP-hard EQUAL CARDINALITY PARTITION problem [7]. Furthermore, the classical BIN PACKING problem naturally reduces to MULTIWAY NUMBER PARTITIONING by adding items of zero size in order to obtain an instance with $t = m \cdot k$ numbers.

Yuan [24] presented a reduction from MULTIWAY NUMBER PARTITIONING with $m = 3$ to $1|\bar{d}_j|\sum U_j$. His reduction uses in its construction the sum of all input numbers $B := \sum_{\ell=1}^t a_\ell$ and a sufficiently large number $M = \frac{3}{2}t(t+1)B + 1$. It creates a job (i, j) for each $i \in [k]$ and $j \in [t]$ with processing time $p_{i,j} = M^2 + i \cdot (M + a_j)$. For each $i \in [k]$, the jobs (i, j) share the same deadline $\bar{d}_{i,j} = \bar{d}^{(i)} = \sum_{\ell=1}^{i-1} P^{(\ell)} + t \cdot \sum_{\ell=1}^m P^{(\ell)}$, where

$P^{(\ell)} = 3M^2 + 3\ell M + 3\ell B$. Furthermore, for each $j \in [t]$, the jobs (i, j) share the same due date $d_{i,j} = d^{(j)} = j \cdot M^2 + \frac{3}{2}Mt(t+1)B$. Yuan [24] proved that if the input MULTIWAY NUMBER PARTITIONING instance $(3, k, a_1, \dots, a_t)$ is a yes-instance then the constructed job set has a feasible schedule with at most $b = 3t^2 - t$ tardy jobs.

Note that the reduction from MULTIWAY NUMBER PARTITIONING of Yuan [24] described above works for any value of $m \geq 3$ and $k \geq 2$. Setting $m = t/2$ and $k = 2$ results in a reduction from EQUAL CARDINALITY PARTITION to $1|\bar{d}_j| \sum U_j$ with two different deadlines. Using arbitrary m and k results in a reduction from BIN PACKING with k bins to $1|\bar{d}_j| \sum U_j$ with k different deadlines. It is known that BIN PACKING with unary encoded numbers is W[1]-hard parameterized by the number k of bins, and assuming ETH it cannot be solved in $f(k) \cdot n^{o(k/\log k)}$ time [13]. Thus, we directly get the following hardness result for $1|\bar{d}_j| \sum U_j$:

► **Theorem 3.1.** *Let k denote the number of different deadlines in a $1|\bar{d}_j| \sum U_j$ instance. Then the $1|\bar{d}_j| \sum U_j$ problem is*

- weakly NP-hard even when $k = 2$,
- W[1]-hard with respect to k even when all numbers are encoded in unary, and
- admits no $f(k) \cdot P^{o(k/\log k)}$ -time algorithm assuming ETH.

3.2 Constant number of deadlines

Let $k = |\{\bar{d}_j : 1 \leq j \leq n\}|$ denote the number of different deadlines in the job instance. By Theorem 3.1, we know that $1|\bar{d}_j| \sum w_j U_j$ is W[1]-hard with respect to k even if $w_j = 1$ for each job j , and all processing times are encoded in unary. Complementing this result, we will show that if $k = O(1)$, then we can solve $1|\bar{d}_j| \sum w_j U_j$ in pseudo-polynomial time. Throughout the subsection we will assume that the input jobs are ordered by ascending due dates, *i.e.* $d_1 \leq \dots \leq d_n$. Furthermore, we denote by $J^{(i)}$ the set of jobs with deadline $\bar{d}^{(i)}$ and by $P^{(i)}$ the total processing time of all jobs from $J^{(i)}$.

Our algorithm for $1|\bar{d}_j| \sum w_j U_j$ is based on dynamic programming. Our algorithm processes the jobs from job 1 to n (*i.e.* according to increasing due date), and creates a table τ_j for each $j \in [n]$. The table τ_j is associated with the job set $\{1, \dots, j\}$, and it contains an entry $\tau_j[x_1, \dots, x_k]$ for each $(x_1, \dots, x_k) \in \{0, \dots, P^{(1)}\} \times \dots \times \{0, \dots, P^{(k)}\}$. The entry $\tau_j[x_1, \dots, x_k]$ shall equal $v_j[x_1, \dots, x_k]$, where we define $v_j[x_1, \dots, x_k]$ to be the maximum number w^* such that there is a feasible schedule of all n jobs fulfilling that

1. the set S of early jobs from $\{1, \dots, j\}$ has weight w^* , and
2. for each $i \in [k]$, the total processing time of early jobs from $S \cap J^{(i)}$ equals x_i .

We stress that all considered schedules schedule all n jobs, even if they correspond to some entry $\tau_j[x_1, \dots, x_k]$ with $j < n$. The minimum value $W - v_n[x_1, \dots, x_k]$ over all $(x_1, \dots, x_k) \in \{0, \dots, P^{(1)}\} \times \dots \times \{0, \dots, P^{(k)}\}$ is the minimum weighted number of tardy jobs of any feasible schedule for our instance. We call a feasible schedule fulfilling the second condition above a $(j; x_1, \dots, x_k)$ -compatible schedule.

Our dynamic program needs to be able to compute $v_j[x_1, \dots, x_k]$, given $v_{j-1}[y_1, \dots, y_k]$ for all y_1, \dots, y_k . There are two cases: First, job j is late in a schedule witnessing the value of $v_j[x_1, \dots, x_k]$. In this case, we have $v_j[x_1, \dots, x_k] = v_{j-1}[x_1, \dots, x_k]$. Second, job j is early in a schedule witnessing the value of $v_j[x_1, \dots, x_k]$. In this case, we have $v_j[x_1, \dots, x_k] = w_j + v_{j-1}[x_1, \dots, x_{i-1}, x_i - p_j, x_{i+1}, \dots, x_k]$ where $i \in [k]$ such that the deadline of j is $\bar{d}^{(i)}$. However, we do not have $v_j[x_1, \dots, x_k] = \max\{v_{j-1}[x_1, \dots, x_k], w_j + v_{j-1}[x_1, \dots, x_{i-1}, x_i - p_j, x_{i+1}, \dots, x_k]\}$ because it is not always possible to modify a schedule corresponding to $v_{j-1}[x_1, \dots, x_{i-1}, x_i - p_j, x_{i+1}, \dots, x_k]$ in such a way that also job j is early.

This is the major difficulty in the design of the dynamic program: Finding a criterion where we can modify the schedule corresponding to $\tau_{j-1}[x_1, \dots, x_{i-1}, x_i - p_j, x_{i+1}, \dots, x_k]$ so we can additionally schedule job j early.

To derive such a criterion, we first observe that if we knew the set of early jobs (from $[n]$, not only from $[j]$), then we can easily compute an optimal schedule: As we have a strict upper bound for the completion time of each job (either its due date if the job is early, or its deadline if the job is late), it is optimal to schedule the jobs ordered by this strict upper bound.

► **Observation 3.2.** *For a given subset $S \subseteq \{1, \dots, n\}$ of jobs, there is a feasible schedule in which each job from S is early if and only if ordering the jobs according to increasing modified due dates*

$$d_j^* = \begin{cases} d_j & \text{if } j \in S \\ \bar{d}_j & \text{if } j \notin S \end{cases}$$

results in a schedule where all jobs are early.

Using Observation 3.2, we now basically know how scheduling job j as well as a given set $S \subseteq [j-1]$ of jobs early looks like: Before j , the early jobs from $[j-1]$ and all jobs whose deadline is smaller than d_j are scheduled. After j , all jobs with deadline larger than d_j (and which are not contained in S) are scheduled, according to increasing deadline. This implies the following criterion on when a schedule witnessing the value of $v_j[x_1, \dots, x_k]$ can also additionally schedule j early.

► **Lemma 3.3.** *Let $j \in [n]$ be a job with deadline $\bar{d}^{(i_1)}$ and $x_1, \dots, x_k \in \{0, 1, \dots, P\}$. Let $i_0 \in [k]$ be minimum such that $d_j \leq \bar{d}^{(i_0)}$. Let σ_{j-1} be a $(j-1; x_1, \dots, x_k)$ -compatible schedule where $S \subseteq [j-1]$ is the set of early jobs from $[j-1]$. Then there exists a $(j; x_1, \dots, x_{i_1-1}, x_{i_1} + p_j, x_{i_1+1}, \dots, x_k)$ -compatible schedule σ_j where $S \cup \{j\}$ is early if and only if*

1. $\sum_{\ell=1}^{i_0-1} P^{(\ell)} + \sum_{\ell=i_0}^k x_\ell + p_j \leq d_j$, and
2. for each $i \geq i_0$, we have $\sum_{\ell=1}^i P^{(\ell)} + \sum_{\ell=i+1}^k x_\ell \leq \bar{d}^{(i)}$.

Proof. (\Leftarrow): We begin with the reverse direction. Assume that there is a $(j; x_1, \dots, x_{i_1-1}, x_{i_1} + p_j, x_{i_1+1}, \dots, x_k)$ -compatible schedule σ_j where $S \cup \{j\}$ is early. By Observation 3.2, we may assume that σ schedules the jobs in non-decreasing order of modified due dates d_j^* (which are set according to the early set of jobs $S \cup \{j\}$).

We first show Item 1. Before job j , all jobs with deadline smaller than d_j and all early jobs with due date smaller than d_j are scheduled. The processing time of jobs with deadline smaller than d_j is precisely $\sum_{\ell=1}^{i_0-1} P^{(\ell)}$. All jobs from S have due date at most d_j (as we ordered the jobs according to increasing due date). Thus, the total processing time of all early jobs with due date at most d_j but deadline at least d_j equals $p_j + \sum_{\ell=i_0}^k x_\ell$. As j is early, we have $\sum_{\ell=1}^{i_0-1} P^{(\ell)} + p_j + \sum_{\ell=i_0}^k x_\ell \leq d_j$, *i.e.* Item 1 holds.

We continue by showing Item 2, so fix $i \geq i_0$. The last job with deadline $\bar{d}^{(i)}$ is processed after all jobs with deadline at most $\bar{d}^{(i)}$, as well as all early jobs with due date at most $\bar{d}^{(i)}$. The processing time of jobs with deadline at most $\bar{d}^{(i)}$ is $\sum_{\ell=1}^i P^{(\ell)}$. Each job $j' \in S \cup \{j\}$ satisfies $d_{j'} \leq d_j \leq \bar{d}^{(i_0)} \leq \bar{d}^{(i)}$. Thus, it follows that the processing time of the early jobs with due date at most $\bar{d}^{(i)}$ and deadline larger than $\bar{d}^{(i)}$ equals $\sum_{\ell=i+1}^k x_\ell$. Because σ is feasible, it follows that $\sum_{\ell=1}^i P^{(\ell)} + \sum_{\ell=i+1}^k x_\ell \leq \bar{d}^{(i)}$, *i.e.* Item 2 holds.

(\implies):) It remains to show the forward direction. We construct a schedule σ_j with the jobs from S as well as j being early as follows: We start with jobs from S and the jobs with deadline at most $\bar{d}^{(i_0-1)}$ (in the same relative order as they are in σ). Afterwards, we schedule job j , followed by the remaining jobs sorted according to increasing deadline.

First, we show that σ_j is a feasible schedule. Note that for each job from S as well as the jobs with deadline at most $\bar{d}^{(i_0-1)}$, their completion time can only decrease. Consequently, the feasibility of σ implies that these jobs are completed before their deadline. Next, consider a job j' with deadline $\bar{d}^{(i)}$ for some $i \geq i_0$. Before j' , all jobs with deadline $\bar{d}^{(\ell)}$ with $\ell < i$, potentially other jobs with deadline $\bar{d}^{(i)}$, and all early jobs from $1, \dots, j$ are scheduled. Thus, job j' is completed at time $\sum_{\ell=1}^i P^{(\ell)} + \sum_{\ell=i+1}^k x_\ell \leq \bar{d}^{(i)}$ where the inequality holds by Item 2. This implies that σ_j is a feasible schedule.

Next, we show that $S \cup \{j\}$ are early. All jobs from S are early as their completion time can only decrease. Job j is completed at time $\sum_{\ell=1}^{i_0-1} P^{(\ell)} + \sum_{\ell=i_0}^k x_\ell + p_j \leq d_j$ by Item 1. Therefore, job j is early. \blacktriangleleft

Note that the criterion from Lemma 3.3 is independent from the set of early jobs S , so indeed the dynamic program does not need to store the early jobs. We finally give the dynamic program and show its correctness in the theorem below.

► **Theorem 3.4.** $1|\bar{d}_j| \sum w_j U_j$ can be solved in $O(P^k \cdot k \cdot n)$ time, where k is the number of different deadlines.

Proof. We give the following dynamic program computing a solution. First, we order the jobs according to ascending due date (we assume that $d_1 \leq d_2 \leq \dots \leq d_n$). For each $j \in [n]$, the dynamic programming table τ_j contains an entry $\tau_j[x_1, \dots, x_k]$ for each $(x_1, \dots, x_k) \in \{0, \dots, P^{(1)}\} \times \dots \times \{0, \dots, P^{(k)}\}$. This entry shall equal $v_j[x_1, \dots, x_k]$; that is, the maximum number w such that there exists a $(j; x_1, \dots, x_k)$ -compatible schedule σ which schedules a subset $S \subseteq [j]$ of total weight w early; if no $(j; x_1, \dots, x_k)$ -compatible schedule exists, then $\tau_j[x_1, \dots, x_k] = -\infty$. The maximum of $\tau_n[x_1, \dots, x_k]$ over all $(x_1, \dots, x_k) \in \{0, \dots, P^{(1)}\} \times \dots \times \{0, \dots, P^{(k)}\}$ will then yield the value of an optimal schedule.

Initialization. We check whether there is a feasible schedule (this can be done e.g. using Observation 3.2). If so, then we set $\tau_0[0, \dots, 0] := 0$ while otherwise we set $\tau_0[0, \dots, 0] := -\infty$. For all $(x_1, \dots, x_k) \neq (0, \dots, 0)$, we set $\tau_0[x_1, \dots, x_k] := -\infty$.

Update. Let $j \in [n]$ and assume that job j has deadline $\bar{d}^{(i)}$. Fix $(x_1, \dots, x_k) \in \{0, \dots, P^{(1)}\} \times \dots \times \{0, \dots, P^{(k)}\}$. First, we check whether Items 1 and 2 of Lemma 3.3 are satisfied. If yes, then we set

$$\tau_j[x_1, \dots, x_k] := \max \left\{ \tau_{j-1}[x_1, \dots, x_k], w_j + \tau_{j-1}[x_1, \dots, x_{i-1}, x_i - p_j, x_{i+1}, \dots, x_k] \right\}.$$

Otherwise, we set $\tau_j[x_1, \dots, x_k] := \tau_{j-1}[x_1, \dots, x_k]$.

Optimal Solution. The minimum weighted number of tardy jobs in an optimal schedule is given by taking the minimum of $W - \tau_n[x_1, \dots, x_k]$ over all $(x_1, \dots, x_k) \in \{0, \dots, P^{(1)}\} \times \dots \times \{0, \dots, P^{(k)}\}$. An optimal schedule can be found by using backtracking to compute the set of early jobs and then applying Observation 3.2.

Correctness. Clearly, $\tau_0[x_1, \dots, x_k] = v_0[x_1, \dots, x_k]$ for every $(x_1, \dots, x_k) \in \{0, \dots, P^{(1)}\} \times \dots \times \{0, \dots, P^{(k)}\}$. Consider $\tau_j[x_1, \dots, x_k]$, where job j has deadline $\bar{d}^{(i)}$. First, we show $\tau_j[x_1, \dots, x_k] \geq v_j[x_1, \dots, x_k]$. Let σ be a schedule witnessing the value of $\tau_j[x_1, \dots, x_k]$ and S the corresponding set of early jobs from $[j]$. If $j \notin S$, then we have $v_j[x_1, \dots, x_k] = v_{j-1}[x_1, \dots, x_k] = \tau_{j-1}[x_1, \dots, x_k] \leq \tau_j[x_1, \dots, x_k]$. Otherwise, we have $v_j[x_1, \dots, x_k] = w_j + v_{j-1}[x_1, \dots, x_{i-1}, x_i - p_j, x_{i+1}, \dots, x_k] = w_j + \tau_{j-1}[x_1, \dots, x_{i-1}, x_i - p_j, x_{i+1}, \dots, x_k]$ and Lemma 3.3 implies that Items 1 and 2 of Lemma 3.3 are satisfied. Thus, $\tau_j[x_1, \dots, x_k] \geq w_j + \tau_{j-1}[x_1, \dots, x_k] = v_j[x_1, \dots, x_k]$.

We finish the proof of correctness by showing that $\tau_j[x_1, \dots, x_k] \leq v_j[x_1, \dots, x_k]$. If $\tau_j[x_1, \dots, x_k] = -\infty$, then there is nothing to show, so assume $\tau_j[x_1, \dots, x_k] > -\infty$. If we have $\tau_j[x_1, \dots, x_k] = \tau_{j-1}[x_1, \dots, x_k]$, then we have $\tau_j[x_1, \dots, x_k] = \tau_{j-1}[x_1, \dots, x_k] = v_{j-1}[x_1, \dots, x_k] \leq v_j[x_1, \dots, x_k]$. So assume that we set $\tau_j[x_1, \dots, x_k] = w_j + \tau_{j-1}[x_1, \dots, x_{i-1}, x_i - p_j, x_{i+1}, \dots, x_k]$. This implies that Items 1 and 2 from Lemma 3.3 are satisfied. Because $\tau_{j-1}[x_1, \dots, x_{i-1}, x_i - p_j, x_{i+1}, \dots, x_k] = v_{j-1}[x_1, \dots, x_{i-1}, x_i - p_j, x_{i+1}, \dots, x_k] \neq -\infty$, Lemma 3.3 implies that we can take the schedule corresponding to $v_{j-1}[x_1, \dots, x_{i-1}, x_i - p_j, x_{i+1}, \dots, x_k]$ and additionally schedule job j early. Thus, we have $v_j[x_1, \dots, x_k] \geq w_j + v_{j-1}[x_1, \dots, x_{i-1}, x_i - p_j, x_{i+1}, \dots, x_k] = \tau_j[x_1, \dots, x_k]$.

Running Time. The dynamic programming table contains $P^k \cdot n$ entries, each of which can be computed in $O(k)$ time. The claimed running time follows. ◀

We remark that the ETH-based lower bound from Theorem 3.1 implies that the exponent is optimal up to a factor of $O(\log k)$.

3.3 Few jobs with nontrivial deadlines

Having the intractability results from Theorem 3.1 in mind, we now consider a larger parameter, namely the number of jobs with a nontrivial deadline. Throughout this section, we denote the set of jobs having a nontrivial deadline by \bar{J} , and we set $k = |\bar{J}|$. We show that $1|\bar{d}_j| \sum w_j U_j$ can be solved in $O(2^k \cdot P \cdot n)$ time.

The basic idea is that for each job with a deadline, we can guess whether the job is early or tardy. Then, we adapt the due dates of these jobs according to Observation 3.2, resulting in modified due dates d_j^* . Finally, we significantly increase the weight of each job in \bar{J} and then call a known algorithm for $1|\sum w_j U_j$ with respect to the modified due dates d_j^* and weights w_j^* .

► **Theorem 3.5.** $1|\bar{d}_j| \sum w_j U_j$ can be solved in $O(2^k \cdot P \cdot n)$ time, where k is the number of jobs with nontrivial deadline.

Proof. Let \bar{J} be the set of jobs with a nontrivial deadline. First, we guess the subset of tardy jobs $\bar{U} \subseteq \bar{J}$ with nontrivial deadlines. We then set modified deadlines for all jobs according to Observation 3.2 and using the set $\bar{S} = \bar{J} \setminus \bar{U}$ as the set of early jobs. Thus, we have $d_j^* = d_j$ if $j \in \bar{S}$, and $d_j^* = \bar{d}_j$ if $j \in \bar{U}$. Moreover, we set $w_j^* = W + 1$ for all jobs $j \in \bar{J}$, and $w_j^* = w_j$ for all other jobs $j \in \{1, \dots, n\} \setminus \bar{J}$.

For these modified due dates d_j^* and weights w_j^* , we run the $O(Pn)$ -time algorithm for $1|\sum w_j U_j$ [16]. If the algorithm returns a schedule σ where the total weight of tardy jobs is at most W , then we store this schedule as a potential solution for the unmodified instance. Otherwise, we conclude that there is no feasible schedule with respect to our guess of the tardy jobs $\bar{U} \subseteq \bar{J}$. Our algorithm finally outputs the potential solution with the minimum weight of tardy jobs.

By Observation 3.2, the set of feasible schedules where each job from $\bar{J} \setminus \bar{U}$ is early is the set of schedules where each job from \bar{J} is early with respect to the modified due dates. This corresponds to schedules where the total weight of tardy jobs with respect to the modified weights is at most W . Thus, the potential solution corresponding to the current guess $\bar{U} \subseteq \bar{J}$ is a schedule minimizing the total weighted number of tardy jobs under the additional constraint that precisely the \bar{U} jobs of \bar{J} are tardy. It follows that for the guess $\bar{U} = U(\sigma^*)$ for some optimal schedule σ^* , the algorithm returns an optimal schedule. ◀

For the unweighted case (i.e., $w_j = 1$ for each $j \in J$), we can get an FPT-algorithm (even for binary encoded numbers) by following the same approach of guessing which of the jobs with nontrivial deadlines are tardy.

► **Theorem 3.6.** $1|\bar{d}_j| \sum U_j$ can be solved in $O(2^k \cdot n \log n)$ time where k is the number of jobs with a nontrivial deadline.

Proof. Let \bar{J} be the set of jobs with a nontrivial deadline. First, we guess the subset $\bar{S} \subseteq \bar{J}$ of early jobs with nontrivial deadlines. Observation 3.2 now reduces $1|\bar{d}_j| \sum U_j$ together with the guess \bar{S} to a variation of $1|\sum U_j$ where we are additionally given a set \bar{J} of jobs which have to be early. This problem is known to be solvable in $O(n \log n)$ time [21]. As there are 2^k possible guesses for \bar{S} , the running time follows. ◀

4 The $1|\bar{d}_j| \sum w_j C_j$ problem

We next examine the objective of minimizing the total weighted completion time. We remark that the unweighted variant of $1|\bar{d}_j| \sum w_j C_j$, the $1|\bar{d}_j| \sum C_j$ problem, is solvable in $O(n \log n)$ time [22]. Weak NP-hardness of $1|\bar{d}_j| \sum w_j C_j$ was shown by Lenstra *et al.* [17], using only a single job with a nontrivial deadline. Therefore, unless $P=NP$, there is no XP-algorithm for $1|\bar{d}_j| \sum w_j C_j$ with binary encoding parameterized by the number of jobs with nontrivial deadlines.

Below we strengthen Lenstra's reduction, and show that $1|\bar{d}_j| \sum w_j C_j$ is $W[1]$ -hard when parameterized by the number of jobs with nontrivial deadlines even if all numbers are encoded in unary. Note that this implies also $W[1]$ -hardness for the problem when the number of different deadlines is taken as a parameter. To compliment our hardness result, we present algorithm running in $O(k \cdot n \cdot P^{2k-2})$ time, where k is the number of different deadlines.

4.1 Hardness

We adapt the reduction from Lenstra *et al.* [17] to show strong NP-hardness and $W[1]$ -hardness parameterized by the number of jobs with nontrivial deadline (also for unary encoding). We will reduce from BIN PACKING restricted to instances where each bin must be filled exactly.

EXACT BIN PACKING

Input: t items with sizes a_1, \dots, a_t , a bin size B , and the number K of bins.

Question: Is there an assignment of the items to the bins such that each bin contains items of total size exactly B ?

EXACT BIN PACKING is strongly NP-hard [6] and $W[1]$ -hard parameterized by the number of bins, even if all numbers are encoded in unary [13]. Let $(a_1, \dots, a_t; B; K)$ be an instance of EXACT BIN PACKING. We create an instance of $1|\bar{d}_j| \sum w_j C_j$ with $t + K - 1$ jobs as follows:

24:10 Single Machine Scheduling with Few Deadlines

- For $j \in [t]$, job j has processing time and weight $p_j = a_j = w_j$ and trivial deadline $\bar{d}_j = K \cdot B + K - 1$.
- For $\ell \in [K - 1]$, job $t + \ell$ has processing time $p_{t+\ell} = 1$, weight $w_{t+\ell} = 0$, and deadline $\bar{d}_{t+\ell} = \ell \cdot (B + 1)$.
- We set the bound on the total weighted completion time to

$$b := \sum_{i_1, i_2 \in [t]} a_{i_1} a_{i_2} + \sum_{\ell=1}^{K-1} (K - \ell) \cdot B.$$

Note that the total processing time of all jobs is $K \cdot B + K - 1 = \bar{d}_j$ for all $j \in [t]$, so only jobs $t + 1, \dots, t + K - 1$ have a nontrivial deadline. The idea behind the reduction is as follows: Because $w_j = p_j$ for all $j \in [t]$, the relative order of jobs j_1 and j_2 does not matter: if j_1 is before j_2 , then this contributes $w_{j_2} \cdot p_{j_1} = a_{j_1} \cdot a_{j_2}$ to the total completion time (as this increases the completion time of j_2 by p_{j_1}), while otherwise this contributes $w_{j_1} \cdot p_{j_2} = a_{j_1} \cdot a_{j_2}$ to the total processing time. Consequently, the total completion time of a schedule only depends on the completion times of the jobs $t + \ell$ for $\ell \in [K - 1]$. As their weight is 0 but they have nonzero processing time, they should be scheduled as late as possible (since they only increase the processing times of other jobs), i.e., directly before their deadline. In any such schedule, the processing time of the jobs between $t + \ell$ and $t + \ell + 1$ equals $\bar{d}_{t+\ell+1} - \bar{d}_{t+\ell} - p_{t+\ell+1} = B$, implying a solution to the EXACT BIN PACKING instance.

► **Lemma 4.1.** *The t items can be packed into K bins iff the $t + K + 1$ jobs constructed have a feasible schedule with at most b total weighted completion time.*

Proof. (\implies): Assume that there is a solution to the EXACT BIN PACKING instance and that the set of items contained in ℓ -th bin is A_ℓ . We denote the set of jobs corresponding to the items from A_ℓ by J_ℓ , i.e. $J_\ell := \{j : a_j \in A_\ell\}$. We construct a schedule σ as follows: We start with the jobs from J_1 in arbitrary order, followed by job $t + 1$. Afterwards, we schedule the jobs from J_2 followed by $t + 2$. We continue scheduling J_ℓ followed by job $t + \ell$ until we schedule job $t + K - 1$. Finally, we schedule all jobs from J_K in arbitrary order. This finishes the construction of σ . It remains to show that σ is feasible and has total weighted completion time at most b .

We start with the feasibility of σ . The only jobs with nontrivial deadline are $t + \ell$ for $\ell \in [K - 1]$. Job $t + \ell$ is completed after the jobs from $A_1 \cup A_2 \cup \dots \cup A_\ell$ and jobs $t + 1, \dots, t + \ell$. The processing time of jobs $t + 1, \dots, t + \ell$ is ℓ . Since $\sum_{a \in A_i} a = B$, the processing time of jobs from $A_1 \cup \dots \cup A_\ell$ is $\ell \cdot B$. Thus, $t + \ell$ is completed at time $\ell + \ell \cdot B = \bar{d}_{t+\ell}$. Consequently, σ is feasible.

We continue by showing that the weighted completion time of σ is at most b . In order to analyze the total completion time, we split it into three parts: First, jobs $t_1, \dots, t + K - 1$ have weight 0 and thus their weighted completion time is 0. Second, we consider the weighted completion time caused by some job j being scheduled after some job $t + \ell$ for $j \in [t]$ and $\ell \in [K - 1]$. The jobs scheduled after $t + \ell$ are $J_{\ell+1} \cup J_{\ell+2} \cup \dots \cup J_K$ as well as $t + \ell + 1, \dots, t + K - 1$. Thus, the weight of all jobs scheduled after $t + \ell$ is

$$\sum_{i=\ell+1}^K \sum_{j \in J_i} w_j = \sum_{i=\ell+1}^K \sum_{a \in A_i} a = \sum_{i=\ell+1}^K B = (K - \ell) \cdot B.$$

Finally, we consider the weighted completion time caused by some job j_1 being scheduled before some job j_2 for $j_1, j_2 \in [t]$. For each $j_1, j_2 \in [t]$, job j_1 is either scheduled before j_2 ,

job j_2 is scheduled before j_1 , or $j_1 = j_2$. In all cases, this contributes $a_{j_1} \cdot a_{j_2} = w_{j_1} \cdot p_{j_2} = w_{j_2} \cdot p_{j_1}$ to the total weighted completion time. Summing all three parts together, the total weighted completion time is

$$\sum_{\ell=1}^{K-1} (K - \ell) \cdot B + \sum_{j_1, j_2 \in [t]} a_{j_1} a_{j_2} = b.$$

(\Leftarrow): In the converse direction, assume that there is a feasible schedule σ with total weighted completion time at most b . As argued in the forward direction, for each $j_1, j_2 \in [t]$ there is a contribution of $a_{j_1} \cdot a_{j_2}$ to the total weighted completion time. We may assume without loss of generality that σ schedules job $t + \ell$ before job $t + \ell'$ for all $\ell < \ell'$ as jobs $t + \ell$ and $t + \ell'$ have the same weight and processing time, but $\bar{d}_{t+\ell} < \bar{d}_{t+\ell'}$.

Job $t + \ell$ contributes 1 to the completion time of all jobs scheduled after $t + \ell$. Let $J^{>t+\ell}$ be the set of jobs from $[t]$ which are scheduled after $t + \ell$, and let $J^{t+\ell}$ be the jobs from $[t]$ which are scheduled before $t + \ell$. The total processing time of $J^{t+\ell}$ cannot exceed $\ell \cdot B$ as $\bar{d}_{t+\ell} = \ell \cdot (B + 1)$, and jobs $t + 1, \dots, t + \ell - 1$ are scheduled before $t + \ell$. Consequently, we have

$$\sum_{j \in J^{>t+\ell}} w_j = \sum_{j \in J^{>t+\ell}} a_j = \sum_{j \in [t]} a_j - \sum_{j \in [t] \setminus J^{>t+\ell}} a_j = K \cdot B - \sum_{j \in J^{<t+\ell}} p_j \geq K \cdot B - \ell \cdot B$$

where equality holds if and only if job $t + \ell$ is completed precisely at time $\bar{d}_{t+\ell}$. Because $b = \sum_{i_1, i_2 \in [t]} a_{i_1} a_{i_2} + \sum_{i=\ell}^{K-1} (K - i) \cdot B$, this implies that each job $t + \ell$ is completed precisely at time $\bar{d}_{t+\ell}$. Thus, between jobs $t + \ell$ and $t + \ell + 1$, jobs of processing time exactly B are scheduled.

Consequently, we construct a solution to the BIN PACKING instance as follows: If job j is scheduled after $t + \ell - 1$ but before $t + \ell$ for some $\ell \in \{2, 3, \dots, K - 1\}$, then we assign a_j to the ℓ -th bin. If j is scheduled before $t + 1$, then we assign a_j to the first bin. If j is scheduled after $t + K - 1$, then we assign a_j to the K -th bin. Thus, the t items can all be packed into K bins of size B each. \blacktriangleleft

Recall that EXACT BIN PACKING is strongly NP-hard, W[1]-hard when parameterized by K , and does not admit an $f(K) \cdot n^{o(K/\log K)}$ -time algorithm assuming ETH even if all numbers are encoded in unary [13]. Thus, as the construction above constructs $K - 1$ jobs with nontrivial deadlines, we get the following Theorem directly from Lemma 4.1:

► Theorem 4.2. *Let k denote the number of jobs with nontrivial deadline in a $1|\bar{d}_j|\sum w_j C_j$ instance. Then the $1|\bar{d}_j|\sum w_j C_j$ problem*

1. *is strongly NP-hard,*
2. *is W[1]-hard parameterized by k even when all numbers are encoded in unary, and*
3. *admits no $f(k) \cdot P^{o(k/\log k)}$ -time algorithm assuming ETH.*

4.2 Constant number of deadlines

We complement the W[1]-hardness in case of unary encoding shown in the previous subsection by presenting a pseudo-polynomial time algorithm for a constant number of different deadlines. Similar to Section 3.2, the first step of the dynamic program consists of sorting the jobs in a favorable manner. Further, the dynamic program contains a table τ_j for each $j \in [n]$, and these tables contain entries $\tau_j[x_1, \dots, x_k]$ where x_i encodes the total processing time of the jobs scheduled between \bar{d}_{i-1} and \bar{d}_i (where $\bar{d}_0 := 0$). However, in contrast to Section 3.2,

24:12 Single Machine Scheduling with Few Deadlines

a schedule corresponding to $\tau_j[x_1, \dots, x_k]$ now only schedules jobs $1, \dots, j$ instead of all jobs, and it may be that the schedule cannot be extended to a feasible schedule for all jobs. Furthermore, the x_i now measure the processing times of jobs scheduled between the $(i-1)$ th and i th deadline. Another difference is that we “guess” certain characteristics of an optimal schedule and adapt the instance to these characteristics before starting the dynamic program.

Our dynamic programming processes the jobs from $1, \dots, n$, and computes a table τ_j for each $j \in [n]$. This table has an entry $\tau_j[x_1, \dots, x_k]$ which stores the minimum weighted completion time of a schedule of jobs $\{1, \dots, j\}$ such that

1. the schedule is feasible for $1, \dots, j$, *i.e.* each job from $1, \dots, j$ is completed not after its deadline, and
2. the total processing time of jobs being completed between $\bar{d}^{(i-1)}$ and $\bar{d}^{(i)}$ equals x_i , for every $i \in [k]$ (where we set $\bar{d}^{(0)} := 0$).

We call a schedule fulfilling these two conditions $(j; x_1, \dots, x_k)$ -obeying ($(j; x_1, \dots, x_k)$ -obeying is the counterpart to $(j; x_1, \dots, x_k)$ -compatible from Section 3.2 in the sense that it determines whether a schedule fulfills the conditions for $\tau_j[x_1, \dots, x_k]$).

To compute the above dynamic program, we need to be able to, given an oracle which tells us the set S^i of jobs being completed between $\bar{d}^{(i-1)}$ and $\bar{d}^{(i)}$ for each $i \in [k]$, find an optimal schedule. This can be done easily: As the relative order of the jobs from S^i does not influence the completion times of jobs from S^ℓ for $i \neq \ell$, we can schedule the jobs from S^i independently from the rest. In this subinstance consisting of the jobs from S^i , the deadlines of jobs from S^i are trivial, and so we can use the structure of an optimal schedule for $1 \parallel \sum w_j C_j$: An optimal schedule for $1 \parallel \sum w_j C_j$ schedules the jobs according to WSPT [22], that is, according to non-decreasing ratio p_j/w_j . Thus, we want to schedule each set S^i according to WSPT. We formalize this argument in the following lemma:

► **Lemma 4.3.** *Let σ be an optimal schedule for a $1 \parallel \sum w_j C_j$ instance. For $i \in [k]$, let S^i be the set of jobs which are completed after $\bar{d}^{(i-1)}$ but not later than $\bar{d}^{(i)}$ (using $\bar{d}^{(0)} := 0$). Then the jobs from S^i are ordered according to WSPT in σ .*

Proof. Assume towards a contradiction that they are not ordered according to WSPT. Then there is a job $j_1 \in S^i$ directly followed (in σ) by a job $j_2 \in S^i$ with $p_{j_1}/w_{j_1} > p_{j_2}/w_{j_2}$. We claim that the schedule $\sigma_{2,1}$ arising from σ by exchanging jobs j_1 and j_2 is a feasible schedule with smaller total weighted completion time, contradicting the optimality of σ . First, we show the feasibility of $\sigma_{2,1}$. All jobs but j_1 and j_2 are completed at the same time and thus are completed by their deadline by the feasibility of σ . For jobs j_1 and j_2 , we have $\min\{\bar{d}_{j_1}, \bar{d}_{j_2}\} \geq \bar{d}^{(i)}$ as σ was a feasible schedule. Further, as j_1 and j_2 are contained in S^i , both jobs are completed not later than $\bar{d}^{(i)}$ in both σ and $\sigma_{2,1}$. Thus, $\sigma_{2,1}$ is feasible. It remains to consider the total weighted completion time of $\sigma_{2,1}$. All jobs except for j_1 and j_2 are completed at the same time and therefore have the same contribution to the weighted completion time. Let t be the starting time of job j_1 in σ . The weighted completion time of j_1 and j_2 in σ is $C := w_{j_1} \cdot (t + p_{j_1}) + w_{j_2} \cdot (t + p_{j_1} + p_{j_2})$, while the weighted completion time of j_1 and j_2 in $\sigma_{2,1}$ is $C_{2,1} := w_{j_2} \cdot (t + p_{j_2}) + w_{j_1} \cdot (t + p_{j_2} + p_{j_1})$. Thus, we have $C - C_{2,1} = w_{j_2} p_{j_1} - w_{j_1} p_{j_2} > 0$ using $w_{j_1}/p_{j_1} < w_{j_2}/p_{j_2}$ for the inequality. Therefore, $\sigma_{2,1}$ has a smaller total weighted completion time than σ , contradicting the optimality of σ . ◀

Using Lemma 4.3, the basic idea of the dynamic program is the following: We order the jobs according to WSPT, *i.e.* $p_j/w_j \leq p_{j+1}/w_{j+1}$ for all $j \in \{1, \dots, n-1\}$. For each $i \in [k-1]$, we guess when the first job which is completed after $\bar{d}^{(i)}$ starts (there are only P^{k-1} possible guesses). For the sake of simplicity, assume that for each $i \in [k-1]$ the guess

is $\bar{d}^{(i)}$, *i.e.* for each $i \in [k-1]$ there is one job starting at time $\bar{d}^{(i)}$. Recall that the dynamic program $\tau_{j-1}[x_1, \dots, x_k]$ contains a $(j-1; x_1, \dots, x_k)$ -obeying schedule of $1, \dots, j-1$ of minimum weighted processing time. There are now up to k possibilities how j is scheduled, namely between $\bar{d}^{(i-1)}$ and $\bar{d}^{(i)}$ for every $i \in [k]$ such that $\bar{d}_j \leq \bar{d}^{(i)}$. Assuming that j is completed between $\bar{d}^{(i-1)}$ and $\bar{d}^{(i)}$, we know how job j is scheduled: As we ordered the jobs according to WSPT, Lemma 4.3 implies that j will be the last job from $1, \dots, j$ which is completed $\bar{d}^{(i-1)}$ and $\bar{d}^{(i)}$. Thus, it is completed at time $\bar{d}^{(i-1)} + x_i + p_j$. Consequently, this results in a $(j; x_1, \dots, x_{i-1}, x_i + p_j, x_{i+1}, \dots, x_k)$ -obeying schedule with weighted completion time $\tau_{j-1}[x_1, \dots, x_k] + w_j \cdot (\bar{d}^{(i-1)} + x_i + p_j)$ if $x_i + p_j \leq \bar{d}^{(i)} - \bar{d}^{(i-1)}$. Therefore, we set $\tau_j[x_1, \dots, x_k] := \min_{i \in [k]: \bar{d}_j \leq \bar{d}^{(i)}} \tau_{j-1}[x_1, \dots, x_{i-1}, x_i - p_j, x_{i+1}, \dots, x_k] + w_j \cdot (\bar{d}^{(i-1)} + x_i)$.

► **Theorem 4.4.** $1|\bar{d}_j| \sum w_j C_j$ can be solved in $O(P^{2k-2} \cdot k \cdot n)$ time, where k is the number of different deadlines.

Proof. First, we guess for each $i \in [k-1]$ the time t_i when the first job which is completed after $\bar{d}^{(i)}$ starts. Afterwards, we reduce the deadline $\bar{d}^{(i)}$ to t_i . This ensures that there will be no job starting before $\bar{d}^{(i)}$ and being completed after $\bar{d}^{(i)}$.

We order the jobs according to WSPT. The dynamic program contains a table τ_j for each $j \in \{0, \dots, n\}$. Each such table contains an entry $\tau_j[x_1, \dots, x_n]$ for $x_i \in \{0, 1, \dots, \bar{d}^{(i)} - \bar{d}^{(i-1)}\}$ (where $\bar{d}^{(0)} := 0$). This entry contains the minimum weighted completion time of a $(j; x_1, \dots, x_k)$ -obeying schedule. We now formally describe how these values can be computed.

Initialization. We set $\tau_0[0, \dots, 0] := 0$ and $\tau_0[x_1, \dots, x_k] := \infty$ otherwise.

Update. Let $j \in [n]$ and assume that job j has deadline $\bar{d}^{(i)}$. Then

$$\tau_j[x_1, \dots, x_k] := \min_{\ell \in [i]} \left\{ \tau_{j-1}[x_1, \dots, x_{\ell-1}, x_\ell - p_j, x_{\ell+1}, \dots, x_k] + w_j \cdot (\bar{d}^{(\ell-1)} + x_\ell) \right\}$$

Optimal Solution. The optimal solution value is $\tau_n[\bar{d}^{(1)}, \bar{d}^{(2)} - \bar{d}^{(1)}, \bar{d}^{(3)} - \bar{d}^{(2)}, \bar{d}^{(4)} - \bar{d}^{(3)}, \dots, \bar{d}^{(k)} - \bar{d}^{(k-1)}]$. An optimal schedule can be found using backtracking.

Correctness. Clearly, $\tau_0[x_1, \dots, x_k]$ equals the minimum weighted processing time of a $(0; x_1, \dots, x_k)$ -feasible schedule. Recall that we assume $\bar{d}^{(k)} = P$, so we have $\sum_{i=1}^k (\bar{d}^{(i)} - \bar{d}^{(i-1)}) = P$. Because an optimal schedule has no idle time, the total processing time of jobs between $\bar{d}^{(\ell-1)}$ and $\bar{d}^{(\ell)}$ is precisely $\bar{d}^{(\ell)} - \bar{d}^{(\ell-1)}$. Thus, $\tau_n[\bar{d}^{(1)}, \bar{d}^{(2)} - \bar{d}^{(1)}, \bar{d}^{(3)} - \bar{d}^{(2)}, \bar{d}^{(4)} - \bar{d}^{(3)}, \dots, \bar{d}^{(k)} - \bar{d}^{(k-1)}]$ contains the value of an optimal solution (assuming correctness of the update step).

It remains to show that the update step is correct. Let σ be a $(j; x_1, \dots, x_k)$ -obeying schedule of minimum weighted processing time. Because no job starts before $\bar{d}^{(i)}$ and is completed after $\bar{d}^{(i)}$ as we adapted the deadlines to our initial guess, each job which is completed between $\bar{d}^{(\ell-1)}$ and $\bar{d}^{(\ell)}$ also starts between $\bar{d}^{(\ell-1)}$ and $\bar{d}^{(\ell)}$. Fix $\ell \in [k]$ such that job j is completed in σ between $\bar{d}^{(\ell-1)}$ and $\bar{d}^{(\ell)}$. Note that $\ell \leq i$ as σ is feasible. As we process the jobs in WSPT order, we may assume by Lemma 4.3 that j is the last job from $[j]$ which starts between $\bar{d}^{(\ell-1)}$ and $\bar{d}^{(\ell)}$. Then removing job j results in a schedule

of jobs $1, \dots, j-1$ where the processing time of jobs starting between $\bar{d}^{(\ell'-1)}$ and $\bar{d}^{(\ell')}$ is exactly $x_{\ell'}$ for $\ell' \neq \ell$ and $x_{\ell} - p_j$ for $\ell' = \ell$. Scheduling j after all other jobs starting between $\bar{d}^{(\ell-1)}$ and $\bar{d}^{(\ell)}$ then results in a completion time of $\bar{d}^{(\ell)} + x_{\ell}$ for job j . The total weighted completion time of jobs $1, \dots, j-1$ is at least $\tau_{j-1}[x_1, \dots, x_{\ell-1}, x_{\ell} - p_j, x_{\ell+1}, \dots, x_k]$. Consequently, a $(j; x_1, \dots, x_k)$ -obeying schedule has weighted completion time at least $w_j(\bar{d}^{(\ell)} + x_{\ell}) + \tau_{j-1}[x_1, \dots, x_{\ell-1}, x_{\ell} - p_j, x_{\ell+1}, \dots, x_k] \geq \tau_j[x_1, \dots, x_k]$.

It remains to show that an $(j; x_1, \dots, x_k)$ -obeying schedule has weighted completion time at most $\tau_j[x_1, \dots, x_k]$. For each $\ell \leq i$, combining the schedule from $\tau_j[x_1, \dots, x_{\ell-1}, x_{\ell} - p_j, x_{\ell+1}, x_k]$ with job j scheduled at time $\bar{d}^{(\ell-1)} + x_{\ell}$ results in a schedule σ_{ℓ} with total weighted completion time $\tau_j[x_1, \dots, x_{\ell-1}, x_{\ell} - p_j, x_{\ell+1}, x_k] + w_j \cdot (\bar{d}^{(\ell-1)} + x_{\ell})$. Schedule σ_{ℓ} completes j before its deadline as $\ell \leq i$ and $x_{\ell} \leq \bar{d}^{(\ell)} - \bar{d}^{(\ell-1)}$. Thus, σ_{ℓ} is $(j; x_1, \dots, x_k)$ -obeying. Therefore, there is a $(j; x_1, \dots, x_k)$ -obeying schedule with weighted completion time at most $\tau_j[x_1, \dots, x_k]$.

Running Time. There are $O(P^{k-1})$ many different guesses (for each $i \in [k-1]$, we guess one time t_i). For each guess, the dynamic programming table as described above contains $O(n \cdot P^k)$ many entries, each of which can be computed in $O(k)$ time. However, note that we only need to consider entries $\tau_j[x_1, \dots, x_k]$ with $\sum_{\ell=1}^k x_{\ell} = \sum_{j'=1}^j p_{j'}$. Thus, for each combination of j and x_1, \dots, x_{k-1} , there is only one value of x_k which we need to consider. This implies that it suffices to compute $O(n \cdot P^{k-1})$ many entries of the dynamic programming table. Thus, the total running time is $O(k \cdot n \cdot P^{2k-2})$. ◀

We remark that the ETH-based lower bound from Theorem 4.2 implies that the exponent is optimal up to a factor of $O(\log k)$.

5 Conclusion

We initiated the study of the parameterized complexity of scheduling problems with deadlines. While we arrived at a complete FPT-vs.-W[1]-hardness-vs.-XP-classification of $1|\bar{d}_j|\sum w_j U_j$ and $1|\bar{d}_j|\sum w_j C_j$ with respect to the number of different deadlines and the number of jobs with nontrivial deadline, there is still ample room for future work: For example, one might study other parameterizations not focusing on the deadlines such as the number of different processing times. Another direction would be to extend our study to further problems such as $1|\bar{d}_j|\sum T_j$. For this problem, one likely gets similar in Theorem 3.1 by modifying the reduction from [3] similar to Theorem 3.1, but it is unclear on whether a pseudopolynomial-time algorithm for a constant number of different deadlines exists. Lastly, we left open the approximability of both $1|\bar{d}_j|\sum w_j C_j$ and $1|\bar{d}_j|\sum w_j U_j$ (note that $1|\sum w_j U_j$ admits an FPTAS [20]). Due to the strong NP-hardness of $1|\bar{d}_j|\sum w_j C_j$ and $1|\bar{d}_j|\sum w_j U_j$, both problems do not admit an FPTAS unless P=NP. However, the existence of a PTAS or a constant-factor approximation algorithm is open.

References

- 1 Philippe Baptiste, Federico Della Croce, Andrea Grosso, and Vincent T'kindt. Sequencing a single machine with due dates and deadlines: an ILP-based approach to solve very large instances. *Journal of Scheduling*, 13(1):39–47, 2010.
- 2 Peter Brucker. *Scheduling algorithms (4. ed.)*. Springer, 2004.
- 3 Rubing Chen and Jinjiang Yuan. Unary NP-hardness of single-machine scheduling to minimize the total tardiness with deadlines. *Journal of Scheduling*, 22(5):595–601, 2019.

- 4 Rubing Chen and Jinjiang Yuan. Unary NP-hardness of preemptive scheduling to minimize total completion time with release times and deadlines. *Discrete Applied Mathematics*, 304:45–54, 2021.
- 5 Rubing Chen, Jinjiang Yuan, C.T. Ng, and T.C.E. Cheng. Single-machine scheduling with deadlines to minimize the total weighted late work. *Naval Research Logistics*, 66(7):582–595, 2019.
- 6 Michael R. Garey and David S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM J. Comput.*, 4(4):397–411, 1975.
- 7 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 8 Ronald L. Graham, Eugene L. Lawler, Jan K. Lenstra, and Alexander H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979.
- 9 A. M. A. Hariri and Chris N. Potts. Single machine scheduling with deadlines to minimize the weighted number of tardy jobs. *Management Science*, 40(12):1712–1719, 1994.
- 10 Cheng He, Hao Lin, Yixun Lin, and Junmei Dou. Minimizing total completion time for preemptive scheduling with release dates and deadline constraints. *Foundations of Computing and Decision Sciences*, 39(1):17–26, 2014.
- 11 Cheng He, Yixun Lin, and Jinjiang Yuan. A note on the single machine scheduling to minimize the number of tardy jobs with deadlines. *European Journal of Operational Research*, 201(3):966–970, 2010.
- 12 Yumei Huo, Joseph Y.-T. Leung, and Hairong Zhao. Bi-criteria scheduling problems: Number of tardy jobs and maximum weighted tardiness. *European Journal of Operational Research*, 177(1):116–134, 2007.
- 13 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.
- 14 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center*, The IBM Research Symposia Series, pages 85–103. Plenum Press, 1972.
- 15 Eugene L. Lawler. Scheduling a single machine to minimize the number of late jobs. Technical Report UCB/CSD-83-139, EECS Department, University of California, Berkeley, 1983.
- 16 Eugene L. Lawler and J. Michael Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16(1):77–84, 1969.
- 17 Jan K. Lenstra, Alexander H.G. Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. In *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343–362. Elsevier, 1977.
- 18 J. Michael Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1):102–109, 1968.
- 19 Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems (5. ed.)*. Springer, 2016.
- 20 Sartaj Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23(1):116–127, 1976.
- 21 Jeffrey B. Sidney. An extension of Moore’s due date algorithm. In *Symposium on the Theory of Scheduling and Its Applications*, pages 393–398. Springer Berlin Heidelberg, 1973.
- 22 Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.
- 23 Long Wan, Jinjiang Yuan, and Zhichao Geng. A note on the preemptive scheduling to minimize total completion time with release time and deadline constraints. *Journal of Scheduling*, 18(3):315–323, 2015.
- 24 Jinjiang Yuan. Unary NP-hardness of minimizing the number of tardy jobs with deadlines. *Journal of Scheduling*, 20(2):211–218, 2017.

Twin-Width of Graphs with Tree-Structured Decompositions

Irene Heinrich  

Technische Universität Darmstadt, Germany

Simon Raßmann  

Technische Universität Darmstadt, Germany

Abstract

The twin-width of a graph measures its distance to co-graphs and generalizes classical width concepts such as tree-width or rank-width. Since its introduction in 2020 [13, 12], a mass of new results has appeared relating twin width to group theory, model theory, combinatorial optimization, and structural graph theory.

We take a detailed look at the interplay between the twin-width of a graph and the twin-width of its components under tree-structured decompositions: We prove that the twin-width of a graph is at most twice its strong tree-width, contrasting nicely with the result of [7, 6], which states that twin-width can be exponential in tree-width. Further, we employ the fundamental concept from structural graph theory of decomposing a graph into highly connected components, in order to obtain optimal linear bounds on the twin-width of a graph given the widths of its biconnected components. For triconnected components we obtain a linear upper bound if we add red edges to the components indicating the splits which led to the components. Extending this approach to quasi-4-connectivity, we obtain a quadratic upper bound. Finally, we investigate how the adhesion of a tree decomposition influences the twin-width of the decomposed graph.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph algorithms; Mathematics of computing → Paths and connectivity problems

Keywords and phrases twin-width, quasi-4 connected components, strong tree-width

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.25

Related Version *Full Version*: <https://arxiv.org/abs/2308.14677>

Funding *Irene Heinrich*: The research leading to these results has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (EngageS: grant agreement No. 820148).

1 Introduction

Twin-width is a new graph parameter introduced in [13, 12]. Since its introduction it has gained considerable attention. The *twin-width*¹ of a graph G , denoted by $\text{tw}(G)$, is the minimum width over all contraction sequences of G and a *contraction sequence* of G is roughly defined as follows: we start with a discrete partition of the vertex set of G into n singletons where n is the order of G . Now we perform a sequence of $n - 1$ merges, where in each step of the sequence precisely two parts are merged causing the partition to become coarser, until eventually, we end up with just one part – the vertex set of G . Two parts of a partition of $V(G)$ are *homogeneously connected* if either all or none of the possible cross

¹ We refer to the preliminaries of this paper (subsections *graphs and trigraphs* as well as *twin-width*) for an equivalent definition of twin-width which is based on merging vertices instead of vertex subsets.



© Irene Heinrich and Simon Raßmann;

licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 25; pp. 25:1–25:17

Leibniz International Proceedings in Informatics



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

edges between the two parts are present in G . The red degree of a part is the number of other parts to which it is not homogeneously connected. Finally, the width of a contraction sequence is the maximum red degree amongst all parts of partitions arising when performing the sequence.

In [13, 12] the authors show that twin-width generalizes other width parameters such as rank-width, and, hence also clique-width and tree-width. Furthermore, given a graph H , the class of H -minor free graphs has bounded twin-width and FO-model checking is FPT on classes of bounded twin-width, see [13, 12]. Many combinatorial problems which are NP-hard in general allow for improved algorithms if the twin-width of the input graph is bounded from above and the graph is given together with a width-minimal contraction sequence [9, 8].

Motivation. To decompose a graph into smaller components and estimate a certain parameter of the original graph from the parameters of its components is an indispensable approach of structural graph theory, which serves in taming branch-and-bound trees as well as for theoretical considerations because it allows for stronger assumptions (i.e., high connectivity) on the considered graphs. There are various ways to decompose a graph, e.g., bi-, tri-, or quasi-4-connected components, tree decompositions of small adhesion (the maximum cardinality of the intersection of two adjacent bags), modular decomposition, or decomposition into the factors of a graph product.

So far there is no detailed analysis of the relation between the twin-width of a graph and the twin-width of its biconnected, triconnected, or quasi-4-connected components. The only result towards (k -)connected components is the basic observation that the twin-width of a graph is obviously the maximum over the twin-width over its (1-connected) components. While there already exists a strong analysis of the interplay of tree-width and twin-width (cf. [19, 20]), it is still open how twin-width behaves with respect to the adhesion of a given tree decomposition, which can be significantly smaller than the tree-width of a graph (as an example, consider a graph whose biconnected components are large cliques – the adhesion is 1 whereas the tree-width is the maximum clique size). Further, there exist many variants of tree-width, for example, strong tree-width [23, 16] for which the interplay with twin-width has not yet been discussed in the literature.

Our results. We prove the following bound on the twin-width of a graph:

► **Theorem 1.** *If G is a graph of strong tree-width k , then*

$$\text{tw}(G) \leq \frac{3}{2}k + 1 + \frac{1}{2}(\sqrt{k + \ln k} + \sqrt{k} + 2 \ln k).$$

This is a strong contrast to the result of [7, 6] that twin-width can be exponential in tree-width. We further provide a class of graphs which asymptotically satisfies that the twin-width equals the strong tree-width. Further, we investigate how to bound the twin-width of a graph in terms of the twin-width of its highly connected components starting with biconnected components.

► **Theorem 2.** *If G is a graph with biconnected components C_1, C_2, \dots, C_ℓ , then*

$$\max_{i \in [\ell]} \text{tw}(C_i) \leq \text{tw}(G) \leq \max_{i \in [\ell]} \text{tw}(C_i) + 2.$$

Next, we consider decompositions into triconnected components:

► **Theorem 3.** *Let C_1, C_2, \dots, C_ℓ be the triconnected components of a biconnected graph G . For $i \in [\ell]$ we construct a trigraph \overline{C}_i from C_i as follows: all virtual edges² of C_i are colored red and all other edges remain black. If C_i contains parallel edges, then we remove all but one of the parallel edges such that the remaining edge is red whenever one of the parallel edges was red. Then*

$$\text{tw}(G) \leq \max \left(8 \max_{i \in [\ell]} \text{tw}(\overline{C}_i) + 6, 18 \right).$$

Similarly clean decompositions into k -connected graphs with $k > 3$ cannot exist [14, 15]; but we move on one more step and consider the twin-width of a graph with respect to its quasi-4 connected components, introduced by [14, 15].

► **Theorem 4.** *Let G be a triconnected graph with quasi-4-connected components C_1, C_2, \dots, C_ℓ .*

1. *For $i \in [\ell]$ we construct a trigraph \widehat{C}_i by adding for every 3-separator S in C_i along which G was split a vertex v_S which we connect via red edges to all vertices in S . Then*

$$\text{tw}(G) \leq \max \left(8 \max_{i \in [\ell]} \text{tw}(\widehat{C}_i) + 14, 70 \right).$$

2. *For $i \in [\ell]$, we construct a trigraph \overline{C}_i by coloring all edges in 3-separators in C_i along which G was split red. Then*

$$\text{tw}(G) \leq \max \left(4 \max_{i \in [\ell]} (\text{tw}(\overline{C}_i)^2 + \text{tw}(\overline{C}_i)) + 14, 70 \right).$$

For the general case of tree decompositions of bounded adhesion, we get the following:

► **Theorem 5.** *For every $k \in \mathbb{N}$ there exist explicit constants D_k and D'_k such that for every graph G with a tree decomposition of adhesion k and parts P_1, P_2, \dots, P_ℓ , the following statements are satisfied:*

1. *For each P_i , we construct a trigraph \widehat{P}_i by adding for each adhesion set S in P_i a new vertex v_S which we connect via red edges to all vertices in S . Then*

$$\text{tw}(G) \leq 2^k \max_{i \in [\ell]} \text{tw}(\widehat{P}_i) + D_k.$$

2. *Assume $k \geq 3$. For each P_i , we construct the torso \overline{P}_i by completing every adhesion set in P_i to a red clique. Then*

$$\text{tw}(G) \leq \frac{2^k}{(k-1)!} \max_{i \in [\ell]} \text{tw}(\overline{P}_i)^{k-1} + D'_k.$$

Finally, we refine the result of [19, 20], where the authors bound the twin-width of a graph given its tree-width.

► **Theorem 6.** *Let G be a graph with a tree decomposition of width w and adhesion k . Then*

$$\text{tw}(G) \leq 3 \cdot 2^{k-1} + \max(w - k - 2, 0).$$

² That is, the pairs of vertices along which G was split to obtain the triconnected components.

Bounding the red degree of decomposition trees. The underlying structure of all the decompositions that we consider in this paper is a tree. We generalize the optimal contraction sequence (cf. [3, 2]) for trees which works as follows: choose a root for the tree. If possible, choose two sibling leaves and contract them (which implies a red edge from the new vertex to its parent). Whenever a parent is joined to two of its leaf-children via red edges, these two children are merged. This ensures that the red degree of any parent throughout the whole sequence never exceeds 2. If there are no sibling leaves, then choose a leaf of highest distance to the root and contract it with its parent. This yields a red edge between the new merged vertex and the former grandparent. Repeat this until we end up with a singleton. We preserve this idea in our proofs to ensure that at no point in time three distinct bag-siblings contribute to the red degree of the vertices in their parent bag.

Further related work. A standard reference on tree-width is [5]. For the basics on graph connectivity and decomposition we refer text books on graph theory such as [24]. The twin-width of a graph given the twin-width of its modular decomposition factors (and in particular, also the twin-width given the width of the factors of a lexicographical product) already has been investigated in [11, 10]. In contrast to the linear-time solvable tree-width decision problem [4] (for a fixed k : is the tree-width of the input graph at most k ?), deciding whether the twin-width of a graph is at most 4 is already NP-complete [3, 2]. The twin-width of a graph in terms of its biconnected components has already been considered in [21], where the author obtains a slightly weaker upper bound than Theorem 2.

Organization of the paper. We provide the preliminaries in Section 2. Our results on strong tree-width can be found in Section 3. In Section 4 we prove new bounds on the twin-width of a graph given the twin-widths of its highly connected components, and, we generalize our approach to graphs which allow for a tree-decomposition of small adhesion. Due to space limitations, some of the proofs are omitted. We refer to [17] for the full version of this paper.

2 Preliminaries

For a natural number n , we denote by $[n]$ the n -element set $\{1, \dots, n\}$. For a set A , we write $\mathcal{P}(A)$ for the power set of A . For a natural number $k \leq |A|$, we write $\binom{A}{k}$ for the set of k -element subsets of A .

Graphs and trigraphs. All graphs in this paper are finite, undirected and contain no loops. For a graph G , we denote its vertex set by $V(G)$ and its edge set by $E(G)$. We write $|G| := |V(G)|$ for the *order* of G .

A *trigraph* is an undirected, edge-colored graph G with disjoint sets $E(G)$ of *black edges* and $R(G)$ of *red edges*. We can interpret every graph as a trigraph by setting $R(G) = \emptyset$. For a vertex subset A of a trigraph G , we denote by $G[A]$ the *subgraph induced on A* and by $G - A$ the subgraph induced on $V(G) \setminus A$. For a vertex $v \in V(G)$, we also write $G - v$ instead of $G - \{v\}$. If G is a graph, then the *degree* of a vertex $v \in V(G)$ is denoted by $d_G(v)$ (or $d(v)$ if G is clear from context). For trigraphs, we write $\text{red-deg}_G(v)$ for the *red degree* of v , i.e., the degree of v in the graph $(V(G), R(G))$. We write $\Delta(G)$ or $\Delta_{\text{red}}(G)$ for the maximum (red) degree of a (tri-)graph G .

A *multigraph* is a graph where we allow multiple edges between each pair of vertices.

Twin-width. Let G be a trigraph and $x, y \in V(G)$ two distinct, not necessarily adjacent vertices of G . We *contract x and y* by merging the two vertices to a common vertex z , leaving all edges not incident to x or y unchanged, connecting z via a black edge to all common black neighbors of x and y , and via a red edge to all red neighbors of x or y and to all vertices which are connected to precisely one of x and y . We denote the resulting trigraph by G/xy . A *partial contraction sequence* of G is a sequence of trigraphs $(G_i)_{i \in [k]}$ where $G_1 = G$ and G_{i+1} can be obtained from G_i by contracting two distinct vertices $x_i, y_i \in V(G_i)$. By abuse of notation, we also call the sequence $(x_i y_i)_{i < |G|}$ of contraction pairs a partial contraction sequence. The width of a partial contraction sequence is the maximal red degree of all graphs G_1, \dots, G_k . If the width of a sequence is at most d , we call it a *d -contraction sequence*. A (*complete*) *contraction sequence* is a partial contraction sequence whose final trigraph is the singleton graph on one vertex. The minimum width over all complete contraction sequences of G is called the *twin-width* of G and is denoted by $\text{tw}(G)$. We often identify a vertex $v \in V(G)$ with the vertices in the graphs G_i that v gets contracted to and sets of vertices with the sets of vertices they get contracted to.

Twin-width has many nice structural properties. For example, it is monotone with respect to induced subgraphs: for every induced subgraph $H \subseteq G$ it holds that $\text{tw}(H) \leq \text{tw}(G)$. Moreover, the twin-width of a disconnected graph is just the maximum twin-width of its connected components.

Tree decompositions and tree-width. Let G be a graph. A *tree decomposition* of G is a pair $\mathcal{T} = (T, \{B_i : i \in V(T)\})$ consisting of a tree T and a family $(B_i)_{i \in V(T)}$ of subsets of $V(G)$, called *bags* satisfying the following conditions

1. every vertex of G is contained in some bag,
2. for every vertex $v \in V(G)$, the set of tree vertices $i \in V(T)$ such that $v \in B_i$ forms a subtree of T ,
3. for every edge $e \in E(G)$, there exists some bag which contains both endpoints of e .

The subgraphs $G[B_i]$ are called the *parts* of the tree decomposition. The width of a tree-decomposition is $\max_{i \in V(T)} |B_i| - 1$ and the minimum width over all tree decompositions of G is the *tree-width* of G and is denoted by $\text{tw}(G)$.

For an edge $ij \in E(T)$, the sets $B_i \cap B_j$ are the *adhesion sets* or *separators* of \mathcal{T} and the maximal size of an adhesion set is the *adhesion* of \mathcal{T} . The graphs obtained from a part $G[B_i]$ by completing all adhesion sets $B_i \cap B_j$ to cliques is called the *torso* of $G[B_i]$.

Strong tree-width. Strong tree-width, which is also called tree-partition width, is a graph parameter independently introduced by [23] and [16]. A *strong tree decomposition* of a graph G is a tuple $(T, \{B_i : i \in V(T)\})$ where T is a tree and $\{B_i : i \in V(T)\}$ is a set of pairwise disjoint subsets of $V(G)$, one for each node of T such that

1. $V(G) = \bigcup_{i \in V(T)} B_i$ and
2. for every edge uv of G there either exists a node $i \in V(T)$ such that $\{u, v\} \subseteq B_i$ or there exist two adjacent nodes i and j in T with $u \in B_i$ and $v \in B_j$.

The sets B_i are called *bags* and $\max_{i \in V(T)} |B_i|$ is the *width* of the decomposition. The minimum width over all strong tree decompositions of G is the *strong tree-width* $\text{stw}(G)$ of G .

The strong tree-width of a graph is bounded in its tree-width via $\text{tw}(G) \leq 2\text{stw}(G) - 1$, see [25]. In the other direction, there is no bound: the strong-tree width of a graph is unbounded in its tree-width [25]. However, it holds that $\text{stw}(G) \in O(\Delta(G) \cdot \text{tw}(G))$, see [25]. Thus, for graphs of bounded degree, the two width notions are linearly equivalent.

► **Remark 7.** In general, the strong tree-width is unbounded in the twin-width of a graph. For example, consider a complete graph on $2n$ vertices. A width-minimal strong tree decomposition of this graph has two bags, each containing n vertices. However the twin-width of a complete graph is 0.

Highly connected components. A *cut vertex* of a graph G is a vertex $v \in V(G)$ such that $G - v$ contains more connected components than G . A maximal connected subgraph of G that has no cut vertex is a *biconnected component* of G . The *block-cut-tree* of G is a bipartite graph where one part is the set of biconnected components of G and the other part is the set of cut vertices of G and a biconnected component is joined to a cut vertex precisely if the vertex is contained in the component. This graph is a forest, and even a tree if G is connected [24]. If we choose a biconnected component as a root of this tree, we can restrict this tree structure to a tree structure on the biconnected components of G . Thus, the decomposition of a graph into biconnected components can also be phrased as follows:

► **Theorem 8** (see [24]). *For every connected graph G , there exists a tree decomposition \mathcal{T} of G such that \mathcal{T} has adhesion at most 1, and every part is either 2-connected or a complete graph of order 2. Moreover, the set of bags of this tree decomposition is isomorphism-invariant.*

Similarly, by splitting a graph at certain separators of size at most 2, we obtain the following:

► **Theorem 9** ([18]). *For every 2-connected graph, there exists a tree decomposition \mathcal{T} of G such that \mathcal{T} has adhesion at most 2, and the torso of every bag is either 3-connected, a cycle, or a complete graph of order 2. Moreover, the set of bags of this tree decomposition is isomorphism-invariant.*

The *triconnected components* of G are multigraphs constructed from the torsos of this tree decomposition. In this work, these multigraphs are not important and we also call the torsos themselves *triconnected components*.

A similarly clean decomposition into 4-connected components arranged in a tree-like fashion does not exist [14, 15]. This motivated Grohe to introduce the notion of quasi-4-connectivity [14, 15]: A graph G is called *quasi-4-connected* if it is 3-connected and all 3-separators split off at most a single vertex. That is, for every separator S of size 3, the graph $G - S$ splits into exactly two connected components, at least one of which consists of a single vertex. The prime example of quasi-4-connected graphs which are not 4-connected are hexagonal grids. For quasi-4-connectivity, we once again get a tree-like decomposition into components:

► **Theorem 10** ([14, 15]). *For every 3-connected graph G , there exists a tree decomposition \mathcal{T} of G such that \mathcal{T} has adhesion at most 3, and the torso of every bag is either quasi-4-connected or of size at most 4.*

The torsos of this tree decomposition are called *quasi-4-connected components* of G .

3 Twin-width of graphs of bounded strong tree-width

► **Theorem 1.** *If G is a graph of strong tree-width k , then*

$$\text{tww}(G) \leq \frac{3}{2}k + 1 + \frac{1}{2}(\sqrt{k + \ln k} + \sqrt{k} + 2 \ln k).$$

Proof. For a graph H and a vertex subset $U \subseteq V(H)$ a partial contraction sequence s of H is a U -contraction sequence if only vertices of U are involved in the contractions in s and s is of length $|U|$, that is, performing all contractions of s yields a partition of $V(H)$ where U forms one part and the rest of the parts are singletons. We denote the minimum width over all U -contraction sequences of H by $\text{tw}_U(H)$.

Let $\mathcal{T} = (T, \{B_i : i \in V(T)\})$ be a strong tree decomposition of G of width k . Fix $r \in V(T)$ and consider T to be a rooted tree with root r from now on. If a bag B_i contains only one vertex v , then we set $v_i := v$. We label all nodes i of T with $|B_i| = 1$ as *merged*. All other nodes of T are labeled as *unmerged*. A node p of T is a *leaf-parent* if all of its children are leaves. If B_i is a bag of \mathcal{T} , then *contracting* B_i means to apply a width-minimal B_i -contraction sequence and then relabel i as *merged*. After a contraction of two vertices u and v to a new vertex x we *update* the strong tree decomposition \mathcal{T} , that is, if u and v were contained in the same bag, then we simply replace u and v by x . If, otherwise, u and v are

■ **Algorithm 1** $\text{CONTRACT}(G, \mathcal{T} = (T, \{B_i, i \in V(T)\}))$.

```

1 while  $|V(T)| \geq 2$  do
2   Choose a leaf  $\ell$  which maximizes the distance to  $r$ 
3   if the parent  $p$  of  $\ell$  has two merged children  $\ell_1, \ell_2$ , then
4     contract  $v_{\ell_1}$  with  $v_{\ell_2}$ ,
5     update  $\mathcal{T}$ 
6   if  $\ell$  is the only child of its parent  $p$  and  $\ell$  is merged, then
7     contract  $B_p$  and denote the resulting vertex  $b_p$ ,
8     contract  $b_p$  with  $v_\ell$ ,
9     update  $\mathcal{T}$ 
10  if amongst  $\ell$  and its siblings there is an unmerged leaf  $\ell'$  and at most one merged
    leaf, then
11    contract  $B_{\ell'}$ ,
12    update  $\mathcal{T}$ 
13 Apply a width-minimal contraction sequence to the remaining graph.

```

contained in adjacent bags, then we remove u and v from its bags and insert x to the bag which is closer to the root. If this causes an empty bag, we remove the bag as well as the corresponding tree-vertex. Observe that updating preserves the strong tree-width. We claim that the algorithm CONTRACT merges G into a single vertex via a contraction sequence of the required width.

First, we check that that the algorithm terminates. Observe that the root r is not part of any of the contractions in the while-loop. In particular, as long as the loop is executed, there exists at least one leaf-parent. In every iteration of the loop at least one of the if-conditions is satisfied and hence, $|V(G)|$ shrinks with every iteration, which proves that the algorithm terminates with a singleton graph, that is, it provides a contraction sequence.

It remains to bound the width of the sequence. For $a \in \mathbb{N}$ we set $f(a) := (a + \sqrt{a + \ln a} + \sqrt{a} + 2 \ln a)/2$. We will exploit the result of [19, 20] that an a -vertex graph has twin-width at most $f(a)$.

Let $(G_i)_{i \leq |G|}$ be the contraction obtained by the algorithm. Fix $i \in [|G|]$ and $v \in G_i$ and let $\mathcal{T}_i = (T_i, \mathcal{B}_i)$ be the strong tree decomposition corresponding to G_i and B_j the bag containing v in \mathcal{T} .

If j is neither a leaf, nor in a leaf-parent, nor the parent of a leaf-parent in T_i , then $\text{red-deg}(v) = 0$.

Assume that j is a leaf of T_i , then all red edges incident to v are either internal edges of B_j or joining v with a vertex of B_p where p is the parent of j in T_i . Since $\text{stw}(G) \leq k$ there are at most k red edges of the latter form. Internal red edges of a bag may only arise during a the contraction of this leaf-bag in Line 10. Since the corresponding partial contraction sequence is chosen to be width-minimal and by the bound of [19, 20] we obtain that $\text{red-deg}_{G_i}(v) \leq k + f(k)$.

Now assume that j is a leaf-parent in T_i . If the bag B_j of T_i was already contained in \mathcal{T} , then there are no internal red edges in B_j and the only red edges incident to v are incident to the vertices of precisely one leaf-bag, or, to the vertices of precisely two leaf-bags one of which is merged. In each of the two cases, the red degree of v in G_i is bounded by $k + 1$. Otherwise B_j is obtained during the contraction in Line 6. In this case, j has precisely one child ℓ in T_i and ℓ is merged. Hence, j has at most $k + f(k) + 1$ red neighbors.

Finally, assume that j is neither a leaf nor a leaf-parent but parent of a leaf-parent in T_i . Let j_1, \dots, j_h be the children of j in T_i . Observe that there are at most two children of j , say, j_1 and j_2 such that v is joined to vertices of the corresponding bags and there are no internal red edges in B_j . The only red edges incident to v are arising during the contraction of B_{j_1} or B_{j_2} in Line 6. Since first, one of the two children is contracted to one vertex before any contraction in the other bag happens, the red degree of v is bounded by $k + 1$. ◀

► **Lemma 11.** *There exists a family of graphs $(H_n)_{n \in \mathbb{N}}$ such that $\lim_{n \rightarrow \infty} \frac{\text{tw}(H_n)}{\text{stw}(H_n)} \geq 1$.*

Proof. For each $n \in \mathbb{N}$ let H_n be the n -th Paley graph. Fix $n \in \mathbb{N}$. It is known that $\text{tw}(H_n) = \frac{|V(H_n)|-1}{2}$, see [19, 20]. By distributing the vertices of H_n to two bags, one of cardinality $\frac{|V(H_n)|+1}{2}$, the other one of cardinality $\frac{|V(H_n)|-1}{2}$, we obtain a strong tree decomposition of H_n of width $\frac{|V(H_n)|+1}{2}$. ◀

4 Twin-width of graphs with small separators

4.1 Biconnected components

We start our investigation of graphs of small adhesion by proving a bound on the twin-width of graphs in terms of the twin-width of their biconnected components. This proof contains many of the ideas we will generalize later to deal with tri- and quasi-4-connected components as well as general graphs with a tree decomposition of bounded adhesion.

The main obstacle to constructing contraction sequences of a graph from contraction sequences of its biconnected components is that naively contracting one component might increase the red degree of the incident cut vertices in the neighboring components arbitrarily. Thus, we need to find contraction sequences of the biconnected components not involving the incident cut vertices.

Let G be a trigraph and \mathcal{P} be a partition of G . Denote by G/\mathcal{P} the trigraph obtained from G by contracting each part of \mathcal{P} into a single vertex. For a vertex $v \in V(G)$ we denote by $\mathcal{P}(v)$ the part of \mathcal{P} that contains v . If $\mathcal{P}(v) \neq \{v\}$, then we obtain a refined partition \mathcal{P}_v by replacing $\mathcal{P}(v)$ in \mathcal{P} by the two parts $\mathcal{P}(v)$ and $\{v\}$. Otherwise, we set $\mathcal{P}_v = \mathcal{P}$. Since G/\mathcal{P} can be obtained from G/\mathcal{P}_v by at most one contraction, and one contraction of a trigraph reduces the maximum red degree by at most 1 we have

$$\Delta_{\text{red}}(G/\mathcal{P}_v) \leq \Delta_{\text{red}}(G/\mathcal{P}) + 1. \quad (1)$$

► **Lemma 12.** *For every trigraph G and every vertex $v \in V(G)$,*

$$\text{tww}_{V(G-v)}(G) \leq \text{tww}(G) + 1.$$

Proof. Let $(\mathcal{P}^{(i)})_{i \in [|G|]}$ be a sequence of partitions corresponding to a width-minimal contraction sequence of G . Further, let j be the maximal index with $\{v\} \in \mathcal{P}^{(j)}$. Then $(\mathcal{P}^{(i)})_{i \in [j]}$ is a partial $\text{tww}(G)$ -contraction sequence which does not involve v , and by (1) the sequence $(\mathcal{P}_v^{(i)})_{i \in [|G|] \setminus [j+1]}$ is a partial $(\text{tww}(G) + 1)$ -contraction sequence which contracts the resulting trigraph until v and one further vertex remain. Combining these two sequences yields the claim. ◀

► **Theorem 2.** *If G is a graph with biconnected components C_1, C_2, \dots, C_ℓ , then*

$$\max_{i \in [\ell]} \text{tww}(C_i) \leq \text{tww}(G) \leq \max_{i \in [\ell]} \text{tww}(C_i) + 2.$$

Proof. The lower bound follows from the fact that all biconnected components are induced subgraphs of G together with the monotonicity of twin-width.

For the upper bound we may assume that G is connected since the twin-width of a disconnected graph is the maximum twin-width of its connected components [13, 12]. Consider the block-cut-tree of G , i.e., the tree T whose vertex set is the union of the biconnected components of G and the cut vertices of G , where every cut vertex joined to precisely those biconnected components containing it. In particular, the biconnected components and the cut vertices form a bipartition of T .

We choose a cut vertex r as a root of T . For every biconnected components $C \in V(T)$, we let v_C be the parent of C in T .

To make our argument simpler, let \widehat{G} be the graph obtained from G by joining a new vertex r_v to every vertex $v \in V(G)$ via a red edge. Similarly, for a biconnected component C , we let \widehat{C} be the graph obtained from C by attaching a new vertex r_v to every vertex v of C . For each cut vertex c , we let \widehat{G}_c be the graph induced by \widehat{G} on the union of all blocks in the subtree T_c of T rooted at c together with all vertices r_v adjacent to these blocks.

We show that $\text{tww}(\widehat{G}) \leq \max_{i \in [\ell]} \text{tww}(C_i) + 2$. The claim then follows since G is an induced subgraph of \widehat{G} .

▷ **Claim 13.** For every biconnected component C of G ,

$$\text{tww}_{V(\widehat{C}-v_C)}(\widehat{C}) \leq \text{tww}(C) + 2.$$

Proof of the Claim. By applying Lemma 12 to C and v_C , we find a $V(C - v_C)$ -contraction sequence S of C of width at most $\text{tww}(C) + 1$. We show how this contraction sequence can be adapted to also contract the vertices r_v for all cut vertices v incident to C . Indeed, before every contraction vw of S , we insert the contraction of r_v and r_w . This keeps the invariant that we never contract a vertex from C with a vertex r_v , and further, every vertex of C is incident to at most one vertex r_v (or a contraction of those vertices). Moreover, the red degree among the vertices r_v also stays bounded by 2. The entire partial contraction sequence constructed so far thus has width at most $\text{tww}(C) + 2$.

After applying this sequence, we end up with at most four vertices: v_C, r_{v_C} , the contraction of $C - v_C$ and the contraction of all vertices r_v for vertices $v \neq v_C$. As r_{v_C} is only connected to v_C and the contraction of all other vertices r_v is not connected to v_C , these four vertices form a path of length four. Thus, the contraction sequence can be completed with trigraphs of width at most 2. ◀

Now, consider again the whole graph \widehat{G} and choose a leaf block C of T . We can apply the partial contraction sequence given by the previous claim to \widehat{C} in \widehat{G} . Because we never contract v_C with any other vertex, this does not create red edges anywhere besides inside \widehat{C} . Thus, it is still a partial $(\text{tw}(C) + 2)$ -contraction sequence of \widehat{G} . Moreover, the resulting trigraph is isomorphic to $\widehat{G} - V(\widehat{C} - v_C)$, i.e., the graph obtained from \widehat{G} by just removing the biconnected component C (but leaving the cut vertex v_C). By iterating this, we can remove all biconnected components one after the other using width at most $\max_{i \in [\ell]} \text{tw}(C_i) + 2$. Finally, we end up with just two vertices: The root cut vertex r , together with its red neighbor r_r , which we can simply contract. \blacktriangleleft

Note that the bounds in Theorem 2 are sharp even on the class of trees: the biconnected components of a tree are just its edges which have twin-width 0. As there are trees both of twin-width 0 and of twin-width 2, both the upper and the lower bound can be obtained.

► **Corollary 14.** *Let \mathcal{C} be a graph class closed under taking biconnected components. Then \mathcal{C} has bounded twin-width if and only if the subclass of 2-connected graphs in \mathcal{C} has.*

Moreover, Theorem 2 also reduces the algorithmic problem of computing or approximating the twin-width of a graph to within some factor to the corresponding problem on biconnected graphs.

4.2 Apices and contractions respecting subsets

To deal with adhesion sets of size at least 2, it no longer suffices to find contraction sequences of the parts that just don't contract vertices in the adhesion sets. Indeed, as those vertices can appear parts corresponding to a subtree of unbounded depth, this could create an unbounded number of red edges incident to vertices in adhesion sets. Instead, we want contraction sequences that create no red edges incident to any vertices of adhesion sets.

For a trigraph G and a set of vertices $A \subseteq V(G)$ of red degree 0, we say that a partial sequence of d -contractions $G = G_0, G_1, \dots, G_\ell$ respects A if $G_i[A] = G[A]$ and $\text{red-deg}_{G_i}(a) = 0$ for all $i \leq \ell$ and $a \in A$. Thus, for every contraction xy in the sequence, we have $x, y \notin A$ and $N(x) \cap A = N(y) \cap A$, which implies that the vertices in A are not incident to any red edges all along the sequence.

A complete d -contraction sequence respecting A is a sequence of d -contractions that respects A of maximal length, i.e., one whose resulting trigraph G_ℓ does not allow a further contraction respecting A . This is equivalent to no two vertices in $V(G_\ell) \setminus A$ having the same neighborhood in A . In particular, a complete contraction sequence respecting A leaves at most $2^{|A|}$ vertices besides A .

We write $\text{tw}(G, A)$ for the minimal d such that there exists a complete d -contraction sequence respecting A . For a single vertex $v \in V(G)$, we also write $\text{tw}(G, v)$ for $\text{tw}(G, \{v\})$. Note that $\text{tw}(G) = \text{tw}(G, \emptyset)$.

It was proven in [13, 12, Theorem 2] that adding a single apex to a graph of twin-width d raises the twin-width to at most $2d + 2$. The proof given there readily works in our setting without any modifications.

► **Theorem 15.** *Let G be a trigraph, $v \in V(G)$ a vertex not incident to any red edges and $A \subseteq V(G) \setminus \{v\}$ a set of vertices. Then*

$$\text{tw}(G, A \cup \{v\}) \leq 2 \text{tw}(G - v, A) + 2.$$

► **Corollary 16.** *Let G be a trigraph and $A \subseteq V(G)$ a subset of vertices with $\text{red-deg}(a) = 0$ for all vertices $a \in A$. Then*

$$\text{tw}(G, A) \leq 2^{|A|} \text{tw}(G) + 2^{|A|+1} - 2.$$

4.3 Tree decompositions of small adhesion

We are now ready to generalize the linear bound on the twin-width of a graph in terms of its biconnected components to allow for larger separators of bounded size. This is most easily expressed in terms of tree decompositions of bounded adhesion.

In all of the following two sections, let G be a graph, $\mathcal{T} = ((T, r), \{B_t : t \in V(T)\})$ a rooted tree decomposition with adhesion $k \geq 1$.

For a vertex $t \in V(T)$, we write $P_t := G[B_t]$ for the part associated to t . For a vertex $t \in V(T)$ with parent $s \in V(T)$ we write $S_t := B_t \cap B_s$ and call S_t the *parent separator* of P_t or a *child separator* of P_s . Moreover, we set $S_r := \emptyset$ to be the *root separator*. For a tree vertex $t \in V(T)$, we write T_t for the subtree of T with root t , $G_t := G[\bigcup_{s \in V(T_t)} B_s]$ for the corresponding subgraph and \mathcal{T}_t for the corresponding tree decomposition of G_t .

We can assume w.l.o.g. that every two vertices $s, t \in V(T)$ with $S_s = S_t$ are siblings. Indeed, if they are not, let s' be a highest vertex in the tree with parent separator S_s and construct another tree decomposition by attaching all vertices t with $S_t = S_s$ directly to the parent of s' instead of their old parent. By repeating this procedure if necessary, we obtain the required property.

For a vertex $t \in T$ with children c_1, \dots, c_ℓ , we set $N_{c_i} := \{N(v) \cap S_{c_i} : v \in V(G_{c_i}) \setminus S_{c_i}\}$ to be the set of (possibly empty) neighborhoods that vertices in $G_{c_i} - S_{c_i}$ have in the separator S_{c_i} (and thus in P_t). We now define a trigraph \tilde{P}_t with vertex set

$$V(\tilde{P}_t) := V(P_t) \dot{\cup} \{s_M^{c_i} : i \in [\ell], M \in N_{c_i}\}.$$

We will often abuse notation and also denote the set $\{s_M^{c_i} : M \in N_{c_i}\}$ by N_{c_i} .

We define the edge set of \tilde{P}_t such that

1. $\tilde{P}_t[V(P_t)] = P_t$,
 2. $\tilde{P}_t[N_{c_i}]$ is a red clique for every i ,
 3. $s_M^{c_i}$ is connected via black edges to all vertices in M ,
- and there are no further red or black edges. Note that in \tilde{P}_t , there are no red edges incident to any vertices in P_t and thus in particular not to any vertices in S_t . A drawing of the gadget attached to S_{c_i} in \tilde{P}_t in comparison with the simpler gadgets we will reduce to later can be seen in Figure 1.

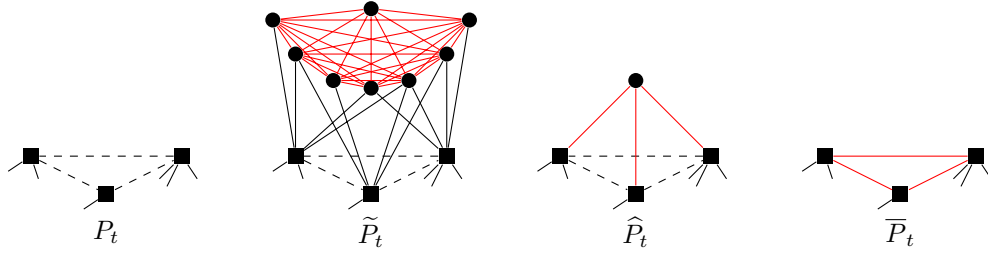
► **Lemma 17.** *Let G and \mathcal{T} be as above. For every $t \in V(T)$, it holds that*

$$\text{tw}(G_t, S_t) \leq \max_{s \in V(T_t)} \text{tw}(\tilde{P}_s, S_s).$$

In particular, $\text{tw}(G) \leq \max_{s \in V(T)} \text{tw}(\tilde{P}_s, S_s) \leq 2^k \max_{s \in V(T)} \text{tw}(\tilde{P}_s) + 2^{k+1} - 2$.

Proof sketch. We proceed inductively, starting by contracting the leaf bags and then moving up the tree. The graphs \tilde{P}_t are defined precisely so that contracting all child bags of some bag yields \tilde{P}_t . ◀

If \mathcal{T} has bounded width, we can proceed as in [19, 20, Lemma 3.1] to bound the twin-width of the graphs \tilde{P}_t and thus the twin-width of G :



■ **Figure 1** A separator S_t on three (square) vertices together with the three versions of gadgets we attach to it. Dashed edges represent either edges or non-edges. In \tilde{P}_t , we add a red clique consisting of one vertex for every neighborhood of vertices in $G_{t'} - S_{t'}$ in $S_{t'}$. In \hat{P}_t , we only add a single vertex with red edges to all vertices in $S_{t'}$. In \bar{P}_t , we add no new vertices but complete all child separators to red cliques.

► **Lemma 18.** *Let G and \mathcal{T} be as above and additionally assume that \mathcal{T} has width at most w . For every $t \in V(T)$, it holds that*

$$\text{tww}(\tilde{P}_t, S_t) \leq 3 \cdot 2^{k-1} + \max(w - k - 2, 0).$$

Proof. We first note that the red degree of \tilde{P} itself is bounded by $2^k - 1$.

Now, let c_1, \dots, c_ℓ be the (possibly empty) list of children of t in T . We first find a contraction sequence of $\bigcup_{i=1}^{\ell} N_{c_i}$ respecting S_t . For this, we argue by induction that $\bigcup_{i=1}^{j-1} N_{c_i}$ can be contracted while preserving the required width. This claim is trivial for $j = 1$. Hence assume we have already contracted $\bigcup_{i=1}^{j-1} N_{c_i}$ to a set B_{j-1} of size at most $2^{|S_P|}$. The vertices of B_{j-1} may be connected via red edges to vertices in B_{j-1} itself and in $P_t \setminus S_t$. Thus, the red degree of vertices in B_{j-1} is bounded by

$$|B_{j-1}| - 1 + |P_t| - |S_t| \leq 2^{|S_t|} + |P_t| - |S_t| - 1 \leq 2^k + w - k - 1,$$

while the red degree of vertices in $P_t \setminus S_t$ is bounded by $|B_j| \leq 2^{|S_t|} \leq 2^k$.

Now, we first apply a maximal contraction sequence of N_{c_j} respecting S_t resulting in a quotient \bar{N}_{c_j} . Because of our assumption on the tree decomposition \mathcal{T} , we know that $S_t \neq S_{c_j}$ for all $j \in [\ell]$. In particular, this implies that $|S_{c_j} \cap S_t| < k$. and thus $|\bar{N}_{c_j}| \leq 2^{k-1}$. During this contraction sequence, there can appear red edges between the contracted vertices of N_{c_j} and vertices in $P_t \setminus S_t$. The vertices of N_{c_j} thus have red degree bounded by $|N_{c_j}| - 1 + |P_t| - |S_t| \leq 2^k + w - k - 1$. Every red neighbor of vertices in $P_t \setminus S_t$ in a quotient of N_{c_j} must be the contraction of at least two vertices of N_{c_j} . Thus, the red degree of these vertices is bounded by

$$|B_{j-1}| + |N_{c_j}|/2 \leq 2^k + 2^{k-1} = 3 \cdot 2^{k-1}.$$

Next, we contract vertices from B_{j-1} and \bar{N}_{c_j} which have equal neighborhoods in S_t . As our bounds already allow every vertex in $P_t \setminus S_t$ to be connected via red edges to all of $B_{j-1} \cup \bar{N}_{c_j}$, it suffices to argue that this keeps the red degree of vertices in $B_{j-1} \cup \bar{N}_{c_j}$ within our bounds. But this set has size at most $3 \cdot 2^{k-1}$. Hence, after one contraction, the red degree is bounded by

$$|B_{j-1}| + |\bar{N}_{c_j}| - 2 + |P_t| - |S_t| \leq 3 \cdot 2^{k-1} + w - k - 2.$$

We have now successfully contracted N_{c_j} into B_{j-1} while keeping the red degree bounded by

$$\max(2^k + w - k - 1, 2^k, 3 \cdot 2^{k-1}, 3 \cdot 2^{k-1} + w - k - 2) = 3 \cdot 2^{k-1} + \max(w - k - 2, 0).$$

By repeating this procedure for all $j \in [\ell]$, we find a contraction sequence of $\bigcup_{i=1}^{\ell} N_{c_i}$ respecting S_t within this width. The resulting graph thus consists of S_t , the vertices of $P_t \setminus S_t$ and the vertices from B_{ℓ} . In total, these are at most $2^{|S_t|} + |P_t| - |S_t| \leq 2^k + w - k$ vertices besides those in S_t . These can further be contracted while keeping the red degree bounded by

$$2^k + w - k - 1 \leq 3 \cdot 2^{k-1} + w - k - 2.$$

In total, our contraction sequence thus has width at most $3 \cdot 2^{k-1} + \max(w - k - 2, 0)$, proving the claim. \blacktriangleleft

By combining Lemma 18 with Lemma 17, we obtain a general bound on the twin-width of graphs admitting a tree decomposition of bounded width and adhesion:

► **Theorem 6.** *Let G be a graph with a tree decomposition of width w and adhesion k . Then*

$$\text{tww}(G) \leq 3 \cdot 2^{k-1} + \max(w - k - 2, 0).$$

This upper bound sharpens the bound given in [19, 20] by making explicit the dependence on the adhesion of the tree decomposition. Our bound shows that, while the twin-width in general can be exponential in the tree-width [7, 6], the exponential dependence comes from the adhesion of the tree decomposition and not from the width itself.

Moreover, our bound is asymptotically sharp. As already mentioned, it is known that there are graphs whose twin-width is exponential in the adhesion of some tree decomposition [7, 6]. By adding into some bag a Paley graph whose twin-width is linear in its size [1], we also achieve asymptotic sharpness in the linear width term.

4.4 Simplifying the parts

Before we apply this general lemma to the special case of the tree of bi-, tri- or quasi-4-connected components, we show that we can simplify the gadgets attached in the graphs \tilde{P} to all separators while raising the twin-width by at most a constant factor.

In a first step, we replace the sets N_{c_i} from the definition of the parts \tilde{P}_t by a single common red neighbor for every separator. For every vertex $t \in V(T)$ with children c_1, \dots, c_{ℓ} , we define the trigraph \hat{P}_t as follows: we set $\mathcal{S}(t) := \{S_{c_i} : i \in [\ell], S_{c_i} \not\subseteq S_{c_j} \text{ for all } j \in [\ell]\}$ to be the set of subset-maximal child separators of P_t . Now, we take a collection of fresh vertices $V_S := \{v_S : S \in \mathcal{S}(t)\}$ and set

$$V(\hat{P}_t) := V(P_t) \dot{\cup} V_S.$$

The subgraph induced by \hat{P}_t on $V(P_t)$ is just P_t itself. The vertex v_S is connected via red edges to all vertices in S and has no further neighbors. A drawing of the gadget attached to S_{c_i} in \hat{P}_t can be found in Figure 1.

► **Lemma 19.** *Let G and \mathcal{T} be as before. Then for every $t \in V(T)$, it holds that*

$$\text{tww}(\tilde{P}_t, S_t) \leq \max(2^k \text{tww}(\hat{P}_t) + 2^{k+1} - 2, 4^k + 2^k - 2).$$

In particular, $\text{tww}(G) \leq \max(2^k \max_{t \in V(T)} \text{tww}(\hat{P}_t) + 2^{k+1} - 2, 4^k + 2^k - 2)$.

Proof sketch. If no two child separators of P_t are contained in each other, the claim can be proven by first applying Corollary 16 and then carefully contracting \tilde{P}_t to \hat{P}_t . The general case can be reduced to this special case. \blacktriangleleft

Next, we want to define a version \overline{P}_t of the parts which does not need extra vertices in P_t but instead marks the separators via red cliques. Indeed, let \overline{P}_t be the trigraph obtained from P_t by completing each of the sets $S \in \mathcal{S}_t$ to a red clique. Thus, the underlying graphs of the trigraphs \overline{P}_t are just the *torsos* of the tree decomposition. We thus call the graphs \overline{P}_t the *red torsos* of the tree decomposition \mathcal{T} .

► **Lemma 20.** *For every $t \in V(T)$, it holds that*

$$\text{tww}(\widehat{P}_t) \leq \max \left(k + 1, \text{tww}(\overline{P}_t) + \binom{\text{tww}(\overline{P}_t)}{k-1}, \text{tww}(\overline{P}_t) + \binom{2k-3}{k-1} \right).$$

In particular,

$$\text{tww}(G) \leq \max \left(\begin{array}{l} 2^k \max_{t \in V(T)} \left(\text{tww}(\overline{P}_t) + \binom{\text{tww}(\overline{P}_t)}{k-1} \right) + 2^{k+1} - 2, \\ 2^k \max_{t \in V(T)} \text{tww}(\overline{P}_t) + 2^k \binom{2k-3}{k-1} + 2^{k+1} - 2, \\ 4^k + 2^k - 2 \end{array} \right)$$

Proof sketch. We extend a contraction sequence of \overline{P}_t to a contraction sequence of \widehat{P}_t while ensuring that the neighborhoods of vertices v_S are not contained in each other. Then, the bound on the red degree of vertices can be proven via a variant of Sperner's theorem [22]. ◀

Combining Lemma 19 and Lemma 20, we get the following two asymptotic bounds on the twin-width of a graph admitting a tree decomposition of small adhesion.

► **Theorem 5.** *For every $k \in \mathbb{N}$ there exist explicit constants D_k and D'_k such that for every graph G with a tree decomposition of adhesion k and parts P_1, P_2, \dots, P_ℓ , the following statements are satisfied:*

1. $\text{tww}(G) \leq 2^k \max_{i \in [\ell]} \text{tww}(\widehat{P}_i) + D_k$,
2. if $k \geq 3$, then $\text{tww}(G) \leq \frac{2^k}{(k-1)!} \max_{i \in [\ell]} \text{tww}(\overline{P}_i)^{k-1} + D'_k$.

4.5 Tri- and quasi-4-connected components

We now want to apply these general results on the interplay between twin-width and tree decompositions of small adhesion to obtain bounds on the twin-width of graphs in terms of the twin-width of their tri- and quasi-4-connected components.

► **Theorem 3.** *Let C_1, C_2, \dots, C_ℓ be the triconnected components of a biconnected graph G . If we write \overline{C}_i for the red torsos of the triconnected components C_i , then*

$$\text{tww}(G) \leq \max \left(8 \max_{i \in [\ell]} \text{tww}(\overline{C}_i) + 6, 18 \right).$$

Proof. This follows from Lemma 20 applied to the tree of triconnected components of G together with the observation that for $k = 2$, the second term in the maximum in Lemma 20 is always bounded by the maximum of the first and third term. ◀

Note that in Theorem 3 we cannot hope for a lower bound similar to the lower bound in Theorem 2 without dropping the virtual edges. Indeed, consider a 3-connected graph G of large twin-width (e.g. Paley graphs or Rook's graphs). By [3, 2], a $(2\lceil \log(|G|) \rceil - 1)$ -subdivision H of G has twin-width at most 4, but its triconnected components are G and multiple long cycles. Thus, there exist graphs of twin-width at most 4 with triconnected components of arbitrarily large twin-width.

Moreover, the red virtual edges in each separator can also not be replaced by black edges.

► **Lemma 21.** *There exists a family of graphs $(G_n)_{n \in \mathbb{N}}$ with unbounded twin-width such that the twin-width of the class of triconnected components of G_n with black virtual edges is bounded.*

Proof. Let G_n be the graph obtained from a clique K_n by subdividing every edge once. The triconnected components of this graph are the K_n and a K_3 for every edge of the K_n , which all have twin-width 0.

In order to show that the twin-width of the family $(G_n)_{n \in \mathbb{N}}$ is unbounded, we show that for every $d \geq 2$ and $n \geq n_d := (d+1)\binom{d}{2} + 1$, we have $\text{tw}(G_n) > d$. For this, consider any d -contraction sequence of G_n for $n \geq n_d$ and let \mathcal{P} be the partition of $V(G_n)$ right before the first contraction in the sequence that does not contract two subdivision vertices. We show that every partition class $P \in \mathcal{P}$ has size at most $\binom{d}{2}$. As no subdivision vertices were contracted so far, we only need to consider classes of subdivision vertices. Thus, let $P = \{v_{e_1}, \dots, v_{e_\ell}\}$ be such a class, where $e_1, \dots, e_\ell \in \binom{V(K_n)}{2}$ are edges of the original K_n . If the edges e_i all have a common endpoint, then P has red edges to all ℓ other endpoints of these edges, meaning that $\ell \leq d \leq \binom{d}{2}$. Otherwise, P has red edges to all endpoints of all e_i . If $\ell > \binom{d}{2}$, these have to be more than d , which is a contraction. Thus, $|P| = \ell \leq \binom{d}{2}$.

Now, let xy be the next contraction in the sequence. If neither x nor y is a subdivision vertex, then G_n contains precisely $2(n-2)$ -many vertices which are connected to either x or y but not both. In the contracted graph, the contraction would thus create at least $\frac{2(n-2)}{\binom{d}{2}} \geq 2d+2$ red edges incident to the contracted vertex. If, on the other hand, either x or y is a subdivision vertex but the other is not, then x and y have no common neighbors. But as non-subdivision vertices have degree $n-1$ in G_n , contracting these two would create at least $\frac{n-1}{\binom{d}{2}} \geq d+1$ red edges incident to the contracted vertex. Thus, no further contraction keeps the red degree of the sequence bounded by d , which implies $\text{tw}(G_n) > d$. ◀

In the case of separators of size 3, we get two bounds on the twin-width of a graph in terms of its quasi-4-connected components: one linear bound in terms of the subgraphs induced on the quasi-4-connected components together with a common red neighbor for every 3-separator along which the graph was split, and one quadratic bound in terms of the (red) torsos of the quasi-4-connected components.

► **Theorem 4.** *Let G be a triconnected graph with quasi-4-connected components C_1, C_2, \dots, C_ℓ .*

1. *For $i \in [\ell]$ we construct a trigraph \widehat{C}_i by adding for every 3-separator S in C_i along which G was split a vertex v_S which we connect via red edges to all vertices in S . Then*

$$\text{tw}(G) \leq \max \left(8 \max_{i \in [\ell]} \text{tw}(\widehat{C}_i) + 14, 70 \right).$$

2. *For $i \in [\ell]$, denote by \overline{C}_i the red torso of the quasi-4-connected component C_i . Then*

$$\text{tw}(G) \leq \max \left(4 \max_{i \in [\ell]} (\text{tw}(\overline{C}_i)^2 + \text{tw}(\overline{C}_i)) + 14, 70 \right).$$

Proof. The two claims follow from Lemma 19 and Lemma 20 applied to the tree of quasi-4-connected components of G [14, 15] together with the observation that also for $k=3$, the second term in the maximum in Lemma 20 is always bounded by the maximum of the first and third term. ◀

5 Conclusion and further research

We proved that $\text{tw}(G) \leq \frac{3}{2}k + 1 + \frac{1}{2}(\sqrt{k + \ln k} + \sqrt{k} + 2 \ln k)$ if G is a graph of strong tree-width at most k (Theorem 1). Moreover, we demonstrated that asymptotically the twin-width of a Paley graph agrees with its strong tree-width (Lemma 11).

We provided a detailed analysis of the relation between the twin-width of a graph and the twin-width of its highly connected components. Concerning 2-connected graphs, the twin-width of a graph is linear in the twin-width of its biconnected components (Theorem 2). There is a linear upper bound for a slightly modified version of triconnected components (Theorem 3). By further providing a quadratic upper bound on the twin-width of graph given the twin-widths of its modified quasi-4-connected components (Theorem 4) we took one important step further to complete the picture of the interplay of the twin-width of a graph with the twin-width of its highly connected components. As a natural generalization of the above decompositions we considered graphs allowing for a tree decomposition of small adhesion (Theorem 5 and Theorem 6).

It seems worthwhile to integrate our new bounds for practical twin-width computations, for example, with a branch-and-bound approach.

References

- 1 Jungho Ahn, Kevin Hendrey, Donggyu Kim, and Sang-il Oum. Bounds for the Twin-Width of Graphs. *SIAM Journal on Discrete Mathematics*, 36(3):2352–2366, 2022. doi:10.1137/21M1452834.
- 2 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is np-complete. *CoRR*, abs/2112.08953, 2021. arXiv:2112.08953.
- 3 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding Twin-Width at Most 4 Is NP-Complete. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2022.18.
- 4 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.
- 5 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.
- 6 Édouard Bonnet and Hugues Déprés. Twin-width can be exponential in treewidth. *CoRR*, abs/2204.07670, 2022. doi:10.48550/arXiv.2204.07670.
- 7 Édouard Bonnet and Hugues Déprés. Twin-width can be exponential in treewidth. *J. Comb. Theory, Ser. B*, 161:1–14, 2023. doi:10.1016/j.jctb.2023.01.003.
- 8 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: max independent set and coloring. *CoRR*, abs/2007.14161, 2020. arXiv:2007.14161.
- 9 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: max independent set, min dominating set, and coloring. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.35.
- 10 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, and Stéphan Thomassé. Twin-width VI: the lens of contraction sequences. *CoRR*, abs/2111.00282, 2021. arXiv:2111.00282.

- 11 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, and Stéphan Thomassé. Twin-width VI: the lens of contraction sequences. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1036–1056. SIAM, 2022. doi:10.1137/1.9781611977073.45.
- 12 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *CoRR*, abs/2004.14789, 2020. arXiv:2004.14789.
- 13 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.
- 14 Martin Grohe. Quasi-4-connected components. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPICs*, pages 8:1–8:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.8.
- 15 Martin Grohe. Quasi-4-connected components. *CoRR*, abs/1602.04505, 2016. arXiv:1602.04505.
- 16 R. Halin. Tree-partitions of infinite graphs. *Discrete Mathematics*, 97(1):203–217, 1991. doi:10.1016/0012-365X(91)90436-6.
- 17 Irene Heinrich and Simon Raßmann. Twin-width of graphs with tree-structured decompositions, 2023. arXiv:2308.14677.
- 18 John E. Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2:135–158, 1973.
- 19 Hugo Jacob and Marcin Pilipczuk. Bounding twin-width for bounded-treewidth graphs, planar graphs, and bipartite graphs. In Michael A. Bekos and Michael Kaufmann, editors, *Graph-Theoretic Concepts in Computer Science - 48th International Workshop, WG 2022, Tübingen, Germany, June 22-24, 2022, Revised Selected Papers*, volume 13453 of *Lecture Notes in Computer Science*, pages 287–299. Springer, 2022. doi:10.1007/978-3-031-15914-5_21.
- 20 Hugo Jacob and Marcin Pilipczuk. Bounding twin-width for bounded-treewidth graphs, planar graphs, and bipartite graphs. *CoRR*, abs/2201.09749, 2022. arXiv:2201.09749.
- 21 Gonne Kretschmer. Calculating twin-width of graphs. Bachelor’s thesis, Technische Universität Darmstadt, 2023.
- 22 D. Lubell. A short proof of sperner’s lemma. *Journal of Combinatorial Theory*, 1(2):299, 1966. doi:10.1016/S0021-9800(66)80035-2.
- 23 D. Seese. Tree-partite graphs and the complexity of algorithms. In Lothar Budach, editor, *Fundamentals of Computation Theory*, Lecture Notes in Computer Science, pages 412–421. Springer, 1985. doi:10.1007/BFb0028825.
- 24 Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.
- 25 David R. Wood. On tree-partition-width. *European Journal of Combinatorics*, 30(5):1245–1253, 2009. doi:10.1016/j.ejc.2008.11.010.

Dynamic Programming on Bipartite Tree Decompositions

Lars Jaffke ✉

Department of Informatics, University of Bergen, Norway

Laure Morelle ✉

LIRMM, Université de Montpellier, CNRS, France

Ignasi Sau ✉

LIRMM, Université de Montpellier, CNRS, France

Dimitrios M. Thilikos ✉

LIRMM, Université de Montpellier, CNRS, France

Abstract

We revisit a graph width parameter that we dub *bipartite treewidth*, along with its associated graph decomposition that we call *bipartite tree decomposition*. Bipartite treewidth can be seen as a common generalization of treewidth and the odd cycle transversal number. Intuitively, a bipartite tree decomposition is a tree decomposition whose bags induce almost bipartite graphs and whose adhesions contain at most one vertex from the bipartite part of any other bag, while the width of such decomposition measures how far the bags are from being bipartite. Adapted from a tree decomposition originally defined by Demaine, Hajiaghayi, and Kawarabayashi [SODA 2010] and explicitly defined by Tazari [Theor. Comput. Sci. 2012], bipartite treewidth appears to play a crucial role for solving problems related to odd-minors, which have recently attracted considerable attention. As a first step toward a theory for solving these problems efficiently, the main goal of this paper is to develop dynamic programming techniques to solve problems on graphs of small bipartite treewidth. For such graphs, we provide a number of **para-NP**-completeness results, FPT-algorithms, and XP-algorithms, as well as several open problems. In particular, we show that K_t -SUBGRAPH-COVER, WEIGHTED VERTEX COVER/INDEPENDENT SET, ODD CYCLE TRANSVERSAL, and MAXIMUM WEIGHTED CUT are FPT parameterized by bipartite treewidth. We also provide the following complexity dichotomy when H is a 2-connected graph, for each of the H -SUBGRAPH-PACKING, H -INDUCED-PACKING, H -SCATTERED-PACKING, and H -ODD-MINOR-PACKING problems: if H is bipartite, then the problem is **para-NP**-complete parameterized by bipartite treewidth while, if H is non-bipartite, then the problem is solvable in XP-time. Beyond bipartite treewidth, we define 1- \mathcal{H} -treewidth by replacing the bipartite graph class by any graph class \mathcal{H} . Most of the technology developed here also works for this more general parameter.

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

Keywords and phrases tree decomposition, bipartite graphs, dynamic programming, odd-minors, packing, maximum cut, vertex cover, independent set, odd cycle transversal

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.26

Related Version *Full Version*: <https://doi.org/10.48550/arXiv.2309.07754>

Funding The second and the third authors were supported by the ANR project ELIT (ANR-20-CE48-0008-01), the three last authors were supported by the French-German Collaboration ANR/DFG Project UTMA (ANR-20-CE92-0027), and the first author was supported by the Research Council of Norway (No 274526).

Acknowledgements We thank Sebastian Wiederrecht and the reviewers for helpful remarks.



© Lars Jaffke, Laure Morelle, Ignasi Sau, and Dimitrios M. Thilikos;
licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 26; pp. 26:1–26:22

Leibniz International Proceedings in Informatics



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

1 Introduction

A graph H is said to be an *odd-minor* of a graph G if it can be obtained from G by iteratively removing vertices, edges, and contracting edge cuts. Hadwiger's conjecture [15], which is open since 1943, states that if a graph excludes K_t as a minor, then its chromatic number is at most $t - 1$. In 1993, Gerards and Seymour [19] generalized this conjecture to odd-minors, hence drawing attention to odd-minors: the Odd Hadwiger's conjecture states that if a graph excludes K_t as an odd-minor, then its chromatic number is at most $t - 1$. Since then, a number of papers regarding odd-minors appeared. Most of them focused to the resolution of the Odd Hadwiger's conjecture (see for instance [11], and [30] for a nice overview of the results), while some others aimed at extending the results of graph minor theory to odd-minors (see for instance [6, 16, 22]). In particular, Demaine, Hajiaghayi, and Kawarabayashi [6] provided a structure theorem which essentially states that graphs excluding an odd-minor can be obtained by clique-sums of almost-embeddable graphs and almost bipartite graphs. To prove this, they implicitly proved the following, which is described more explicitly by Tazari [31].

► **Proposition 1** ([31], adapted from [6]). *Let H be a fixed graph and let G be a given H -odd-minor-free graph. There exists a fixed graph H' , $\kappa, \mu \in \mathbb{N}$ depending only on H , and an explicit uniform algorithm that computes a rooted tree decomposition of G such that:*

- *the adhesion of two nodes has size at most κ , and*
- *the torso of each bag B either consists of a bipartite graph W_B together with μ additional vertices (bags of Type 1) or is H' -minor-free (bags of Type 2).*

Furthermore, the following properties hold:

1. *Bags of Type 2 appear only in the leaves of the tree decomposition,*
2. *if B_2 is a bag that is a child of a bag B_1 in the tree decomposition, then $|B_2 \cap V(W_{B_1})| \leq 1$; and if B_2 is of Type 1, then $|B_1 \cap V(W_{B_2})| \leq 1$ as well,*
3. *the algorithm runs in time $\mathcal{O}_H(|V(G)|^4)$, and*
4. *the μ additional vertices of the bags of Type 1, called apex vertices, can be computed within the same running time.*

It is worth mentioning that Condition 2 of Proposition 1 is slightly stronger than what is stated in [31], but it follows from the proof of [6, Theorem 4.1].

The tree decomposition described in Proposition 1 seems hence adapted to study problems related to odd-minors. As a first step toward building a theory for solving such problems, we study in this paper a new type of tree decomposition, which we call *bipartite tree decomposition*, corresponding to the tree decompositions of Proposition 1, but where all bags are only of Type 1. We also stress that this decomposition has also been implicitly used in [21] and is also introduced, under the same name, in [4].

Bipartite treewidth. Let \mathcal{B} denotes the class of bipartite graphs. A *bipartite tree decomposition* of a graph G is a triple (T, α, β) , where T is a tree and $\alpha, \beta : V(T) \rightarrow 2^{V(G)}$, such that

- $(T, \alpha \cup \beta)$ is a tree decomposition of G ,
- for every $t \in V(T)$, $\alpha(t) \cap \beta(t) = \emptyset$,
- for every $t \in V(T)$, $G[\beta(t)] \in \mathcal{B}$, and
- for every $tt' \in E(T)$, $|(\alpha \cup \beta)(t') \cap \beta(t)| \leq 1$.

The *width* of (T, α, β) is equal to $\max \{ |\alpha(t)| \mid t \in V(T) \}$. The *bipartite treewidth* of G , denoted by $\text{btw}(G)$, is the minimum width over all bipartite tree decompositions of G .

It follows easily from the definition that $\text{btw}(G) = 0$ if and only if G is bipartite (indeed, to prove the sufficiency, just take a single bag containing the whole bipartite graph, with no apex vertices). More generally, for every graph G it holds that $\text{btw}(G) \leq \text{oct}(G)$, where oct denotes the size of a minimum *odd cycle transversal*, that is, a vertex set intersecting every odd cycle. On the other hand, since a bipartite tree decomposition is a tree decomposition whose width is not larger than the maximum size of a bag (in each bag, just declare all vertices as apices), for every graph G it holds that $\text{btw}(G) \leq \text{tw}(G) + 1$, where tw denotes treewidth. Thus, bipartite treewidth can be seen as a common generalization of treewidth and the odd cycle transversal number. Hence, an FPT-algorithm parameterized by btw should generalize *both* FPT-algorithms parameterized by tw and by oct . Since our goal is to develop a theory for solving problems related to odd-minors, the first prerequisite is that bipartite treewidth is closed under odd-minors. Fortunately, this is indeed the case (cf. [17, Lemma 3.2]). Interestingly, this would *not* be true anymore if, in Condition 2 of Proposition 1, the considered intersections were required to be upper-bounded by some integer larger than one (cf. [17, Lemma 3.3]).

This type of tree decomposition has been already used implicitly by Kawarabayashi and Reed [21] in order to solve ODD CYCLE TRANSVERSAL parameterized by the solution size. Independently of our work, Campbell, Gollin, Hendrey, and Wiederrecht [4] are also currently studying bipartite tree decompositions. In particular, they provide universal obstructions characterizing bounded btw in the form of a “grid theorem” (actually the result of [4] apply in the much more general setting of undirected group labeled graphs). They also designed an FPT-approximation algorithm that can construct a bipartite tree decomposition in time $g(k) \cdot n^4 \log n$. This FPT-approximation is an important prerequisite for our algorithmic results as it permits us to assume that, for the implementation of our algorithms, some (approximate) bipartite tree decomposition is provided in advance.

Our aim is to provide a general framework for the design of dynamic programming algorithms on bipartite tree decompositions and, more generally, on a broader type of decompositions that we call *1- \mathcal{H} -tree decompositions*. These decompositions generalize bipartite tree decompositions, in the sense that the role of bipartite graphs is replaced by a general graph class \mathcal{H} .

Our results. In this article we formally introduce bipartite treewidth and bipartite tree decompositions (noticing that they were implicitly already used before, as discussed above). We then focus on the complexity of various problems when the bipartite treewidth of the input graph is taken as a parameter. In particular, we show the following (cf. Table 1):

- While a graph with btw at most k is $(k + 2)$ -colorable (cf. [17, Lemma 6.1]), 3-COLORING is NP-complete even on graphs of oct of size three (cf. [17, Lemma 6.2]), and thus btw at most three.
- K_t -SUBGRAPH-COVER, WEIGHTED VERTEX COVER/INDEPENDENT SET, ODD CYCLE TRANSVERSAL, and MAXIMUM WEIGHTED CUT are FPT parameterized by btw (cf. [17, Corollaries 4.3, 4.5, 4.6, 4.7]). In particular, our FPT-algorithms extend the domain where these well-studied problems can be solved in polynomial time to graphs that are “locally close to being bipartite”. Furthermore, as $\text{btw}(G) \leq \text{oct}(G)$ for any graph G , we think that the fact that ODD CYCLE TRANSVERSAL is FPT parameterized by btw is relevant by itself, as it generalizes the well-known FPT-algorithms parameterized by the solution size [25, 28]. We would like to mention that combining in a win-win manner our dynamic programming algorithm with the FPT-approximation and the Grid Exclusion Theorem

- of [4] we may derive an FPT-algorithm for ODD CYCLE TRANSVERSAL parameterized by the solution size, whose running time is considerably better than the one in [4], which has been obtained independently by using the irrelevant vertex technique (see also [21]).
- Let H be a 2-connected graph. We prove that H -MINOR-PACKING is para-NP-complete parameterized by btw. For each of the H -SUBGRAPH-PACKING, H -INDUCED-SUBGRAPH-PACKING, H -SCATTERED-PACKING, and H -ODD-MINOR-PACKING problems (cf. Appendix B for the definitions), we obtain the following complexity dichotomy: if H is bipartite, then the problem is para-NP-complete parameterized by btw (in fact, even for btw = 0), and if H is non-bipartite, then the problem is solvable in XP-time. The definition of the problems and the XP-algorithms are presented in [17, Section 5] and the hardness results in [17, Lemma 6].
 - In view of the definition of bipartite tree decompositions, it seems natural to consider, instead of bipartite graphs as the “free part” of the bags, any graph class \mathcal{H} . This leads to the more general definition of 1 - \mathcal{H} -tree decomposition and 1 - \mathcal{H} -treewidth (cf. [17, Section 3]), with 1 - $\{\emptyset$ -treewidth being equivalent to the usual treewidth and 1 - \mathcal{B} -treewidth being the bipartite treewidth if \mathcal{B} is the class of bipartite graphs. We introduce these more general definitions because our dynamic programming technologies easily extend to 1 - \mathcal{H} -treewidth. It also seems natural to consider, instead of allowing at most one “bipartite vertex” in each adhesion, allowing any number q of them. For $q = 0$, this corresponds to the \mathcal{H} -treewidth defined in [8] (see also [1, 18] on the study of \mathcal{H} -treewidth for several instantiations of \mathcal{H}). However, as mentioned above, while 1 - \mathcal{B} -treewidth is closed under odd-minors (cf. [17, Lemma 3.2]), this is not the case anymore for $q \geq 2$ (cf. [17, Lemma 3.3]). For $q \geq 2$, some problems remain intractable even when H is not bipartite. As an illustration of this phenomenon, we prove that H -SCATTERED-PACKING (where there cannot be an edge in G among the copies of H to be packed) is para-NP-complete parameterized by q - \mathcal{B} -treewidth for $q \geq 2$ even if H is not bipartite (cf. [17, Lemma 6.7]).

In the statements of the running time of our algorithms, we always let n (resp. m) be the number of vertices (resp. edges) of the input graph of the considered problem.

■ **Table 1** Summary of the results obtained in this article.

Problem	Complexity	Constraints on H /Running time
H (-INDUCED)-SUBGRAPH/ODD-MINOR-COVER [32] H -MINOR-COVER [32]	para-NP-complete, $k = 0$	H bipartite containing P_3 as a subgraph
H (-INDUCED)-SUBGRAPH-PACKING H -MINOR-PACKING		H containing P_3 as a subgraph H bipartite containing P_3 as a subgraph H 2-connected with $ V(H) \geq 3$
H -ODD-MINOR-PACKING H -SCATTERED-PACKING		H 2-connected bipartite with $ V(H) \geq 3$ H 2-connected bipartite with $ V(H) \geq 2$
3-COLORING		
K_t -SUBGRAPH-COVER INDEPENDENT SET WEIGHTED INDEPENDENT SET ODD CYCLE TRANSVERSAL MAXIMUM WEIGHTED CUT		FPT
H -SUBGRAPH-PACKING H -INDUCED-SUBGRAPH-PACKING H -SCATTERED-PACKING H -ODD-MINOR-PACKING	XP	H non-bipartite 2-connected $n^{\mathcal{O}(k)}$

Related results. Other types of tree decompositions integrating some “free bipartite parts” have been defined recently. As we already mentioned, Eiben, Galian, Hamm, and Kwon [8] defined \mathcal{H} -treewidth for a fixed graph class \mathcal{H} . The \mathcal{H} -treewidth of a graph G is essentially the minimum treewidth of the graph induced by some set $X \subseteq V(G)$ such that the connected components of $G \setminus X$ belong to \mathcal{H} , and is equal to 0- \mathcal{H} -treewidth minus one (cf. [17, Section 3]). In particular, when \mathcal{H} is the class of bipartite graphs \mathcal{B} , Jansen and de Kroon [18] provided an FPT-algorithm to test whether the \mathcal{B} -treewidth of a graph is at most k .

Recently, as a first step to provide a systematic theory for odd-minors, Gollin and Wiederrecht [12] defined the \mathcal{H} -blind-treewidth of a graph G , where \mathcal{H} is a property of annotated graphs. Then the \mathcal{H} -blind-treewidth is the smallest k such that G has a tree decomposition where every bag $\beta(t)$ such that $(G, \beta(t)) \notin \mathcal{H}$ has size at most k . For the case where \mathcal{C} consists of every (G, X) where every odd cycle in H as at most one vertex in X , we obtain the \mathcal{C} -blind-treewidth, for which [12] gives an analogue of the Grid Exclusion Theorem [5, 29] under the odd-minor relation. Moreover, [12] provides an FPT-algorithm for INDEPENDENT SET parameterized by \mathcal{C} -blind-treewidth. According to [12], the *bipartite-blind treewidth* of a graph G is lower-bounded by a function of the maximum treewidth over all non-bipartite blocks of G . This immediately implies that bipartite-blind treewidth is lower-bounded by bipartite treewidth. Hence, our FPT-algorithm for INDEPENDENT SET is more general than the one of [12]. Independently of our work, [4] presents an FPT-algorithm to solve ODD CYCLE TRANSVERSAL parameterized by btw in time $f(\text{btw}) \cdot n^4 \log n$ (in fact, they solve a more general group labeled problem). Our algorithm for ODD CYCLE TRANSVERSAL (cf. [17, Corollary 4.6]) is considerably faster.

Organization of the paper. Due to space restrictions, many definitions, results and proofs cannot be provided here, but are available in the full version of the paper [17]. In Section 2 we provide an overview of our techniques. In Section 3 we give a general dynamic programming algorithm to obtain FPT-algorithms, and apply it to MAXIMUM WEIGHTED CUT. Finally, we present several questions for further research in Section 4. Additional necessary definitions are provided in Appendix A.

2 Overview of our techniques

In this section we present an overview of the techniques that we use to obtain our results.

2.1 Dynamic programming algorithms

Compared to dynamic programming on classical tree decompositions, there are two main difficulties for doing dynamic programming on (rooted) *bipartite* tree decompositions. The first one is that the bags in a bipartite tree decomposition may be arbitrarily large, which prevents us from applying typical brute-force approaches to define table entries. The second one, and apparently more important, is the lack of an upper bound on the number of children of each node of the decomposition. Indeed, unfortunately, a notion of “nice bipartite tree decomposition” preserving the width (even approximately) does not exist (cf. [17, Lemma 3.4]). We discuss separately the main challenges involved in our FPT-algorithms and in our XP-algorithms.

2.1.1 FPT-algorithms

In fact, for most of the considered problems, in order to obtain FPT-algorithms parameterized by btw , it would be enough to bound the number of children as a function of btw , but we were not able to come up with a general technique that achieves this property (cf. [17, Lemma 3.4]). For particular problems, however, we can devise ad-hoc solutions. Namely, for K_t -SUBGRAPH-COVER, WEIGHTED VERTEX COVER/INDEPENDENT SET, ODD CYCLE TRANSVERSAL, and MAXIMUM WEIGHTED CUT parameterized by btw , we overcome the above issue by managing to replace the children by constant-sized bipartite gadgets. More specifically, we guess an annotation of the “apex” vertices of each bag t , whose number is bounded by btw , that essentially tells which of these vertices go to the solution or not (with some extra information depending on each particular problem; for instance, for ODD CYCLE TRANSVERSAL, we also guess the side of the bipartition of the non-solution vertices). Having this annotation, each adhesion of the considered node t with a child contains, by the definition of bipartite tree decompositions, at most one vertex v that is not annotated. At this point, we crucially observe that, for the considered problems, we can make local computation for each child, independent from the computations at other children, depending only on the values of the optimum solutions at that child that are required to *contain* or to *exclude* v (note that we need to be able to keep this extra information at the tables of the children). Using the information given by these local computations, we can replace the children of t by constant-sized bipartite gadgets (sometimes empty) so that the newly built graph, which we call a *nice reduction*, is an equivalent instance modulo some constant. If a nice reduction can be efficiently computed for a problem Π , then we say that Π is a *nice problem* (cf. Appendix A, and [17, Section 4] for additional intuition). The newly modified bag has bounded oct , so we can then use an FPT-algorithm parameterized by oct to find the optimal solution with respect to the guessed annotation.

An illustrative example. Before entering into some more technical details and general definitions, let us illustrate this idea with the WEIGHTED VERTEX COVER problem. We want to compute the dynamic programming tables at a bag associated with a node t of the rooted tree given by the bipartite tree decomposition. Remember that the vertices of the bag at t are partitioned into two sets: $\beta(t)$ induces a bipartite graph and its complement, denoted by $\alpha(t)$, corresponds to the apex vertices, whose size is bounded by the parameter, namely btw . The first step is to guess, in time at most 2^{btw} , which vertices in $\alpha(t)$ belong to the desired minimum vertex cover. After such a guess, all the vertices in $\alpha(t)$ can be removed from the graph, by also removing the neighborhood of those that were *not* taken into the solution. The definition of bipartite tree decomposition implies that, in each adhesion with a child of the current bag, there is at most one “surviving” vertex. Let v be such a vertex belonging to the adhesion with a child t' of t . Suppose that, inductively, we have computed in the tables for t' the following two values, subject to the choice that we made for $\alpha(t)$: the minimum weight w_v of a vertex cover in the graph below t' that contains v , and the minimum weight $w_{\bar{v}}$ of a vertex cover in the graph below t' that does *not* contain v . Then, the trick is to observe that, having these two values at hand, we can totally forget the graph below t' : it is enough to delete this whole graph, except for v , and attach a new pendant edge vu , where u is a new vertex, such that v is given weight w_v and u is given weight $w_{\bar{v}}$. It is easy to verify that this gadget mimics, with respect to the current bag, the behavior of including vertex v or not in the solution for the child t' . Adding this gadget for every child results in a bipartite graph, where we can just solve WEIGHTED VERTEX COVER in

polynomial time using a classic algorithm [23, 27], and add the returned weight to our tables. The running time of this whole procedure, from the leaves to the root of the decomposition, is clearly FPT parameterized by the bipartite treewidth of the input graph.

Extensions and limitations. Note that the algorithm sketched above for WEIGHTED VERTEX COVER is problem-dependent, in particular the choice of the gadgets for the children, and the fact of deleting the neighborhood of the vertices chosen in the solution. Which type of replacements and reductions can be afforded in order to obtain an FPT-algorithm for bipartite treewidth? For instance, concerning the gadgets for the children, as far as the considered problem can be solved in polynomial time on bipartite graphs, we could attach to the “surviving” vertices an arbitrary bipartite graph instead of just an edge. If we assume that the considered problem is FPT parameterized by *oct* (which is a reasonable assumption, as *btw* generalizes *oct*), then one could think that it may be sufficient to devise gadgets with bounded *oct*. Unfortunately, this will not work in general: even if each of the gadgets has bounded *oct* (take, for instance, a triangle), since we do not have any upper bound, in terms of *btw*, on the number of children (hence, the number of different adhesions), the resulting graph after the gadget replacement may have unbounded *oct*. In order to formalize the type of replacements and reductions that can be allowed, we introduce in Appendix A the notions of *nice reduction* and *nice problem*, along with an illustration (cf. Figure 1). Additional insights into these definitions, which are quite lengthy, are provided in [17, Section 4.1].

Another sensitive issue is that of “guessing the vertices into the solution”. While this is quite simple for WEIGHTED VERTEX COVER (either a vertex is in the solution, or it is not), for some other problems we may have to guess a richer structure in order to have enough information to combine the tables of the children into the tables of the current bag. This is the reason for which, in the general dynamic programming scheme that we present in Section 3, we deal with *annotated problems*, i.e., problems that receive as input, apart from a graph, a collection of annotated sets in the form of a partition \mathcal{X} of some $X \subseteq V(G)$. For instance, for WEIGHTED VERTEX COVER, we define its *annotated extension*, which we call ANNOTATED WEIGHTED VERTEX COVER, that takes as an input a graph G and two disjoint sets R and S of vertices of G , and asks for a minimum vertex cover S^* such that $S \subseteq S^*$ and $S^* \cap R = \emptyset$.

General dynamic programming scheme. Our general scheme essentially says that if a problem Π has an annotated extension Π' that is

- a nice problem and
- solvable in FPT-time parameterized by *oct*,

then Π is solvable in FPT-time parameterized by *btw*. More specifically, it is enough to prove that Π' is solvable in time $f(|X|) \cdot n^{\mathcal{O}(1)}$ on an instance (G, \mathcal{X}) such that $G \setminus X$ is bipartite, where \mathcal{X} is a partition of X corresponding to the annotation. This general dynamic programming algorithm works in a wider setting, namely for a general graph class \mathcal{H} that plays the role of bipartite graphs, as far as the annotated extension Π' is what we call \mathcal{H} -*nice*; cf. Lemma 2 for the details.

Applications. We then apply this general framework to give FPT-algorithms for several problems parameterized by bipartite treewidth. For each of MAXIMUM WEIGHTED CUT (Subsection 3.4), K_t -SUBGRAPH-COVER (cf. [17, Section 4.4.1]), WEIGHTED VERTEX COVER/INDEPENDENT SET (cf. [17, Section 4.4.2]), and ODD CYCLE TRANSVERSAL (cf. [17, Section 4.4.3]), we prove that the problem has an annotated extension that is 1) nice and 2) solvable in FPT-time parameterized by *oct*, as discussed above.

To prove that an annotated problem has a nice reduction, we essentially use two ingredients. Given two compatible boundaried graphs \mathbf{F} and \mathbf{G} with boundary X (a boundaried graph is essentially a graph along with some labeled vertices that form a boundary, see the formal definition in Appendix A), an annotated problem is usually nice if the following hold:

- (*Gluing property*) Given that we have guessed the annotation \mathcal{X} in the boundary X , a solution compatible with the annotation is optimal in the graph $\mathbf{F} \oplus \mathbf{G}$ obtained by gluing \mathbf{F} and \mathbf{G} if and only if it is optimal in each of the two glued graphs. In this case, it means that the optimum on $(\mathbf{F} \oplus \mathbf{G}, \mathcal{X})$ is equal to the optimum on (F, \mathcal{X}) modulo some constant depending only on G and \mathcal{X} .
- (*Gadgetization*) Given that we have guessed the annotation in the boundary $X \setminus \{v\}$ for some vertex v in X , there is a small boundaried graph G' , that is bipartite (maybe empty), such that the optimum on $(\mathbf{F} \oplus \mathbf{G}, \mathcal{X})$ is equal to the optimum on $(\mathbf{F} \oplus \mathbf{G}', \mathcal{X})$ modulo some constant depending only on G and \mathcal{X} .

The gluing property seems critical to show that a problem is nice. This explains why we solve H -SUBGRAPH-COVER only when H is a clique. For any graph H , ANNOTATED H -SUBGRAPH-COVER is defined similarly to ANNOTATED WEIGHTED VERTEX COVER by specifying vertices that must or must not be taken in the solution. If H is a clique, then we crucially use the fact that H is a subgraph of $\mathbf{F} \oplus \mathbf{G}$ if and only if it is a subgraph of either F or G to prove that ANNOTATED H -SUBGRAPH-COVER has the gluing property. However, we observe that if H is not a clique, then ANNOTATED H -SUBGRAPH-COVER does not have the gluing property (cf. [17, Lemma 4.3]). This is the main difficulty that we face to solve H -SUBGRAPH-COVER in the general case.

Note also that if we define the annotated extension of ODD CYCLE TRANSVERSAL in a similar fashion (that is, a set S of vertices contained in the solution and a set R of vertices that do not belong to the solution), then we can prove that this annotated extension does not have the gluing property. However, if we define ANNOTATED ODD CYCLE TRANSVERSAL as the problem that takes as an input a graph G and three disjoint sets S, X_1, X_2 of vertices of G and aims at finding an odd cycle transversal S^* of minimum size such that $S \subseteq S^*$ and X_1 and X_2 are on different sides of the bipartition obtained after removing S^* , then ANNOTATED ODD CYCLE TRANSVERSAL does have the gluing property (cf. [17, Lemma 4.9]).

For MAXIMUM WEIGHTED CUT, the annotation is pretty straightforward: we use two annotation sets X_1 and X_2 , corresponding to the vertices that will be on each side of the cut. It is easy to see that this annotated extension has the gluing property (cf. Lemma 3).

Finding the right gadgets is the main difficulty to prove that a problem is nice. As explained above, for ANNOTATED WEIGHTED VERTEX COVER, we replace the boundaried graph \mathbf{G} by an edge that simulates the behavior of G with respect to v , which is the only vertex that interest us (cf. [17, Lemma 4.7]). For ANNOTATED MAXIMUM WEIGHTED CUT, if $\mathcal{X} = (X_1, X_2)$, the behavior of G can be simulated by an edge between v and a vertex in X_1 of weight equal to the optimum on $(G, (X_1, X_2 \cup \{v\}))$ and an edge between v and a vertex in X_2 of weight equal to the optimum on $(G, (X_1 \cup \{v\}, X_2))$ (see Lemma 4). For ANNOTATED K_t -SUBGRAPH-COVER, if $\mathcal{X} = (R, S)$, depending on the optimum on $(G, (R \cup \{v\}, S))$ and the one on $(G, (R, S \cup \{v\}))$, we can show that the optimum on $(\mathbf{F} \oplus \mathbf{G}, \mathcal{X})$ is equal to the optimum on (F, \mathcal{X}) or $(F \setminus \{v\}, \mathcal{X})$ modulo some constant (cf. [17, Lemma 4.4]). For ANNOTATED ODD CYCLE TRANSVERSAL, if $\mathcal{X} = (S, X_1, X_2)$, we can show that the optimum on $(\mathbf{F} \oplus \mathbf{G}, \mathcal{X})$ is equal modulo some constant to the optimum on either (F, \mathcal{X}) , or $(F \setminus \{v\}, \mathcal{X})$, or (F', \mathcal{X}) , where F' is obtained from F by adding an edge between v and either a vertex of X_1 or a vertex of X_2 (cf. [17, Lemma 4.10]).

Finally, let us now mention some particular ingredients used to prove that the considered annotated problems are solvable in time $f(|X|) \cdot n^{\mathcal{O}(1)}$ on an instance (G, \mathcal{X}) such that $G \setminus X$ is bipartite, where \mathcal{X} is a partition of a vertex set X corresponding to the annotation. For ANNOTATED K_t -SUBGRAPH-COVER and ANNOTATED WEIGHTED VERTEX COVER, this is simply a reduction to (WEIGHTED VERTEX) COVER on bipartite graphs. For ODD CYCLE TRANSVERSAL, we adapt the algorithm of Reed, Smith, and Vetta [28] that uses iterative compression to solve ANNOTATED ODD CYCLE TRANSVERSAL in FPT-time parameterized by `oct`, so that it takes annotations into account (cf. [17, Lemma 4.12]). As for MAXIMUM WEIGHTED CUT parameterized by `oct`, the most important trick is to reduce to a K_5 -odd-minor-free graph, and then use known results of Grötschel and Pulleyblank [13] and Guenin [14] to solve the problem in polynomial time (Proposition 6).

2.1.2 XP-algorithms

We now sketch some of the basic ingredients of the XP-algorithms that we present in [17, Section 5] for H (-INDUCED)-SUBGRAPH/SCATTERED/ODD-MINOR-PACKING. The main observation is that, if H is 2-connected and non-bipartite, since the “non-apex” part of each bag is bipartite and H is non-bipartite, in any H -subgraph/induced/scattered/odd-minor-packing and every bag of the decomposition, there are at most `btw` occurrences of H that intersect that bag. We thus guess these occurrences, and how they intersect the children, which allow us to reduce the number of children by just deleting those not involved in the packing. The guess of these occurrences is the dominant term in the running time of the resulting XP-algorithm using this method. Note that for H (-INDUCED)-SUBGRAPH/SCATTERED-PACKING, we can indeed easily guess those occurrences in XP-time parameterized by `btw`, as the total size of the elements of the packing intersecting a given bag is bounded by a function of `btw` and H . However, for H -ODD-MINOR-PACKING, this is not the case anymore, as an element of the packing may contain an arbitrary number of vertices in the bipartite part of a bag. We overcome this issue as follows. As stated in [17, Lemma 3.1], the existence of an H -odd-minor is equivalent to the existence of a so-called *odd H -expansion*, which is essentially a collection of trees connected by edges preserving the appropriate parities of the resulting cycles. In an odd H -expansion, the *branch vertices* are those that have degree at least three, or that are incident to edges among different trees. Note that, in an odd H -expansion, the number of branch vertices depends only on H (cf. [17, Lemma 5.2]). Equipped with this property, we first guess, at a given bag, the branch vertices of the packing that intersect that bag. Note that this indeed yields an XP number of choices, as required. Finally, for each such a choice, we use an FPT-algorithm of Kawarabayashi, Reed, and Wollan [22] solving the PARITY k -DISJOINT PATHS to check whether the guessed packing exists or not. This approach is formalized in [17, Lemma 5.3].

It is worth mentioning that, as discussed in Section 4, we leave as an open problem the existence of FPT-algorithms for the above packing problems parameterized by `btw`.

2.2 Hardness results

Finally, we discuss some of the tools that we use to obtain the `para-NP`-completeness results summarized in Table 1, which can be found in [17, Section 6]. We present a number of different reductions, some of them consisting of direct simple reductions, such as the one we provide for 3-COLORING in [17, Lemma 6.2].

26:10 Dynamic Programming on Bipartite Tree Decompositions

Except for 3-COLORING, all the considered problems fall into two categories: covering or packing problems. For the first family (cf. [17, Section 6.2]), the para-NP-completeness is an immediate consequence of a result of Yannakakis [32] that characterizes hereditary graph classes \mathcal{G} for which VERTEX DELETION TO \mathcal{G} on bipartite graphs is polynomial-time solvable and those for which VERTEX DELETION TO \mathcal{G} remains NP-complete.

For the packing problems (cf. [17, Section 6.2]), we do not have such a general result as for the covering problems, and we provide several reductions for different problems. For instance, we prove in [17, Lemma 6.3] that if H is a bipartite graph containing P_3 as a subgraph, then H -SUBGRAPH-PACKING and H -INDUCED-SUBGRAPH-PACKING are NP-complete on bipartite graphs. The proof consists in a careful analysis and a slight modification of a reduction of Kirkpatrick and Hell [24] for the problem of partitioning the vertex set of an input graph G into subgraphs isomorphic to a fixed graph H . The hypothesis about containing P_3 is easily seen to be tight.

For the minor version, we prove in [17, Lemma 6.4] that if H is a 2-connected graph with at least three vertices, then H -MINOR-PACKING is NP-complete on bipartite graphs. The proof uses a reduction from P_3 -SUBGRAPH-PACKING on bipartite graphs, which was proved to be NP-complete by Monnot and Toulouse [26]. The 2-connectivity of H is crucially used in the proof. Given that odd-minors preserve cycle parity (cf. [17, Lemma 3.1]), when H is bipartite, H -ODD-MINOR-PACKING and H -MINOR-PACKING are the same problem on bipartite graphs. Hence, the same hardness result holds for H -ODD-MINOR-PACKING when H is 2-connected and bipartite (cf. [17, Lemma 6.5]).

In [17, Lemma 6.6] we prove that, if H is a 2-connected bipartite graph with at least one edge, then H -SCATTERED-PACKING is NP-complete on bipartite graphs, by a simple reduction from the INDUCED MATCHING on bipartite graphs, which is known to be NP-complete [3].

Finally, in [17, Lemma 6.7] we prove that if H is a (non-necessarily bipartite) 2-connected graph containing an edge and $q \in \mathbb{N}_{\geq 2}$, then H -SCATTERED-PACKING is para-NP-complete parameterized by q - \mathcal{B} -treewidth. In fact, this reduction is exactly the same as the one when $q = 1$, with the extra observation that, if G' is the graph constructed in the reduction, then the “bipartite” treewidth of G' is at most the one of H for $q \geq 2$.

3 General dynamic programming to obtain FPT-algorithms

In this section, we give introduce a framework for giving FPT-algorithms for problems parameterized by the width of a given bipartite tree decomposition of the input graph. In Subsection 3.1 we provide some preliminary definitions and notations, especially concerning *annotated problems*. Due to space constraints *treewidth*, *boundaried graphs*, and *nice problems* are defined in Appendix A. In Subsection 3.2 we provide a dynamic programming scheme for nice problems, along with some generalizations of this scheme in Subsection 3.3. Finally, we give an application to MAXIMUM WEIGHTED CUT in Subsection 3.4. Applications to K_t -VERTEX COVER, WEIGHTED VERTEX COVER, and ODD CYCLE TRANSVERSAL are additionally given in [17].

3.1 Preliminaries

Partitions. Given $p \in \mathbb{N}$, a p -partition of a set X is a tuple (X_1, \dots, X_p) of pairwise disjoint subsets of X such that $X = \bigcup_{i \in [p]} X_i$. We denote by $\mathcal{P}_p(X)$ the set of all p -partitions of X . Given a partition $\mathcal{X} \in \mathcal{P}_p(X)$, its domain X is also denoted as $\cup \mathcal{X}$. A *partition* is a p -partition for some $p \in \mathbb{N}$. Note that this corresponds to the usual definition of an ordered near-partition, since we allow empty sets in a p -partition and since the order

matters. Given $Y \subseteq X$, $\mathcal{X} = (X_1, \dots, X_p) \in \mathcal{P}_p(X)$, and $\mathcal{Y} = (Y_1, \dots, Y_p) \in \mathcal{P}_p(Y)$, we say that $\mathcal{Y} \subseteq \mathcal{X}$ if $Y_i \subseteq X_i$ for each $i \in [p]$. Given a set U , two subsets $X, A \subseteq U$, and $\mathcal{X} = (X_1, \dots, X_p) \in \mathcal{P}_p(X)$, $\mathcal{X} \cap A$ denotes the partition $(X_1 \cap A, \dots, X_p \cap A)$ of $X \cap A$.

Optimization problems. A *p-partition-evaluation function on graphs* is a function f that receives as input a graph G along with a p -partition \mathcal{P} of its vertices and outputs a non-negative integer. Given such a function f and some choice $\text{opt} \in \{\max, \min\}$ we define the associated graph parameter $\mathfrak{p}_{f, \text{opt}}$ where, for every graph G ,

$$\mathfrak{p}_{f, \text{opt}}(G) = \text{opt}\{f(G, \mathcal{P}) \mid \mathcal{P} \text{ is a } p\text{-partition of } V(G)\}.$$

An *optimization problem* is a problem that can be expressed as follows.

Input: A graph G .
Objective: Compute $\mathfrak{p}_{f, \text{opt}}(G)$.

The *annotated extension* of $\mathfrak{p}_{f, \text{opt}}$ is the parameter $\hat{\mathfrak{p}}_{f, \text{opt}}$ such that

$$\hat{\mathfrak{p}}_{f, \text{opt}}(G, \mathcal{X}) = \text{opt}\{f(G, \mathcal{P}) \mid \mathcal{P} \text{ is a } p\text{-partition of } V(G) \text{ with } \mathcal{X} \subseteq \mathcal{P}\}.$$

Observe that $\mathfrak{p}_{f, \text{opt}}(G) = \hat{\mathfrak{p}}_{f, \text{opt}}(G, \emptyset^p)$, for every graph G . The problem Π' is a *p-annotated extension* of the optimization problem Π if Π can be expressed by some p -partition-evaluation function f and some choice $\text{opt} \in \{\max, \min\}$, and that Π' can be expressed as follows.

Input: A graph G and $\mathcal{X} \in \mathcal{P}_p(X)$ for some $X \subseteq V(G)$.
Objective: Compute $\hat{\mathfrak{p}}_{f, \text{opt}}(G, \mathcal{X})$.

We also say that Π' is a *p-annotated problem*.

While our goal in this article is to study bipartite treewidth, defined below, we define a more general parameter, namely 1- \mathcal{H} -treewidth, with the hope of it finding some application in future work. We use the term 1- \mathcal{H} -treewidth to signify that the “ \mathcal{H} -part” of each bag intersects each neighboring bag in at most one vertex. This also has the benefit of avoiding confusion with \mathcal{H} -treewidth defined in [8], which would be another natural name for this class of parameters.

1- \mathcal{H} -tree decompositions. Let \mathcal{H} be a graph class. A *1- \mathcal{H} -tree decomposition* is defined exactly like a tree decomposition, but by replacing the class \mathcal{B} of bipartite graphs by \mathcal{H} .

3.2 General dynamic programming scheme

We now have all the ingredients for our general scheme dynamic programming algorithm on bipartite tree decompositions. We essentially prove that if a problem Π has an annotated extension that is \mathcal{B} -nice and solvable in FPT-time parameterized by oct , then Π is solvable in FPT-time parameterized by btw . This actually holds for more general \mathcal{H} .

► **Lemma 2.** *Let $p \in \mathbb{N}$. Let \mathcal{H} be a graph class. Let Π be an optimization problem. Let Π' be a problem that is:*

- *a p-annotated extension of Π corresponding to some choice of p-partition-evaluation function g and some $\text{opt} \in \{\max, \min\}$,*
- *\mathcal{H} -nice, and*
- *solvable on instances (G, \mathcal{X}) such that $G \setminus \cup \mathcal{X} \in \mathcal{H}$ in time $f(|\cup \mathcal{X}|) \cdot n^c \cdot m^d$, for some $c, d \in \mathbb{N}$.*

26:12 Dynamic Programming on Bipartite Tree Decompositions

Then, there is an algorithm that, given a graph G and a 1- \mathcal{H} -tree decomposition of G of width k , computes $\mathfrak{p}_{f,\text{opt}}(G)$ in time $\mathcal{O}(p^k \cdot f(k + \mathcal{O}(1)) \cdot (k \cdot n)^c \cdot m^d)$ (or $\mathcal{O}(p^k \cdot f(k + \mathcal{O}(1)) \cdot (m + k^2 \cdot n)^d)$ if $c = 0$).

Proof. Let **Alg** be the algorithm that solves instances (G, \mathcal{X}) such that $G \setminus \cup \mathcal{X} \in \mathcal{H}$ in time $f(|\cup \mathcal{X}|) \cdot n^c \cdot m^d$.

Let (T, α, β, r) be a rooted 1- \mathcal{H} -tree decomposition of G of width at most k . Let $\sigma : V(G) \rightarrow \mathbb{N}$ be an injection. For $t \in V(T)$, let $\mathbf{G}_t = (G_t, \delta_t, \sigma|_{\delta_t})$, let $X_t = \alpha(t) \cup \delta_t \cup \bigcup_{t' \in \text{ch}_r(t)} \delta_{t'}$, let $\mathbf{X}_t = (G[X_t], X_t, \sigma|_{X_t})$, let $\mathbf{H}_t = \mathbf{X}_t \boxplus (\boxplus_{t' \in \text{ch}_r(t)} \mathbf{G}_{t'})$, let \mathbf{F}_t be such that $G_t = \mathbf{F}_t \oplus \mathbf{H}_t$. Let $A_t = \alpha(t) \cup \delta_t$, and let $B_t = X_t \setminus A_t = X_t \cap \beta(t) \setminus \delta_t$. Note that $|\text{bd}(\mathbf{G}_{t'}) \setminus A_t| \leq 1$ for $t' \in \text{ch}_r(t)$.

We proceed in a bottom-up manner to compute $s_t^{\mathcal{X}} := \hat{\mathfrak{p}}_{g,\text{opt}}(G_t, \mathcal{X})$, for each $t \in V(T)$, for each $\mathcal{X} \in \mathcal{P}_p(\delta_t)$. Hence, given that $\delta_r = \emptyset$, $s_r^\emptyset = \mathfrak{p}_{g,\text{opt}}(G)$.

Let $t \in V(T)$. By induction, for each $t' \in \text{ch}_r(t)$ and for each $\mathcal{X}_{t'} \in \mathcal{P}_p(\delta_{t'})$, we compute the value $s_{t'}^{\mathcal{X}_{t'}}$. Let $\mathcal{X} \in \mathcal{P}_p(\delta_t)$. Let \mathcal{Q} be the set of all $\mathcal{A} \in \mathcal{P}_p(A_t)$ such that $\mathcal{A} \cap \delta_t = \mathcal{X}$. Let $\mathcal{A} \in \mathcal{Q}$. Since Π' is \mathcal{H} -nice, there is an \mathcal{H} -nice reduction $(\mathbf{H}_{\mathcal{A}}, \mathcal{A}', s_{\mathcal{A}})$ of $(\mathbf{H}_t, \mathcal{A})$ with respect to Π' . Hence, $\hat{\mathfrak{p}}_{g,\text{opt}}(G_t, \mathcal{A}) = \hat{\mathfrak{p}}_{g,\text{opt}}(\mathbf{H}_{\mathcal{A}} \triangleright \mathbf{F}_t, \mathcal{A}') + s_{\mathcal{A}}$. Let us compute $\hat{\mathfrak{p}}_{g,\text{opt}}(\mathbf{H}_{\mathcal{A}} \triangleright \mathbf{F}_t, \mathcal{A}')$.

By definition of a \mathcal{H} -reduction, $(\mathbf{H}_{\mathcal{A}} \triangleright \mathbf{F}_t) \setminus (\cup \mathcal{A}') \in \mathcal{H}$. Hence, we can compute $\hat{\mathfrak{p}}_{g,\text{opt}}(\mathbf{H}_{\mathcal{A}} \triangleright \mathbf{F}_t, \mathcal{A}')$, and thus $\hat{\mathfrak{p}}_{g,\text{opt}}(G_t, \mathcal{A})$, using **Alg** on the instance $(\mathbf{H}_{\mathcal{A}} \triangleright \mathbf{F}_t, \mathcal{A}')$. Finally, $s_t^{\mathcal{X}} = \text{opt}_{\mathcal{A} \in \mathcal{Q}} \hat{\mathfrak{p}}_{g,\text{opt}}(G_t, \mathcal{A})$.

It remains to calculate the complexity. Throughout, we make use of the fact that p is a fixed constant. We can assume that T has at most n nodes: for any pair of nodes t, t' with $(\alpha \cup \beta)(t) \subseteq (\alpha \cup \beta)(t')$, we can contract the edge tt' of T to a new vertex t'' with $\alpha(t'') = \alpha(t)$ and $\beta(t'') = \beta(t')$. This defines a valid 1- \mathcal{H} -tree decomposition of same width. For any leaf t of T , there is a vertex $u \in V(G)$ that only belongs to the bag of t . From this observation, we can inductively associate each node of T to a distinct vertex of G . So this \mathcal{H} -tree decomposition has at most n bags. Hence, if $c_t = |\text{ch}_r(t)|$, then we have $\sum_{t \in V(T)} c_t \leq n$. Let also $n_t = |(\alpha \cup \beta)(t)|$ and $m_t = |E(G[(\alpha \cup \beta)(t)])|$. Note that $|A_t| = |\alpha(t)| + |\delta_t \cap \beta(t)| \leq k + 1$ and that $|B_t| = |\bigcup_{t' \in V(T)} \delta_{t'} \cap \beta(t)| \leq c_t$, so $|X_t| \leq k + 1 + c_t$. Moreover, the properties of the tree decompositions imply that the vertices in $\beta(t) \setminus X_t$ are only present in node t . Then, $\sum_{t \in V(T)} n_t = \sum_{t \in V(T)} (|X_t| + |\beta(t) \setminus X_t|) = \mathcal{O}(k \cdot n)$. Also, let \bar{m}_t be the number of edges only present in the bag of node t . The edges that are present in several bags are those in the adhesion of t and its neighbors. t is adjacent to its $|c_t|$ children and its parent, and an adhesion has size at most $k + 1$. Thus, $\sum_{t \in V(T)} m_t \leq \sum_{t \in V(T)} (\bar{m}_t + k^2(1 + c_t)) = \mathcal{O}(m + k^2 \cdot n)$.

There are $p^{|A_t|} \leq p^{k+1} = \mathcal{O}(p^k)$ partitions of $\mathcal{P}_p(A_t)$. For each of them, we compute in time $\mathcal{O}(k \cdot c_t)$ a \mathcal{H} -nice reduction $(\mathbf{H}_{\mathcal{A}}, \mathcal{A}', s_{\mathcal{A}})$ with $|\cup \mathcal{A}'| = |A_t| + \mathcal{O}(1) = k + \mathcal{O}(1)$ and with $\mathcal{O}(|B_t|) = \mathcal{O}(c_t)$ additional vertices and edges. We thus solve Π' on $(\mathbf{H}_{\mathcal{A}} \triangleright \mathbf{F}_t, \mathcal{A}')$ in time $f(k + \mathcal{O}(1)) \cdot \mathcal{O}((n_t + c_t)^c \cdot (m_t + c_t)^d)$. Hence, the running time is $\mathcal{O}(p^k \cdot f(k + \mathcal{O}(1)) \cdot (k \cdot n)^c \cdot m^d)$ (or $\mathcal{O}(p^k \cdot f(k + \mathcal{O}(1)) \cdot (m + k^2 \cdot n)^d)$ if $c = 0$). \blacktriangleleft

3.3 Generalizations

For the sake of simplicity, we assumed in Lemma 2 that the problem Π under consideration takes as input just a graph. However, a similar statement still holds if we add labels/weights on the vertices/edges of the input graph. This is in particular the case for MAXIMUM WEIGHTED CUT (Subsection 3.4) and WEIGHTED INDEPENDENT SET where the vertices or edges are weighted.

Moreover, again for the sake of simplicity, we assumed that Π' is solvable in FPT-time, while other complexities such as XP-time could be considered. Similarly, in the definition of the nice reduction, the constraints $|A'| = |A| + \mathcal{O}(1)$, $|V(G')| \leq |X| + \mathcal{O}(|B|)$, $|E(G')| \leq |E(G[X])| + \mathcal{O}(|B|)$ can be modified. In both cases, the dynamic programming algorithm still holds, but the running time of Lemma 2 changes.

To give a precise running time for MAXIMUM WEIGHTED CUT (Subsection 3.4), K_t -SUBGRAPH-COVER, and WEIGHTED INDEPENDENT SET, let us observe that, if Π' is solvable in time $f(|\cup \mathcal{X}|) \cdot n'^c \cdot m'^d$, where $G' = G \setminus \cup \mathcal{X}$, $n' = |V(G')|$, and $m' = |E(G')|$, then the running time of Lemma 2 is better. Indeed, in the proof of the complexity of Lemma 2, we now solve Π' on $(\mathbf{H}_{\mathcal{A}} \triangleright \mathbf{F}, \mathcal{A}')$ in time $f(k + \mathcal{O}(1)) \cdot \mathcal{O}((n'_t + c_t)^c \cdot (m'_t + c_t)^d)$, where $n'_t = |\beta(t)|$ and $m'_t = |E(G[\beta(t)])|$. We have $\sum_{t \in V(T)} n'_t = \sum_{t \in V(T)} (|B| + |\beta(t) \cap \delta_t| + |\beta(t) \setminus X|) = \mathcal{O}(n)$ and $\sum_{t \in V(T)} m'_t \leq m$. Hence, the total running time is $\mathcal{O}(p^k \cdot (k \cdot n + f(k + \mathcal{O}(1)) \cdot n^c \cdot m^d))$.

3.4 Application to Maximum Cut

We now apply the above framework to give an FPT-algorithm for MAXIMUM WEIGHTED CUT parameterized by bipartite treewidth. Thanks to Lemma 2, this now reverts to showing that the problem under consideration has an \mathcal{B} -nice annotated extension that is solvable in FPT time when parameterized by oct, where \mathcal{B} is the class of bipartite graphs.

The MAXIMUM WEIGHTED CUT problem is defined as follows.

MAXIMUM WEIGHTED CUT

Input: A graph G and a weight function $w : E(G) \rightarrow \mathbb{N}$.

Objective: Find an edge cut of maximum weight.

Let H be a graph. We define f_{cut} as the 2-partition-evaluation function where, for every graph G with edge weight w and for every $\mathcal{P} = (X_1, X_2) \in \mathcal{P}_2(V(G))$,

$$f_{\text{cut}}(G, \mathcal{P}) = w(\mathcal{P}) = w(E(X_1, X_2)).$$

Hence, MAXIMUM WEIGHTED CUT is the problem of computing $p_{f_{\text{cut}}, \text{max}}(G)$. We call its annotated extension ANNOTATED MAXIMUM WEIGHTED CUT. In other words, ANNOTATED MAXIMUM WEIGHTED CUT is defined as follows.

ANNOTATED MAXIMUM WEIGHTED CUT

Input: A graph G , a weight function $w : E(G) \rightarrow \mathbb{N}$, and two disjoint sets $X_1, X_2 \subseteq V(G)$.

Objective: Find an edge cut of maximum weight such that the vertices in X_1 belongs to one side of the cut, and the vertices in X_2 belong to the other side.

The following property seems critical to show that a problem is \mathcal{H} -nice.

Gluing property. Let Π be a p -annotated problem corresponding to some choice of p -partition-evaluation function f and some $\text{opt} \in \{\text{max}, \text{min}\}$. We say that Π has the *gluing property* if, given two compatible boundaried graphs \mathbf{F} and \mathbf{G} with boundary X , $\mathcal{X} \in \mathcal{P}_p(X)$, and $\mathcal{P} \in \mathcal{P}_p(V(\mathbf{F} \oplus \mathbf{G}))$ such that $\mathcal{X} \subseteq \mathcal{P}$, then $\hat{p}_{f, \text{opt}}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}) = f(\mathbf{F} \oplus \mathbf{G}, \mathcal{P})$ if and only if $\hat{p}_{f, \text{opt}}(\mathbf{F}, \mathcal{X}) = f(\mathbf{F}, \mathcal{P} \cap V(\mathbf{F}))$ and $\hat{p}_{f, \text{opt}}(\mathbf{G}, \mathcal{X}) = f(\mathbf{G}, \mathcal{P} \cap V(\mathbf{G}))$.

We first prove that ANNOTATED MAXIMUM WEIGHTED CUT has the gluing property.

26:14 Dynamic Programming on Bipartite Tree Decompositions

► **Lemma 3** (Gluing property). *ANNOTATED MAXIMUM WEIGHTED CUT has the gluing property. More precisely, given two boundaried graphs $\mathbf{F} = (F, B_F, \rho_F)$ and $\mathbf{G} = (G, B_G, \rho_G)$, a weight function $w : E(\mathbf{F} \oplus \mathbf{G}) \rightarrow \mathbb{N}$, a set $X \subseteq V(\mathbf{F} \oplus \mathbf{G})$ such that $B_F \cap B_G \subseteq X$, and $\mathcal{X} = (X_1, X_2) \in \mathcal{P}_2(X)$, if we set $\bar{w} = w(\mathcal{X} \cap B_F \cap B_G)$, then we have*

$$\hat{\rho}_{f_{\text{cut}}, \text{max}}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}, w) = \hat{\rho}_{f_{\text{cut}}, \text{max}}(F, \mathcal{X} \cap V(F), w) + \hat{\rho}_{f_{\text{cut}}, \text{max}}(G, \mathcal{X} \cap V(G), w) - \bar{w}.$$

Proof. Let $\mathcal{P} \in \mathcal{P}_2(V(\mathbf{F} \oplus \mathbf{G}))$ be such that $\mathcal{X} \subseteq \mathcal{P}$ and $\hat{\rho}_{f_{\text{cut}}, \text{max}}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}, w) = f_{\text{cut}}(\mathbf{F} \oplus \mathbf{G}, \mathcal{P}, w)$. Then,

$$\begin{aligned} \hat{\rho}_{f_{\text{cut}}, \text{max}}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}, w) &= w(\mathcal{P}) \\ &= w(\mathcal{P} \cap V(F)) + w(\mathcal{P} \cap V(G)) - \bar{w} \\ &\leq \hat{\rho}_{f_{\text{cut}}, \text{max}}(F, \mathcal{X} \cap V(F), w) + \hat{\rho}_{f_{\text{cut}}, \text{max}}(G, \mathcal{X} \cap V(G), w) - \bar{w}. \end{aligned}$$

Reciprocally, for $H \in \{F, G\}$, let $\mathcal{P}_H = (X_1^H, X_2^H) \in \mathcal{P}_2(V(H))$ be such that $\mathcal{X} \cap V(H) \subseteq \mathcal{P}_H$ and $\hat{\rho}_{f_{\text{cut}}, \text{max}}(H, \mathcal{X} \cap V(H), w) = f_{\text{cut}}(H, \mathcal{P}_H, w)$. Then, since $\mathcal{P}_H \cap B_F \cap B_G = \mathcal{X} \cap B_F \cap B_G$ for $H \in \{F, G\}$, we have

$$\begin{aligned} \hat{\rho}_{f_{\text{cut}}, \text{max}}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}, w) &\geq w(E(X_1^F \cup X_1^G, X_2^F \cup X_2^G)) \\ &= w(E(X_1^F, X_2^F)) + w(E(X_1^G, X_2^G)) - \bar{w} \\ &= \hat{\rho}_{f_{\text{cut}}, \text{max}}(F, \mathcal{X} \cap V(F), w) + \hat{\rho}_{f_{\text{cut}}, \text{max}}(G, \mathcal{X} \cap V(G), w) - \bar{w}. \quad \blacktriangleleft \end{aligned}$$

We now show how to reduce a graph $\mathbf{F} \oplus \mathbf{G}$ to a graph F' when the boundary of \mathbf{F} and \mathbf{G} has a single vertex v that is not annotated.

► **Lemma 4** (Gadgetization). *Let $\mathbf{F} = (F, B_F, \rho_F)$ and $\mathbf{G} = (G, B_G, \rho_G)$ be two boundaried graphs, let $w : E(\mathbf{F} \oplus \mathbf{G}) \rightarrow \mathbb{N}$ be a weight function, let $X \subseteq V(\mathbf{F} \oplus \mathbf{G})$ be such that $B_F \cap B_G \subseteq X$, let $v \in B_F \cap B_G$, and let $\mathcal{X} = (X_1, X_2) \in \mathcal{P}_2(X \setminus \{v\})$. Suppose that there is $v_1 \in X_1$ and $v_2 \in X_2$ adjacent to v with $w(vv_1) = w(vv_2) = 0$. Let $\mathcal{X}^1 = (X_1 \cup \{v\}, X_2)$ and $\mathcal{X}^2 = (X_1, X_2 \cup \{v\})$. For $a \in [2]$, let $g_a = \hat{\rho}_{f_{\text{cut}}, \text{max}}(G, \mathcal{X}^a \cap V(G), w)$. Let $\bar{w} = w(\mathcal{X} \cap B_F \cap B_G)$. Let $w' : E(F) \rightarrow \mathbb{N}$ be such that $w'(vv_1) = g_2 - \bar{w}$, $w'(vv_2) = g_1 - \bar{w}$, and $w'(e) = w(e)$ otherwise. Then*

$$\hat{\rho}_{f_{\text{cut}}, \text{max}}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}, w) = \hat{\rho}_{f_{\text{cut}}, \text{max}}(F, \mathcal{X}, w').$$

Proof. For $a \in [2]$, let $f_a = \hat{\rho}_{f_{\text{cut}}, \text{max}}(F, \mathcal{X}^a \cap V(F), w)$. Note that in F with partition \mathcal{X} , if v is on the same side as X_1 , then we must count the weight of the edge vv_2 , but not the weight of vv_1 , and vice versa when exchanging 1 and 2. Thus, using Lemma 3, we have

$$\begin{aligned} \hat{\rho}_{f_{\text{cut}}, \text{max}}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}, w) &= \max\{\hat{\rho}_{f_{\text{cut}}, \text{max}}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}^1, w), \hat{\rho}_{f_{\text{cut}}, \text{max}}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}^2, w)\} \\ &= \max\{f_1 + g_1 - \bar{w}, f_2 + g_2 - \bar{w}\} \\ &= \max\{f_1 + w'(vv_2), f_2 + w'(vv_1)\} \\ &= \max\{\hat{\rho}_{f_{\text{cut}}, \text{max}}(F', \mathcal{X}^1, w'), \hat{\rho}_{f_{\text{cut}}, \text{max}}(F', \mathcal{X}^2, w')\} \\ &= \hat{\rho}_{f_{\text{cut}}, \text{max}}(F, \mathcal{X}, w'). \quad \blacktriangleleft \end{aligned}$$

Using Lemma 3 and Lemma 4, we can prove that ANNOTATED MAXIMUM WEIGHTED CUT is \mathcal{H} -nice. Essentially, given an instance $(\mathbf{G} = \mathbf{X} \boxplus (\boxplus_{i \in [d]} \mathbf{G}_i), (A, B), \mathcal{A}, w)$, we reduce \mathbf{G} to \mathbf{X} where we add two new vertices in \mathcal{A} and add every edges between this new vertices and the vertices in B . We then show that if the appropriate weight is given to each new edge, then the resulting boundaried graph is equivalent to \mathbf{G} modulo some constant s .

► **Lemma 5** (Nice problem). *Let \mathcal{H} be a graph class. ANNOTATED MAXIMUM WEIGHTED CUT is \mathcal{H} -nice.*

MAXIMUM WEIGHTED CUT is a NP-hard problem [20]. However, there exists a polynomial-time algorithm when restricted to some graph classes. In particular, Grötschel and Pulleyblank [13] proved that MAXIMUM WEIGHTED CUT is solvable in polynomial-time on weakly bipartite graphs, and Guenin [14] proved that weakly bipartite graphs are exactly K_5 -odd-minor-free graphs, which gives the following result.

► **Proposition 6** ([13,14]). *There is a constant $c \in \mathbb{N}$ and an algorithm that solves MAXIMUM WEIGHTED CUT on K_5 -odd-minor-free graphs in time $\mathcal{O}(n^c)$.*

Moreover, we observe the following.

► **Lemma 7.** *A graph G such that $\text{oct}(G) \leq 2$ does not contain K_5 as an odd-minor.*

Proof. Let $u, v \in V(G)$ be such that $G' = G \setminus \{u, v\}$ is bipartite. G' does not contain K_3 as an odd-minor, so G does not contain K_5 as an odd-minor. ◀

Combining Proposition 6 and Lemma 7, we have that ANNOTATED MAXIMUM WEIGHTED CUT is FPT parameterized by oct .

► **Lemma 8.** *There is an algorithm that, given a graph G , a weight function $w : E(G) \rightarrow \mathbb{N}$, and two disjoint sets $X_1, X_2 \subseteq V(G)$, such that $G' = G \setminus (X_1 \cup X_2)$ is bipartite, solves ANNOTATED MAXIMUM WEIGHTED CUT on (G, X_1, X_2, w) in time $\mathcal{O}(k \cdot n' + n^c)$, where $k = |X_1 \cup X_2|$ and $n' = |V(G')|$.*

Proof. Let G'' be the graph obtained from G by identifying all vertices in X_1 (resp. X_2) to a new vertex x_1 (resp. x_2). Let $w' : V(G'') \rightarrow \mathbb{N}$ be such that $w'(x_1x_2) = \sum_{e \in E(G)} w(e) + 1$, $w'(x_iu) = \sum_{x \in X_i} w(xu)$ for $i \in [2]$ and $u \in N_G(X_i)$, and $w'(e) = w(e)$ otherwise. Let $(X_1^*, X_2^*) \in \mathcal{P}_2(V(G))$ be such that $(X_1, X_2) \subseteq (X_1^*, X_2^*)$. For $i \in [2]$, let $X_i' = X_i^* \setminus X_i$. Then

$$\begin{aligned} w(X_1^*, X_2^*) &= w(X_1, X_2) + w(X_1', X_2') + \sum_{xy \in E(X_1, X_2')} w(xy) + \sum_{xy \in E(X_1', X_2)} w(xy) \\ &= w(X_1, X_2) + w'(X_1', X_2') + \sum_{u \in X_2 \cap N_G(X_1)} w'(x_1u) + \sum_{u \in X_1 \cap N_G(X_2)} w'(x_2u) \\ &= w'(X_1' \cup \{x_1\}, X_2' \cup \{x_2\}) + w(X_1, X_2) - w(x_1x_2) \end{aligned}$$

Let \bar{w} be the constant $w(X_1, X_2) - w(x_1x_2)$. Hence, $f_{\text{cut}}(G, (X_1^*, X_2^*)) = f_{\text{cut}}(G'', (X_1' \cup \{x_1\}, X_2' \cup \{x_2\})) + \bar{w}$, and so $\hat{p}_{f_{\text{cut}}, \text{max}}(G, (X_1, X_2)) = \hat{p}_{f_{\text{cut}}, \text{max}}(G'', (\{x_1\}, \{x_2\})) + \bar{w}$. Furthermore, given that the weight of the edge x_1x_2 is larger than the sum of all other weights, x_1 and x_2 are never on the same side of a maximum cut in G'' . Hence, $\hat{p}_{f_{\text{cut}}, \text{max}}(G'', (\{x_1\}, \{x_2\})) = p_{f_{\text{cut}}, \text{max}}(G'')$, and therefore, $\hat{p}_{f_{\text{cut}}, \text{max}}(G, (X_1, X_2)) = p_{f_{\text{cut}}, \text{max}}(G'') + \bar{w}$.

Constructing G'' takes time $\mathcal{O}(k \cdot n)$ and computing \bar{w} takes time $\mathcal{O}(k^2)$. Since $\text{oct}(G'') = 2$, according to Proposition 6 and Lemma 7, an optimal solution to MAXIMUM WEIGHTED CUT on G'' can be found in time $\mathcal{O}(n'^c)$, and thus, an optimal solution to ANNOTATED MAXIMUM WEIGHTED CUT on (G, X_1, X_2) can be found in time $\mathcal{O}(k \cdot (k + n') + n^c)$. ◀

We apply Lemma 5 and Lemma 8 to the dynamic programming algorithm of Lemma 2 to obtain the following result.

► **Corollary 9.** *Given a graph G and a bipartite tree decomposition of G of width k , there is an algorithm that solves MAXIMUM WEIGHTED CUT on G in time $\mathcal{O}(2^k \cdot (k \cdot (k + n) + n^c))$.*

4 Further research

In this paper we study the complexity of several problems parameterized by bipartite treewidth, denoted by btw . In particular, our results extend the graph classes for which VERTEX COVER/INDEPENDENT SET, MAXIMUM WEIGHTED CUT, and ODD CYCLE TRANSVERSAL are polynomial-time solvable. A number of interesting questions remain open.

Except for 3-COLORING, all the problems we consider are covering and packing problems. We are still far from a full classification of the variants that are para-NP -complete, and those that are not (FPT or XP). For instance, concerning H -SUBGRAPH-COVER, we provided an FPT-algorithms when H is a clique. This case is particularly well-behaved because we know that in a tree decomposition every clique appears in some bag. On the other hand, as an immediate consequence of the result of Yannakakis [32], we know that H -SUBGRAPH-COVER is para-NP -complete for every bipartite graph H containing P_3 . We do not know what happens when H is not bipartite nor a clique. An apparently simple but challenging case is C_5 -SUBGRAPH-COVER. The main difficulty seems to be that C_5 -SUBGRAPH-COVER does not have the gluing property, which is the main ingredient in this paper to show that a problem is nice, and therefore to obtain an FPT-algorithm. We do not exclude the possibility that the problem is para-NP -complete, as we were not even able to obtain even an XP algorithm.

Concerning the packing problems, namely H -SUBGRAPH/INDUCED/SCATTERED/ODD-MINOR-PACKING, we provide XP-algorithms for them when H is non-bipartite. Unfortunately, we do not know whether any of them admits an FPT-algorithm, although we suspect that it is indeed the case. We would like to mention that it is possible to apply the framework of equivalence relations and representatives (see for instance [2, 9, 10]) to obtain an FPT-algorithm for K_t -SUBGRAPH-PACKING parameterized by btw . However, since a number of definitions and technical details are required to present this algorithm, we decided not to include it in this paper (which is already quite long). However, when H is not a clique, we do not know whether H -SUBGRAPH-PACKING admits an FPT-algorithm. A concrete case that we do not know how to solve is when H is the *paw*, i.e., the 4-vertex graph consisting of one triangle and one pendent edge.

Beyond bipartite tree decompositions, we introduce a more general type of decompositions that we call q (-torso)- \mathcal{H} -tree decompositions. For \mathcal{B} being the class of bipartite graphs, we prove that for every $q \geq 2$ and every 2-connected graph H with an edge, H -SCATTERED-PACKING is para-NP -complete parameterized by q (-torso)- \mathcal{B} -treewidth. It should be possible to prove similar results for other covering and packing problems considered in this article.

Most of our para-NP -completeness results consist just in proving NP-completeness on bipartite graph. There are two exceptions. On the one hand, the NP-completeness of 3-COLORING on graphs with odd cycle transversal at most three and H -SCATTERED-PACKING parameterized by q - \mathcal{B} -treewidth for every integer $q \geq 2$. Interestingly, none of our hardness results really exploits the structure of bipartite tree decompositions (i.e., for $q = 1$), beyond being bipartite or having bounded odd cycle transversal.

Finally, as mentioned in the introduction, the goal of this article is to make a first step toward efficient algorithms to solve problems related to odd-minors. We already show in this paper that bipartite treewidth can be useful in this direction, by providing an XP-algorithm for H -ODD-MINOR-PACKING. Bipartite treewidth, or strongly related notions, also plays a strong role in the recent series of papers about odd-minors by Campbell, Gollin, Hendrey, and Wiederrecht [4, 12]. This looks like an emerging topic that is worth investigating.

References

- 1 Akanksha Agrawal, Lawqueen Kanesh, Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Deleting, eliminating and decomposing to hereditary classes are all fpt-equivalent. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1976–2004. SIAM, 2022. doi:10.1137/1.9781611977073.79.
- 2 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. A complexity dichotomy for hitting connected minors on bounded treewidth graphs: the chair and the banner draw the boundary. In *Proc. of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 951–970, 2020. doi:10.1137/1.9781611975994.57.
- 3 Kathie Cameron. Induced matchings. *Discrete Applied Mathematics*, 24(1-3):97–102, 1989. doi:10.1016/0166-218X(92)90275-F.
- 4 Rutger Campbell, J. Pascal Gollin, Kevin Hendrey, and Sebastian Wiederrecht. Odd-Minors II: Bipartite treewidth. Manuscript under preparation (private communication), 2023.
- 5 Julia Chuzhoy and Zihan Tan. Towards tight(er) bounds for the Excluded Grid Theorem. *Journal of Combinatorial Theory, Series B*, 146:219–265, 2021. doi:10.1016/j.jctb.2020.09.010.
- 6 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Decomposition, approximation, and coloring of odd-minor-free graphs. In *Proc. of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 329–344. SIAM, 2010. doi:10.1137/1.9781611973075.28.
- 7 Reinhard Diestel. *Graph Theory*, volume 173. Springer-Verlag, 5th edition, 2017. doi:10.1007/978-3-662-53622-3.
- 8 Eduard Eiben, Robert Ganian, Thekla Hamm, and O-joung Kwon. Measuring what matters: A hybrid approach to dynamic programming with treewidth. *Journal of Computer and System Sciences*, 121:57–75, 2021. doi:10.1016/j.jcss.2021.04.005.
- 9 Valentin Garnero, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. Explicit linear kernels via dynamic programming. *SIAM Journal on Discrete Mathematics*, 29(4):1864–1894, 2015. doi:10.1137/140968975.
- 10 Valentin Garnero, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. Explicit linear kernels for packing problems. *Algorithmica*, 81(4):1615–1656, 2019. doi:10.1007/s00453-018-0495-5.
- 11 Jim Geelen, Bert Gerards, Bruce A. Reed, Paul D. Seymour, and Adrian Vetta. On the odd-minor variant of Hadwiger’s conjecture. *Journal of Combinatorial Theory, Series B*, 99(1):20–29, 2009. doi:10.1016/j.jctb.2008.03.006.
- 12 J. Pascal Gollin and Sebastian Wiederrecht. Odd-Minors I: Excluding small parity breaks. *CoRR*, abs/2304.04504, 2023. arXiv:2304.04504.
- 13 Martin Grötschel and William R. Pulleyblank. Weakly bipartite graphs and the max-cut problem. *Operations Research Letters*, 1(1):23–27, 1981. doi:10.1016/0167-6377(81)90020-1.
- 14 Bertrand Guenin. A characterization of weakly bipartite graphs. *Journal of Combinatorial Theory, Series B*, 83(1):112–168, 2001. doi:10.1006/jctb.2001.2051.
- 15 Hugo Hadwiger. Über eine klassifikation der streckenkomplexe. *Vierteljschr. Naturforsch. Ges. Zürich*, 88(2):133–142, 1943. URL: https://www.ngzh.ch/archiv/1943_88/88_2/88_17.pdf.
- 16 Huynh, Tony. *The Linkage Problem for Group-labelled Graphs*. PhD thesis, University of Waterloo, 2009. URL: <http://hdl.handle.net/10012/4716>.
- 17 Lars Jaffke, Laure Morelle, Ignasi Sau, and Dimitrios M. Thilikos. Dynamic programming on bipartite tree decompositions. *CoRR*, abs/2309.07754, 2023. doi:10.48550/arXiv.2309.07754.
- 18 Bart M. P. Jansen and Jari J. H. de Kroon. FPT algorithms to compute the elimination distance to bipartite graphs and more. In *Proc. of the 47th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 12911 of *LNCS*, pages 80–93, 2021. doi:10.1007/978-3-030-86838-3_6.

- 19 Tommy R Jensen and Bjarne Toft. *Graph coloring problems*. Wiley, 2011. doi:10.1002/9781118032497.
- 20 Richard M. Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 219–241. Springer, 2010. doi:10.1007/978-3-540-68279-0_8.
- 21 Ken-ichi Kawarabayashi and Bruce A. Reed. An (almost) linear time algorithm for odd cycles transversal. In *Proc. of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 365–378. SIAM, 2010. doi:10.1137/1.9781611973075.31.
- 22 Ken-ichi Kawarabayashi, Bruce A. Reed, and Paul Wollan. The graph minor algorithm with parity conditions. In *Proc. of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 27–36. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.52.
- 23 Valerie King, S. Rao, and Robert Endre Tarjan. A faster deterministic maximum flow algorithm. *Journal of Algorithms*, 17(3):447–474, 1994. doi:10.1006/jagm.1994.1044.
- 24 David G. Kirkpatrick and Pavol Hell. On the complexity of general graph factor problems. *SIAM Journal on Computing*, 12(3):601–609, 1983. doi:10.1137/0212040.
- 25 Daniel Lokshantov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Transactions on Algorithms*, 11(2):15:1–15:31, 2014. doi:10.1145/2566616.
- 26 Jérôme Monnot and Sophie Toulouse. The path partition problem and related problems in bipartite graphs. *Operations Research Letter*, 35(5):677–684, 2007. doi:10.1016/j.orl.2006.12.004.
- 27 James B. Orlin. Max flows in $O(nm)$ time, or better. In *Proc. of the 45th annual ACM Symposium on Theory of Computing Conference (STOC)*, pages 765–774. ACM, 2013. doi:10.1145/2488608.2488705.
- 28 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004. doi:10.1016/j.orl.2003.10.009.
- 29 Neil Robertson, Paul D. Seymour, and Robin Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 62(2):323–348, 1994. doi:10.1006/jctb.1994.1073.
- 30 Raphael Steiner. Improved bound for improper colourings of graphs with no odd clique minor. *Combinatorics, Probability and Computing*, 32(2):326–333, 2023. doi:10.1017/S0963548322000268.
- 31 Siamak Tazari. Faster approximation schemes and parameterized algorithms on (odd)- h -minor-free graphs. *Theoretical Computer Science*, 417:95–107, 2012. doi:10.1016/j.tcs.2011.09.014.
- 32 Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM Journal on Computing*, 10(2):310–327, 1981. doi:10.1137/0210022.

A Graphs, treewidth, bounded graphs, and nice problems

Functions. Given two sets A and B , and two functions $f, g : A \rightarrow 2^B$, we denote by $f \cup g$ the function that maps $x \in A$ to $f(x) \cup g(x) \in 2^B$. Let $f : A \rightarrow B$ be an injection. Let $K \subseteq B$ be the image of f . By convention, if f is referred to as a bijection, it means that we consider that f maps A to K . Given a function $w : A \rightarrow \mathbb{N}$, and $A' \subseteq A$, $w(A') = \sum_{x \in A'} w(x)$.

Basic concepts on graphs. All graphs considered in this paper are undirected, finite, and without loops or multiple edges. We use standard graph-theoretic notation and we refer the reader to [7] for any undefined terminology. For convenience, we use uv instead of $\{u, v\}$ to denote an edge of a graph. Let G be a graph. In the rest of this paper we always use n for the cardinality of $V(G)$, and m for the cardinality of $E(G)$, where G is the input

graph of the problem under consideration. For $S \subseteq V(G)$, we set $G[S] = (S, E \cap \binom{S}{2})$ and use the shortcut $G \setminus S$ to denote $G[V(G) \setminus S]$. Given a vertex $v \in V(G)$, we denote by $N_G(v)$ the set of vertices of G that are adjacent to v in G . Moreover, given a set $A \subseteq V(G)$, $N_G(A) = \bigcup_{v \in A} N_G(v) \setminus A$. For $k \in \mathbb{N}$, we denote by P_k the path with k vertices, and we say that P_k has length $k - 1$ (i.e., the *length* of a path is its number of edges). We denote by $\text{cc}(G)$ the set of connected components of a graph G . For $A, B \subseteq V(G)$, $E(A, B)$ denotes the set of edges of G with one endpoint in A and the other in B . We say that $E' \subseteq E(G)$ is an *edge cut* of G if there is a partition (A, B) of $V(G)$ such that $E' = E(A, B)$. We say that a pair $(L, R) \in 2^{V(G)} \times 2^{V(G)}$ is a *separation* of G if $L \cup R = V(G)$ and $E(L \setminus R, R \setminus L) = \emptyset$. The *order* of (L, R) is $|L \cap R|$. $L \cap R$ is called a $|L \cap R|$ -*separator* of G . A graph G is k -*connected* if, for any separation (L, R) of G of order at most $k - 1$, either $L \subseteq R$ or $R \subseteq L$. A graph class \mathcal{H} is *hereditary* if for any $G \in \mathcal{H}$ and $v \in V(G)$, $G \setminus \{v\} \in \mathcal{H}$.

Treewidth. A *tree decomposition* of a graph G is a pair (T, χ) where T is a tree and $\chi : V(T) \rightarrow 2^{V(G)}$ such that

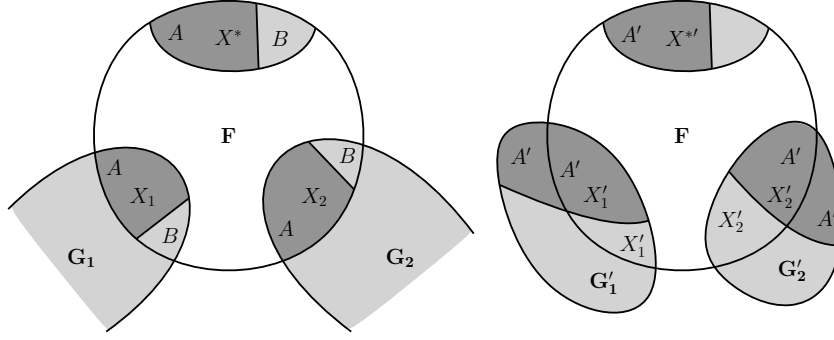
- $\bigcup_{t \in V(T)} \chi(t) = V(G)$,
- for every $e \in E(G)$, there is a $t \in V(T)$ such that $\chi(t)$ contains both endpoints of e , and
- for every $v \in V(G)$, the subgraph of T induced by $\{t \in V(T) \mid v \in \chi(t)\}$ is connected.

The *width* of (T, χ) is equal to $\max \{ |\chi(t)| - 1 \mid t \in V(T) \}$ and the *treewidth* of G , denoted by $\text{tw}(G)$, is the minimum width over all tree decompositions of G .

For every node $t \in V(T)$, $\chi(t)$ is called *bag* of t . Given $tt' \in E(T)$, the *adhesion* of t and t' , denoted by $\text{adh}(t, t')$, is the set $\chi(t) \cap \chi(t')$.

A *rooted tree decomposition* is a triple (T, χ, r) where (T, χ) is a tree decomposition and (T, r) is a *rooted tree* (i.e., T is a tree and $r \in V(T)$). Given $t \in V(T)$, we denote by $\text{ch}_r(t)$ the set of children of t and by $\text{par}_r(t)$ the parent of t (if $t \neq r$). We set $\delta_t^r = \text{adh}(t, \text{par}_r(t))$, with the convention that $\delta_r^r = \emptyset$. Moreover, we denote by G_t^r the graph induced by $\bigcup_{t' \in V(T_t)} \chi(t')$ where (T_t, t) is the rooted subtree of (T, r) . We may use δ_t and G_t instead of δ_t^r and G_t^r when there is no risk of confusion.

Boundaried graphs. Let $t \in \mathbb{N}$. A t -*boundaried graph* is a triple $\mathbf{G} = (G, B, \rho)$ where G is a graph, $B \subseteq V(G)$, $|B| = t$, and $\rho : B \rightarrow \mathbb{N}$ is an injection. We say that B is the *boundary* of \mathbf{G} and we write $B = \text{bd}(\mathbf{G})$. We call \mathbf{G} *trivial* if all its vertices belong to the boundary. We say that two t -boundaried graphs $\mathbf{G}_1 = (G_1, B_1, \rho_1)$ and $\mathbf{G}_2 = (G_2, B_2, \rho_2)$ are *isomorphic* if $\rho_1(B_1) = \rho_2(B_2)$ and there is an isomorphism from G_1 to G_2 that extends the bijection $\rho_2^{-1} \circ \rho_1$. A triple (G, B, ρ) is a *boundaried graph* if it is a t -boundaried graph for some $t \in \mathbb{N}$. We denote by \mathcal{B}^t the set of all (pairwise non-isomorphic) t -boundaried graphs. A boundaried graph \mathbf{F} is a *boundaried induced subgraph* (resp. *boundaried subgraph*) of \mathbf{G} if \mathbf{F} can be obtained from \mathbf{G} by removing vertices (resp. and edges). A boundaried graph \mathbf{F} is a *boundaried odd-minor* of \mathbf{G} if \mathbf{F} can be obtained from a boundaried subgraph \mathbf{G}' of \mathbf{G} by contracting an edge cut such that every vertex in $\text{bd}(\mathbf{G}')$ is on the same side of the cut. We say that two boundaried graphs $\mathbf{G}_1 = (G_1, B_1, \rho_1)$ and $\mathbf{G}_2 = (G_2, B_2, \rho_2)$ are *compatible* if $\rho_1(B_1) = \rho_2(B_2)$ and $\rho_2^{-1} \circ \rho_1$ is an isomorphism from $G_1[B_1]$ to $G_2[B_2]$. Given two boundaried graphs $\mathbf{G}_1 = (G_1, B_1, \rho_1)$ and $\mathbf{G}_2 = (G_2, B_2, \rho_2)$, we define $\mathbf{G}_1 \oplus \mathbf{G}_2$ as the unboundaried graph obtained if we take the disjoint union of G_1 and G_2 and, for every $i \in \rho_1(B_1) \cap \rho_2(B_2)$, we identify vertices $\rho_1^{-1}(i)$ and $\rho_2^{-1}(i)$. If v is the result of the identification of $v_1 := \rho_1^{-1}(i)$ and $v_2 := \rho_2^{-1}(i)$ then we say that v is the *heir* of v_i from \mathbf{G}_i , $i \in [2]$. If v is either a vertex of G_1 where $\rho_1(v) \notin \rho_1(B_1) \cap \rho_2(B_2)$ (if $v \in B_1$) or a vertex of G_2 where $\rho_2(v) \notin \rho_1(B_1) \cap \rho_2(B_2)$ (if $v \in B_2$), then v is also a (non-identified) vertex of



■ **Figure 1** Illustration of the setting of the nice problem and reduction. The shaded area on the left is \mathbf{G} where $X = X_1 \cup X_2 \cup X^*$, and the shaded area on the right is \mathbf{G}' where $X' = X'_1 \cup X'_2 \cup X^{*'}$.

$\mathbf{G}_1 \oplus \mathbf{G}_2$ and is a *heir* of itself (from \mathbf{G}_1 or \mathbf{G}_2 respectively). For $i \in [2]$, and given an edge vu in $\mathbf{G}_1 \oplus \mathbf{G}_2$, we say that vu is the *heir* of an edge $v'u'$ from \mathbf{G}_i if v' (resp. u') is the heir of v (resp. u) from \mathbf{G}_i and $v'u'$ is an edge of G_i . If x' is an heir of x from $\mathbf{G} = (G, B, \rho)$ in \mathbf{G}' , then we write $x = \text{heir}_{\mathbf{G}, \mathbf{G}'}(x')$. If $B' \subseteq B$, then $\text{heir}_{\mathbf{G}, \mathbf{G}'}(B) = \bigcup_{v \in B'} \text{heir}_{\mathbf{G}, \mathbf{G}'}(x')$. We also define $\mathbf{G}_1 \boxplus \mathbf{G}_2$ as the *boundaried graph* $(\mathbf{G}_1 \oplus \mathbf{G}_2, B, \rho)$, where B is the sets of all heirs from \mathbf{G}_1 and \mathbf{G}_2 and $\rho : B \rightarrow \mathbb{N}$ is the union of ρ_1 and ρ_2 after identification. Note that in circumstances where \boxplus is repetitively applied, the heir relation is maintained due to its transitivity. Moreover, we define $\mathbf{G}_1 \triangleright \mathbf{G}_2$ as the unboundaried graph G obtained from $\mathbf{G}_1 \oplus \mathbf{G}_2$ by removing all heirs from \mathbf{G}_2 that are not heirs from \mathbf{G}_1 and all heirs of edges from \mathbf{G}_2 that are not heirs of edges from \mathbf{G}_1 . Note that \triangleright is not commutative. For the sake of simplicity, with a slight abuse of notation, we sometimes identify a vertex with its heir.

Nice problem and nice reduction. Let $p \in \mathbb{N}$, let \mathcal{H} be a graph class, and let Π be a p -annotated problem corresponding to some choice of p -partition-evaluation function f and some $\text{opt} \in \{\max, \min\}$. We say that Π is a \mathcal{H} -*nice problem* if there exists an algorithm that receives as input

- a boundaried graph $\mathbf{G} = (G, X, \rho)$,
 - a trivial boundaried graph $\mathbf{X} = (G[X], X, \rho_X)$ and a collection $\{\mathbf{G}_i = (G_i, X_i, \rho_i) \mid i \in [d]\}$ of boundaried graphs, such that $d \in \mathbb{N}$ and $\mathbf{G} = \mathbf{X} \boxplus (\boxplus_{i \in [d]} \mathbf{G}_i)$,
 - a partition (A, B) of X such that for all $i \in [d]$, $|\text{heir}_{\mathbf{G}_i, \mathbf{G}}(X_i) \setminus A| \leq 1$,
 - some $\mathcal{A} \in \mathcal{P}_p(A)$, and
 - for every $i \in [d]$ and each $\mathcal{X}_i \in \mathcal{P}_p(X_i)$, the value $\hat{p}_{f, \text{opt}}(G_i, \mathcal{X}_i)$,
- and outputs, in time $\mathcal{O}(|A| \cdot d)$, a tuple $(\mathbf{G}' = (G', X', \rho'), \mathcal{A}', s')$, called \mathcal{H} -*nice reduction of the pair $(\mathbf{G}, \mathcal{A})$ with respect to Π* , such that the following hold.
- There is a set $A' \subseteq V(G')$ such that $|A'| = |A| + \mathcal{O}(1)$, and $\mathcal{A}' \in \mathcal{P}_p(A')$.
 - There is a trivial boundaried graph $\mathbf{X}' = (G[X'], X', \rho_{X'})$ and a collection $\{\mathbf{G}'_i = (G'_i, X'_i, \rho'_i) \mid i \in [d']\}$, where $d' \in \mathbb{N}$, of boundaried graphs such that $\mathbf{G}' = \mathbf{X}' \boxplus (\boxplus_{i \in [d']} \mathbf{G}'_i)$ and $|V(G')| \leq |X| + \mathcal{O}(|B|)$, $|E(G')| \leq |E(G[X])| + \mathcal{O}(|B|)$.
 - For any boundaried graph \mathbf{F} compatible with \mathbf{G} , it holds that

$$\hat{p}_{f, \text{opt}}(\mathbf{G} \oplus \mathbf{F}, \mathcal{A}) = \hat{p}_{f, \text{opt}}(\mathbf{G}' \triangleright \mathbf{F}, \mathcal{A}') + s'.$$
 - For any boundaried graph $\mathbf{F} = (F, X_F, \rho_F)$ compatible with \mathbf{G} , if $\bar{F} \setminus A_F \in \mathcal{H}$, where $\bar{F} = (\mathbf{F} \oplus \mathbf{G})[\text{heir}_{\mathbf{F}, \mathbf{G} \oplus \mathbf{F}}(V(F))]$ and $A_F = \text{heir}_{\mathbf{G}, \mathbf{G} \oplus \mathbf{F}}(A)$, then $(\mathbf{G}' \triangleright \mathbf{F}) \setminus A' \in \mathcal{H}$.

See Figure 1 for an illustration.

B Definition of the problems and their annotated extensions

K_t -Subgraph-Cover. Let \mathcal{G} be a graph class. We define the problem VERTEX DELETION TO \mathcal{G} as follows.

(WEIGHTED) VERTEX DELETION TO \mathcal{G}

Input: A graph G (and a weight function $w : V(G) \rightarrow \mathbb{N}$).

Objective: Find the set $S \subseteq V(G)$ of minimum size (resp. weight) such that $G \setminus S \in \mathcal{G}$.

If \mathcal{G} is the class of edgeless (resp. acyclic, planar, bipartite, (proper) interval, chordal) graphs, then we obtain the VERTEX COVER (resp. FEEDBACK VERTEX SET, VERTEX PLANARIZATION, ODD CYCLE TRANSVERSAL, (PROPER) INTERVAL VERTEX DELETION, CHORDAL VERTEX DELETION) problem. Also, given a graph H , if \mathcal{G} is the class of graphs that do not contain H as a subgraph (resp. a minor/odd-minor/induced subgraph), then the corresponding problem is called H -SUBGRAPH-COVER (resp. H -MINOR-COVER/ H -ODD-MINOR-COVER/ H -INDUCED-SUBGRAPH-COVER).

Let H be a graph and $w : V(G) \rightarrow \mathbb{N}$ be a weight function (constant equal to one in the unweighted case). We define f_H as the 2-partition-evaluation function where, for every graph G , for every $(R, S) \in \mathcal{P}_2(V(G))$,

$$f_H(G, (R, S)) = \begin{cases} +\infty & \text{if } H \text{ is a subgraph of } G \setminus S, \\ w(S) & \text{otherwise.} \end{cases}$$

Seen as an optimization problem, (WEIGHTED) H -SUBGRAPH-COVER is the problem of computing $\mathfrak{p}_{f_H, \min}(G)$. We call its annotated extension (WEIGHTED) ANNOTATED H -SUBGRAPH-COVER. In other words, (WEIGHTED) ANNOTATED H -SUBGRAPH-COVER is defined as follows.

(WEIGHTED) ANNOTATED H -SUBGRAPH-COVER

Input: A graph G , two disjoint sets $R, S \subseteq V(G)$ (and a weight function $w : V(G) \rightarrow \mathbb{N}$).

Objective: Find, if it exists, the minimum size (resp. weight) of a set $S^* \subseteq V(G)$ such that $R \cap S^* = \emptyset$, $S \subseteq S^*$, and $G \setminus S^*$ does not contain H as a subgraph.

Odd Cycle Transversal. Let H be a graph. We define f_{oct} as the 3-partition-evaluation function where, for every graph G and for every $(S, X_1, X_2) \in \mathcal{P}_3(V(G))$,

$$f_{\text{oct}}(G, (S, X_1, X_2)) = \begin{cases} |S| & \text{if } G \setminus S \in \mathcal{B}, \text{ witnessed by the bipartition } (X_1, X_2), \\ +\infty & \text{otherwise.} \end{cases}$$

Hence, seen as an optimization problem, ODD CYCLE TRANSVERSAL is the problem of computing $\mathfrak{p}_{f_{\text{oct}}, \min}(G)$. We call its annotated extension ANNOTATED ODD CYCLE TRANSVERSAL. In other words, ANNOTATED ODD CYCLE TRANSVERSAL is defined as follows.

(WEIGHTED) ANNOTATED ODD CYCLE TRANSVERSAL

Input: A graph G , three disjoint sets $S, X_1, X_2 \subseteq V(G)$ (and a weight function $w : V(G) \rightarrow \mathbb{N}$).

Objective: Find, if it exists, a set S^* of minimum size (resp. weight) such that $S \subseteq S^*$, $(X_1 \cup X_2) \cap S^* = \emptyset$, and $G \setminus S^*$ is bipartite with X_1 and X_2 on different sides of the bipartition.

26:22 Dynamic Programming on Bipartite Tree Decompositions

Packing. Let \mathcal{G} be a graph class. We define the \mathcal{G} -PACKING problem as follows.

\mathcal{G} -PACKING

Input: A graph G .

Objective: Find the maximum number k of pairwise-disjoint subgraphs H_1, \dots, H_k such that, for each $i \in [k]$, $H_i \in \mathcal{G}$.


Let H be a graph. If $\mathcal{G} = \{H\}$ (resp. \mathcal{G} is the class of all graphs containing H as a minor/odd-minor/induced subgraph), then we refer to the corresponding problem as H -SUBGRAPH-PACKING (resp. H -MINOR-PACKING/ H -ODD-MINOR-PACKING/ H -INDUCED-SUBGRAPH-PACKING). Note, in particular, that K_3 -ODD-MINOR-PACKING is exactly ODD CYCLE PACKING.

If in the definition of \mathcal{G} -PACKING we add the condition that there is no edge in the input graph between vertices of different H_i 's, then we refer to the corresponding problem as H -SCATTERED-PACKING, where we implicitly assume that we refer to the subgraph relation, and where we do *not* specify a degree of “scatteredness”, as it is usual in the literature when dealing, for instance, with the scattered version of INDEPENDENT SET. For instance, K_2 -SCATTERED-PACKING is exactly INDUCED MATCHING.

Kernelization for Counting Problems on Graphs: Preserving the Number of Minimum Solutions

Bart M. P. Jansen  

Eindhoven University of Technology, The Netherlands

Bart van der Steenhoven  

Eindhoven University of Technology, The Netherlands

Abstract

A kernelization for a parameterized decision problem \mathcal{Q} is a polynomial-time preprocessing algorithm that reduces any parameterized instance (x, k) into an instance (x', k') whose size is bounded by a function of k alone and which has the same YES/NO answer for \mathcal{Q} . Such preprocessing algorithms cannot exist in the context of counting problems, when the answer to be preserved is the number of solutions, since this number can be arbitrarily large compared to k . However, we show that for counting minimum feedback vertex sets of size at most k , and for counting minimum dominating sets of size at most k in a planar graph, there is a polynomial-time algorithm that either outputs the answer or reduces to an instance (G', k') of size polynomial in k with the same number of minimum solutions. This shows that a meaningful theory of kernelization for counting problems is possible and opens the door for future developments. Our algorithms exploit that if the number of solutions exceeds $2^{\text{poly}(k)}$, the size of the input is exponential in terms of k so that the running time of a parameterized counting algorithm can be bounded by $\text{poly}(n)$. Otherwise, we can use gadgets that slightly increase k to represent choices among $2^{\mathcal{O}(k)}$ options by only $\text{poly}(k)$ vertices.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Graph algorithms analysis; Mathematics of computing \rightarrow Graph algorithms

Keywords and phrases kernelization, counting problems, feedback vertex set, dominating set, protrusion decomposition

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.27

Related Version *Full Version:* <http://arxiv.org/abs/2310.04303> [16]

Funding *Bart M. P. Jansen:* Funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803421, Reduce-Search).

Bart van der Steenhoven: Supported by Project No. ICT22-029 of the Vienna Science Foundation (WWTF).



1 Introduction

Background and motivation. Counting problems, whose answer is an integer giving the number of objects of a certain kind rather than merely YES or NO, have important applications in fields of research such as artificial intelligence [24, 25], statistical physics [17, 20, 31] and network science [22]. They have been studied extensively in classical complexity, underpinning fundamental results such as Toda's theorem [29] and the $\#P$ -completeness of the permanent [30]. A substantial research effort has targeted the parameterized complexity of counting problems, leading to parametric complexity-notions like $\#W[1]$ -hardness [7, 11, 21] and FPT algorithms to solve several counting problems. For example, FPT algorithms were developed to count the number of size- k vertex covers [10], or the number of occurrences of a size- k pattern graph H in a host graph G [8].



© Bart M. P. Jansen and Bart van der Steenhoven;
licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 27; pp. 27:1–27:15

Leibniz International Proceedings in Informatics



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

This paper is concerned with an aspect of parameterized algorithms which has been largely neglected for counting problems: that of efficient preprocessing with performance guarantees, i.e., kernelization. A kernelization for a parameterized decision problem \mathcal{Q} is a polynomial-time preprocessing algorithm that reduces any parameterized instance (x, k) into an instance (x', k') whose size is bounded by a function of k alone and which has the same YES/NO answer for \mathcal{Q} . Over the last decade, kernelization has developed into an important subfield of parameterized algorithms, as documented in a textbook dedicated to the topic [12]. Given the success of kernelization for decision problems, one may wonder: can a theory of provably-efficient preprocessing for counting problems be developed?

Consider a prototypical problem such as FEEDBACK VERTEX SET in undirected graphs, in which the goal is to find a small vertex set whose removal breaks all cycles. What could be an appropriate notion of counting kernelization for such a problem? The concept of efficient preprocessing towards a provably small instance with the same answer could be instantiated as follows: given a pair (G, k) , the preprocessing algorithm should output a pair (G', k') whose size is bounded by a function of k such that the number of size- k feedback vertex sets in G is equal to the number of size- k' feedback vertex sets in G' . However, this task is clearly impossible. Given a graph consisting of a length- n cycle with parameter $k = 1$, the number of solutions is n which can be arbitrarily large compared to k , while for any reduced instance (G', k') of size bounded in k , the number of solutions can be at most $2^{|V(G')|} \leq f(k)$. Without allowing the size of the reduced instance to depend on n , it seems that preprocessing while preserving the answer to the counting problem is impossible.

Over the years, there have been two approaches to deal with this obstacle¹. Thurley [28], inspired by concepts in earlier work [23], proposed a notion of counting kernelization which effectively reduces a counting problem to an enumeration problem. He considered problems such as VERTEX COVER and d -HITTING SET. In his framework, the preprocessing algorithm has to output an instance of size bounded by a function of k , in such a way that for any solution to the reduced instance, we can efficiently determine to how many solutions of the original instance it corresponds. Hence by enumerating *all* solutions on the reduced instance, we can obtain the number of solutions to the original instance. A significant drawback of this approach therefore lies in the fact that to solve the counting problem on the original instance, we have to enumerate all solutions on the reduced instance. Since counting can potentially be done much faster than enumeration, it is not clear that this preprocessing step is always beneficial.

A second notion for counting kernelization was proposed by Kim, Selma, and Thilikos [18, 26]. Their framework (which also applies to FEEDBACK VERTEX SET) considers two types of algorithms: a *condenser* that maps an input instance (G, k) to an instance (G', k') of an auxiliary *annotated* problem involving weights on the vertices of G' , and an *extractor* that recovers (typically not in polynomial time) the number of solutions to (G, k) from the weighted instance (G', k') . The number of vertices of G' is required to be bounded in k , but the weights are allowed to be arbitrarily large, thereby sidestepping the issue described above. This means that in terms of the total encoding size, the weighted graph (G', k') is not guaranteed to be smaller than (G, k) and in general the total number of bits needed to encode the weighted graph cannot be bounded by a function of k alone. The condenser-extractor

¹ A third [19] approach was announced shortly before this paper went to print; it allows a polynomial-time lifting step to compute the number of solutions to the original instance from the number of solutions to the reduced instance.

framework has the same drawback as the framework by Thurley: a standard counting problem is reduced to a more complicated type of problem, in this case one involving weights and annotations.

The goal of this paper is to show that there is an alternative way to overcome the obstacle for counting kernelization, which leads to a notion of preprocessing in which the problem to be solved on the reduced instance is of exactly the same nature as the original. Our solution is inspired by the typical behavior of kernelization algorithms for decision problems: we formalize the option of already finding the answer during the preprocessing phase. Note that many algorithms, such as the famous Buss [5] kernelization for VERTEX COVER, work by applying reduction rules to arrive at the reduced instance, or discover the YES/NO answer to the decision problem during preprocessing. Our kernelization algorithms for counting problems will have the same behavior: they will either reduce to a $\text{poly}(k)$ -sized instance of the same problem that has exactly the same answer to the counting problem, or they outright answer the counting problem during their polynomial-time computation. To our initial surprise, such preprocessing algorithms exist for several classic problems.

Our results. To begin the exploration of this new type of counting kernelization, we revisit two prominent graph problems: FEEDBACK VERTEX SET in general undirected graphs and DOMINATING SET in planar graphs. The decision versions of these problems (does graph G have a solution of size at most k ?) have kernels with $\mathcal{O}(k^2)$ [15, 27] and $\mathcal{O}(k)$ vertices [1, 4, 14], respectively. We consider the problem of counting the number of *minimum-size* solutions, parameterized by the size k of a minimum solution. (We discuss counting inclusion-minimal solutions in the conclusion.) For a graph G and integer k , we denote by $\#\text{minFVS}(G, k)$ the number of minimum feedback vertex sets in G of size at most k in G . Hence $\#\text{minFVS}(G, k)$ is equal to 0 if the feedback vertex number of G exceeds k , and otherwise is equal to the number of minimum solutions. The analogous concept for minimum dominating sets is denoted $\#\text{minDS}(G, k)$. Our result for FEEDBACK VERTEX SET reads as follows.

► **Theorem 1.1.** *There is a polynomial-time algorithm that, given a graph G and integer k , either*

- *outputs $\#\text{minFVS}(G, k)$, or*
- *outputs a graph G' and integer k' such that $\#\text{minFVS}(G, k) = \#\text{minFVS}(G', k')$ and $|V(G')| = \mathcal{O}(k^5)$ and $k' = \mathcal{O}(k^5)$.*

For DOMINATING SET on planar graphs, we give an analogous algorithm that either outputs $\#\text{minDS}(G, k)$ or reduces to a *planar* instance (G', k') with $|V(G')|, k' = \mathcal{O}(k^3)$ such that $\#\text{minDS}(G, k) = \#\text{minDS}(G', k')$. Hence if the parameter is small, the task of counting the number of minimum solutions can efficiently be reduced to the *same* counting task on a provably small instance.

► **Theorem 1.2 (★).** *There is a polynomial-time algorithm that, given a planar graph G and integer k , either*

- *outputs $\#\text{minDS}(G, k)$, or*
- *outputs a planar graph G' and integer k' such that $\#\text{minDS}(G, k) = \#\text{minDS}(G', k')$ and $|V(G')| = \mathcal{O}(k^3)$ and $k' = \mathcal{O}(k^3)$.*

The high-level approach is the same for both problems. We use insights from existing kernels for the decision version of the problem to reduce an input instance (G, k) into one (G', k') with the same number of minimum solutions, such that G' can be decomposed into a “small” core together with $\text{poly}(k)$ “simply structured but potentially large” parts. For

DOMINATING SET, this takes the form of a protrusion decomposition; for FEEDBACK VERTEX SET the decomposition is more elementary. Then we consider two cases. If $|V(G)| > 2^k$, we employ an FPT algorithm running in time $2^{\mathcal{O}(k)} \cdot \text{poly}(n)$ to count the number of minimum solutions and output it. Since $n > 2^k$, this step runs in polynomial time. If $|V(G)| \leq 2^k$, then we show that each of the $\text{poly}(k)$ simply structured parts can be replaced with a gadget of size $\text{poly}(k)$ without affecting the number of minimum solutions. In this step, we typically increase the size of minimum solutions slightly to allow a small vertex set to encode exponentially many potential solutions. For example, an instance of FEEDBACK VERTEX SET consisting of a cycle of length 2^{10} (which has 2^{10} different optimal solutions), can be reduced to the graph consisting of 10 pairs (a_i, b_i) , each pair connected by two parallel edges. The latter graph also has 2^{10} minimum solutions, each of size 10. To carry out this approach, the most technical part is to show how to decompose the input instance into parts in which it is easy to analyze how many different choices an optimal solution can make.

Organization. The remainder of the paper is structured as follows. After presenting preliminaries in Section 2, we illustrate our approach for FEEDBACK VERTEX SET in Section 3. The more technical application to DOMINATING SET on planar graphs is deferred to the full version [16] due to space limitations. We conclude in Section 4 with a reflection on the potential of this approach to counting kernelization.

2 Preliminaries

All graphs we consider are undirected; they may have parallel edges but no self-loops. A graph G therefore consists of a set $V(G)$ of vertices and a multiset $E(G)$ of edges of the form $\{u, v\}$ for distinct $u, v \in V(G)$. For a vertex $v \in V(G)$, we refer to the *open neighborhood* of v in G as $N_G(v)$ and to the *closed neighborhood* of v as $N_G[v]$. For a set of vertices $X \subseteq V(G)$, the open and closed neighborhoods are defined as $N_G(X) = (\bigcup_{v \in X} N_G(v)) \setminus X$ and $N_G[X] = \bigcup_{v \in X} N_G[v]$. The degree of vertex v in graph G , denoted by $\deg_G(v)$, is equal to the number of edges incident to v in G . We refer to the subgraph of G induced by a vertex set $X \subseteq V(G)$ as $G[X]$. We use $G - X$ as a way to write $G[V(G) \setminus X]$ and $G - v$ as a shorthand for $G - \{v\}$. A graph H is a *minor* of G if H can be formed by contracting edges of a subgraph of G .

A *feedback vertex set* of a graph G is a set $S \subseteq V(G)$ such that $G - S$ is a forest, i.e., acyclic. The *feedback vertex number* of a graph is the size of a smallest feedback vertex set of that graph. A *dominating set* of a graph G is a set $D \subseteq V(G)$ such that $N_G[D] = V(G)$. The *domination number* of a graph is the size of a smallest dominating set of that graph. We say that a set $X \subseteq V(G)$ dominates $U \subseteq V(G)$ if $U \subseteq N_G[X]$.

We define $V_{\neq 2}(G)$ to be the set of vertices of graph G that do not have degree two. We refer to a *chain* C of G as a connected component of $G - V_{\neq 2}(G)$. We say that a chain C is a *proper chain* if $N_G(C) \neq \emptyset$ and we then refer to $N_G(C)$ as the *endpoints* of C .

3 Counting feedback vertex sets

In this section, we explain the technique that allows us to either count the number of minimum feedback vertex sets of a graph G in polynomial time, or reduce G to a provably small instance with the same number of minimum solutions. We start by showing that, by using a few reduction rules, we can already reduce G to an equivalent instance with a specific structure. This reduction is based on the $\mathcal{O}(k^3)$ -vertex kernel for the decision

FEEDBACK VERTEX SET problem presented by Jansen [3]. We choose to use this kernel over the better-known and smaller-size kernels by Thomassé [27] and Iwata [15] because those rely on multiple reduction rules that are not safe for counting minimum solutions.

As is common for FEEDBACK VERTEX SET, we consider the graph we are working with to be undirected and we allow parallel edges. In this section, we make use of two reduction rules which are common for kernels of the decision variant of FEEDBACK VERTEX SET.

(R1) If there is an edge of multiplicity larger than two, reduce its multiplicity to two.

(R2) If there is a vertex v with degree at most one, remove v .

It can easily be verified that if an instance (G, k) is reduced to (G', k') by one of the rules above, we have $\#\text{minFVS}(G, k) = \#\text{minFVS}(G', k')$. Hence, these rules are safe in the context of counting minimum feedback vertex sets. Observe that if (R2) has exhaustively been applied on a graph G , then all vertices in $V_{\neq 2}(G)$ have degree at least three. For our purposes, we will need one more method to reduce the graph. We first present a lemma that motivates this third reduction rule.

► **Lemma 3.1.** *Let X be a (not necessarily minimum) feedback vertex set of a graph G that is reduced with respect to (R2) and let \mathcal{C} be the set of connected components of $G - V_{\neq 2}(G)$. Then:*

(a) $|V_{\neq 2}(G)| \leq |X| + \sum_{v \in X} \deg_G(v)$, and

(b) $|\mathcal{C}| \leq |X| + 2 \sum_{v \in X} \deg_G(v)$.

Proof. Consider the forest $F := G - X$. Partition the vertices of $V_{\neq 2}(G) \cap V(F) = V_{\neq 2}(G) \setminus X$ into sets $V'_{\leq 1}$, V'_2 and $V'_{\geq 3}$ for vertices that have respectively degree at most one, degree two or degree at least three in F . Furthermore, let V_ℓ denote the leaf nodes of F that have degree two in G . Since G has no vertices of degree at most one by (R2), the leaves of F are exactly the vertices $V_\ell \cup V'_{\leq 1}$. In any tree, the number of vertices of degree at least three is less than the number of leaves, thus $|V'_{\geq 3}| \leq |V'_{\leq 1}| + |V_\ell|$. Each vertex in V'_2 has at least one edge to X since they have degree at least three in G and degree exactly two in F . For a similar reason, each vertex in $V'_{\leq 1}$ has at least two edges to X . Each vertex in V_ℓ has one edge to X . Putting this together gives the following inequality, from which (a) directly follows.

$$\sum_{v \in X} \deg_G(v) \geq |V'_2| + 2|V'_{\leq 1}| + |V_\ell| \geq |V'_2| + |V'_{\leq 1}| + |V'_{\geq 3}| = |V_{\neq 2}(G) \setminus X| \quad (1)$$

To bound the size of the set \mathcal{C} of connected components of $G - V_{\neq 2}(G)$, we instead bound the size of the set \mathcal{C}' of connected components of $G - V_{\neq 2}(G) - X$. Observe that $|\mathcal{C}| \leq |\mathcal{C}'| + |X|$ since removing a vertex from a graph reduces the number of connected components by at most one. (Such a removal can *increase* the number of components by an arbitrary number, which is irrelevant for our argument.) Since X is an FVS, the connected components in \mathcal{C}' can be seen as proper chains. Chains in \mathcal{C}' that have both endpoints in F act as edges between those endpoints when it comes to the connectivity of F . This means that, since F is a forest, there can be at most $|V_{\neq 2}(G) \cap V(F)| \leq \sum_{v \in X} \deg_G(v)$ of such chains by Equation 1. All other chains will have at least one endpoint in X , which means there can be at most $\sum_{v \in X} \deg_G(v)$ of them, implying (b). ◀

Based on Lemma 3.1, the goal of the third reduction rule is to decrease the degree of the vertices of a feedback vertex set. This idea is captured in Lemma 3.2. After presenting this lemma, we combine these results in Lemma 3.3 to create an algorithm to reduce a graph G to an, in context of counting minimum feedback vertex sets, equivalent graph with a bounded number of vertices of degree other than two and a bounded number of chains.

► **Lemma 3.2.** *There exists a polynomial-time algorithm that, given a graph G reduced with respect to (R1), an integer k , a vertex $v \in V(G)$ and a feedback vertex set $Y_v \subseteq V(G) \setminus \{v\}$ of G , outputs a graph G' obtained by removing edges from G such that $\deg_{G'}(v) \leq |Y_v| \cdot (k + 4)$ and $\#\text{minFVS}(G, k) = \#\text{minFVS}(G', k)$.*

Proof. Consider forest $F := G - (Y_v \cup \{v\})$. For each $u \in Y_v$, mark trees of F that have an edge to both v and u until either all such trees are marked or at least $k + 2$ of them are marked. Then, we construct a graph G' from G by removing all edges between v and trees of F that were not marked.

We shall first prove the bound on the degree of v in G' . The vertex v can have edges to vertices in Y_v or in F . Since (R1) has been exhaustively applied, there can be at most $2|Y_v|$ edges between v and Y_v . Each tree in F has at most one edge to v , since otherwise Y_v would not be an FVS of G . In G' , only trees that were marked still have an edge to v , and since we mark at most $k + 2$ trees per vertex in Y_v , we have at most $|Y_v| \cdot (k + 2)$ of such trees. Combining this gives $\deg_{G'}(v) \leq |Y_v| \cdot (k + 4)$.

To show that $\#\text{minFVS}(G, k) = \#\text{minFVS}(G', k)$, we prove that a vertex set $X \subseteq V(G)$ with $|X| \leq k$ is an FVS of G if and only if it is an FVS of G' . Clearly any FVS of G is an FVS of G' since G' is constructed from G by deleting edges. For the opposite direction, assume that X is an FVS of G' and assume for a contradiction that X is not an FVS of G . Then $G - X$ has a cycle W . This cycle must contain an edge $\{v, w\}$ of $E(G) \setminus E(G')$ since $G' - X$ is acyclic. By construction of G' , we know that w is a vertex that belongs to an unmarked tree T . Since cycle W intersects T and since T is a connected component of $G - (Y_v \cup \{v\})$, there must be a vertex $u \in Y_v$ that has an edge to T . Since the edge between v and T is removed in G' , there exist $k + 2$ other trees that are marked and have an edge to both v and u . Since $|X| \leq k$, at least two of these trees are not hit by X . Furthermore, since v and u are part of W , they are also not contained in X . Therefore, there is a cycle in $G' - X$ through v , u and two of the aforementioned trees, contradicting that X is an FVS of G' . ◀

► **Lemma 3.3.** *There is a polynomial-time algorithm that, given a graph G and integer k , outputs a graph G' and integer k' such that the following properties are satisfied.*

- $\#\text{minFVS}(G, k) = \#\text{minFVS}(G', k')$.
- $k' \leq k$.
- $|V_{\neq 2}(G')| = \mathcal{O}(k^3)$.
- $G' - V_{\neq 2}(G')$ has $\mathcal{O}(k^3)$ connected components.

Proof. In our approach, we make use of the linear-time 4-approximation algorithm by Bar-Yehuda et al. [2] that can also approximate the more general problem of: for a given graph, find the smallest FVS that does not contain a given vertex. Our first step is to exhaustively apply (R1) on G and compute a 4-approximate FVS X of the graph. If $|X| > 4k$ then the feedback vertex number of G is larger than k , so we can return a trivial, constant size instance G' and k' such that $\#\text{minFVS}(G', k') = 0$. Otherwise, let G' and k' be a copy of G and k . For each vertex $v \in X$, compute a 4-approximate FVS Y_v of G' that does not contain v . If $|Y_v| > 4k$, then there does not exist a solution of size at most k that does not contain v , so remove v from G' and reduce k' by one. Otherwise, use Lemma 3.2 to reduce the degree of v in G' . Finally, we exhaustively apply reduction rule (R2) on G' .

Computing the 4-approximations and applying the reduction rules can be done in polynomial time. Each rule can only be applied a polynomial number of times, thus the algorithm runs in polynomial time. The fact that $\#\text{minFVS}(G, k) = \#\text{minFVS}(G', k')$

follows from safety of the reduction rules used in the algorithm and $k' \leq k$ follows from the fact that k' is never increased. We know that $|V_{\neq 2}(G')| = \mathcal{O}(k^3)$ and that $G' - V_{\neq 2}(G')$ has $\mathcal{O}(k^3)$ connected components due to Lemma 3.1 and the following bound:

$$\sum_{v \in X} \deg_{G'}(v) \leq \sum_{v \in X} |Y_v| \cdot (k+4) \leq \sum_{v \in X} 4k \cdot (k+4) = |X| \cdot 4k \cdot (k+4) = \mathcal{O}(k^3). \quad \blacktriangleleft$$

The result of Lemma 3.3 is in and of itself not a proper kernel yet, since the chains of the graph it produces can be of arbitrary length. Our strategy to address this is as follows. If these chains are large in terms of k , then we can run an FPT algorithm in $\text{poly}(n)$ time to count the number of minimum solutions. Otherwise, the chains can be replaced by structures of size $\text{poly}(k)$ that do not change the number of minimum feedback vertex sets the instance has. This approach is captured in the following two lemmas and combined in the proof of Theorem 1.1.

► **Lemma 3.4.** *There exists a polynomial-time algorithm that, given a graph G with a chain C and an integer k , outputs a graph G' obtained from G by replacing C with a vertex set C' , and an integer k' , such that:*

- $\#\text{minFVS}(G, k) = \#\text{minFVS}(G', k')$,
- $G - C = G' - C'$,
- $N_G(C) = N_{G'}(C')$,
- $|C'| = \mathcal{O}(\log(|C|)^2)$, and
- $k' = k + \mathcal{O}(\log(|C|)^2)$.

Proof. In case C is a proper chain, the endpoints are defined as $N_G(C)$. If C is not a proper chain, which happens if C is a cycle in G , we choose an arbitrary vertex of C to act as its endpoint. For simplicity, we consider C to have two endpoints, where in some cases these two endpoints might be the same vertex.

First, we assume that the number of vertices of the chain, not including its endpoints, is a power of two, i.e. $|C| = 2^p$ for some integer p . Let $v, u \in V(G)$ be the endpoints of C (possibly $v = u$) and let $C = \{c_0, c_1, \dots, c_{2^p-1}\}$. Then we construct our graph G' by replacing C by a gadget C' , of which an example can be seen in Figure 1. It consists of the following elements.

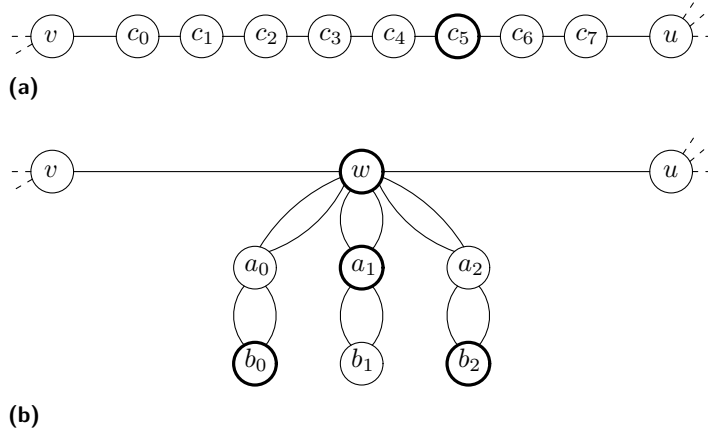
- A vertex w with edges to both v and u .
- Pairs of vertices a_i, b_i for $0 \leq i < p$ such that there is an edge of multiplicity two between w and a_i and between a_i and b_i .

Additionally, we set $k' = k + p$.

We shall now prove that $\#\text{minFVS}(G, k) = \#\text{minFVS}(G', k')$. To this end, we define a mapping f from the set of minimum feedback vertex sets of G to those of G' and show that this is a bijection, which immediately implies that the two sets have the same cardinality. For a natural number m , define $\text{bin}(m)$ to be the binary representation of m on p bits and define $\text{bin}(m)_i$ to be the i 'th least significant bit of $\text{bin}(m)$.

$$f(X) = \begin{cases} X \cup \{a_i \mid 0 \leq i < p\} & \text{if } X \cap C = \emptyset \\ (X \setminus C) \cup \{w\} \cup \{a_i \mid 0 \leq i < p \wedge \text{bin}(m)_i = 0\} \\ \cup \{b_i \mid 0 \leq i < p \wedge \text{bin}(m)_i = 1\} & \text{if } X \cap C = \{c_m\} \end{cases}$$

As a first observation, note that for any minimum feedback vertex set X of G , we have $|X \cap C| \leq 1$ as picking any one vertex from C will already break all cycles that go through the chain.



■ **Figure 1** (a) A chain structure of size eight with two endpoints. (b) The replacement of the structure. An example of the mapping f is also illustrated through the vertices with a thicker border.

▷ **Claim 3.5.** If a set X is a minimum FVS of G , then $f(X)$ is a minimum FVS of G' .

Proof. We prove this in two parts. First we prove that $f(X)$ is an FVS of G' and then we prove that $f(X)$ is indeed an FVS of G' of minimum size.

To prove that $f(X)$ is an FVS of G' , we use a proof by contradiction. Assume $f(X)$ is not an FVS of G' , in which case a (simple) cycle W exists in $G' - f(X)$. This cycle must intersect C' as $X \setminus C = f(X) \setminus C'$ and $G - C = G' - C'$ thus $G - C - X = G' - C' - f(X)$, which means W would otherwise also exist in $G - X$, contradicting that X is an FVS of G . The cycle W cannot contain a b_j vertex since by definition of f , for $0 \leq i < p$, either a_i or b_i is in $f(X)$, which would either mean b_j is isolated in $G' - f(X)$ or is removed. Also, W can not contain any a_j vertex as, by definition of f , if a_j is not in $f(X)$, then both its neighbors w and b_j are in $f(X)$. This leaves only the option that W intersects C' through only vertex w . This means that $W - \{w\}$ contains a path from v to u in $G' - f(X) - \{w\}$. However, since as mentioned before $G - C - X = G' - C' - f(X)$, this same path also exists in $G - X$. Furthermore, since $w \notin f(X)$, that must mean that $X \cap C = \emptyset$ so $(W \setminus \{w\}) \cup C$ would form a cycle in $G - X$ contradicting that X is an FVS of G .

Next we prove that $f(X)$ is a minimum FVS of G' . Assume for sake of a contradiction that $f(X)$ is not a minimum FVS of G' due to the existence of a $Y \subseteq V(G')$ with $|Y| < |f(X)|$ such that Y is an FVS of G' . We first observe that any FVS of G' contains at least p vertices from C' since all p of the pairs (a_i, b_i) form vertex disjoint cycles. Similarly so, if an FVS of G' contains w , then it contains at least $p + 1$ vertices from C' . We distinguish two cases.

Case $w \notin Y$: Then $Y \setminus C'$ is an FVS of G . If $G - (Y \setminus C')$ would contain a cycle, then there would also exist a cycle in $G' - Y$ since $G - C = G' - C'$ and both graphs $G - (Y \setminus C')$ and $G' - Y$ have a vu path, the former through C and the latter through w . Furthermore, $|Y \setminus C'| \leq |Y| - p < |f(X)| - p = |X| + p - p = |X|$, contradicting that X is a minimum FVS of G .

Case $w \in Y$: Then $X' := (Y \setminus C') \cup \{c_j\}$ is an FVS of G for any $0 \leq j < 2^p$. If this were not the case, then, since X' contains a chain vertex, a cycle would need to exist completely in $G - C - X'$ which is the same graph as $G' - C' - Y$. This would contradict Y being an FVS of G' . Furthermore, $|X'| \leq |Y| - (p + 1) + 1 < |f(X)| - p = |X| + p - p = |X|$, contradicting that X is a minimum FVS of G .

As both cases lead to a contradiction, we conclude that $f(X)$ is a minimum FVS of G' . ◁

▷ Claim 3.6. The function f is bijective.

Proof. We first argue that f is injective. Let X and X' be two minimum feedback vertex sets of G such that $X \neq X'$. That means $X \setminus C \neq X' \setminus C$ or $X \cap C \neq X' \cap C$. The former immediately allows us to conclude that $f(X) \neq f(X')$ since $X \setminus C = f(X) \setminus C'$ and $X' \setminus C = f(X') \setminus C'$. In the second case, we have two options. The first is that $X \cap C = \{c_j\}$ and $X' \cap C = \{c_m\}$ for some $j \neq m$, and since binary representations are unique, they lead to different sets $f(X)$ and $f(X')$. The second option is that either X or X' contains no vertex from C while the other one does. Then only one of $f(X)$ or $f(X')$ contains w and the other one does not, so $f(X) \neq f(X')$.

It remains to show that f is surjective. To this end, we take an arbitrary minimum FVS Y of G' and show that there exists a minimum FVS X of G such that $Y = f(X)$. We distinguish two cases:

Case $w \notin Y$: For this case, first observe that $\{a_i \mid 0 \leq i < p\} \subseteq Y$ since otherwise w would form a cycle with one of these a_i vertices in $G - Y$. Furthermore, $b_i \notin Y$ for $0 \leq i < p$ since Y already contains a_i , leaving b_i isolated in $G - Y$ and thus a redundant choice for a minimum FVS. From this we can conclude that $Y \cap C' = \{a_i \mid 0 \leq i < p\}$. Therefore, taking $X := Y \setminus C'$ would give $f(X) = Y$. We have already argued in case $w \notin Y$, that $Y \setminus C'$ is an FVS of G . To show that it is a minimum FVS of G , note that if there was an $X' \subseteq V(G)$, $|X'| < |X|$ such that X' is an FVS of G , then $f(X')$ is an FVS of G' and $|f(X')| = |X'| + p < |X| + p = |Y \setminus C'| + p = |Y| - p + p = |Y|$ which would contradict Y being a minimum FVS of G' .

Case $w \in Y$: For this case, we instead observe that for each pair $\{a_i, b_i\}$, exactly one of $\{a_i, b_i\}$ is in Y , as otherwise Y would not be an FVS of minimum size. Since there are p of such pairs, there is a unique value $0 \leq j < 2^p$ such that the binary representation of j corresponds to the choice of a and b vertices in Y . We can then choose $X := (Y \setminus C') \cup \{c_j\}$. We clearly have that $f(X) = Y$. Also, we have already shown before that X is an FVS of G and the argument that X is a minimum FVS of G is analogous to that in case $w \notin Y$. From this reasoning we can conclude that f is a bijective function from the set of minimum feedback vertex sets of G to the set of minimum feedback vertex sets of G' . ◀

In the argument above, we assumed that the length of the chain is a power of two. We can address this by noting that any natural number can be written as a sum of unique powers of two. Similarly, we can decompose the chain C into a number of subpaths each having a number of vertices that is a unique power of two. We can then apply the replacement described above on each subpath individually to get multiple replacement structures in a chain between the endpoints of C . As seen before, the size of the replacement is linear in the exponent of the length of the chain. In the worst case, when expressing $|C|$ in binary as a sum of distinct powers of two, the exponents of these powers sum up to $\mathcal{O}(\log(|C|)^2)$, which is also the bound on the number of vertices used in our replacement and on the increase in the parameter value. ◀

We remark for Lemma 3.4 that a similar chain replacement gadget without parallel edges can be constructed, at the expense of a linear increase in the size of the gadget.

By adapting the iterative compression algorithm by Cao et al. [6] for the decision FEEDBACK VERTEX SET problem, we can derive the following lemma. Its proof is given in Appendix A.

► **Lemma 3.7.** *There is an algorithm that, given a graph G and integer k , computes $\#\text{minFVS}(G, k)$ in $2^{\mathcal{O}(k)} \cdot \text{poly}(n)$ time.*

► **Theorem 1.1.** *There is a polynomial-time algorithm that, given a graph G and integer k , either*

- *outputs $\#\text{minFVS}(G, k)$, or*
- *outputs a graph G' and integer k' such that $\#\text{minFVS}(G, k) = \#\text{minFVS}(G', k')$ and $|V(G')| = \mathcal{O}(k^5)$ and $k' = \mathcal{O}(k^5)$.*

Proof. First we use the algorithm from Lemma 3.3 to find a graph G^* and integer $k^* \leq k$ such that $\#\text{minFVS}(G, k) = \#\text{minFVS}(G^*, k^*)$, $|V_{\neq 2}(G^*)| = \mathcal{O}(k^3)$ and $G^* - V_{\neq 2}(G^*)$ has $\mathcal{O}(k^3)$ connected components (chains). Then, if there is a chain of size larger than 2^k , we can run the algorithm from Lemma 3.7 to compute $\#\text{minFVS}(G^*, k^*)$ in $\text{poly}(n)$ time since $n > 2^k$. Otherwise, all chains of G^* have size at most 2^k and we can use Lemma 3.4 on G^* to find a graph G' and integer k' such that:

- $\#\text{minFVS}(G', k') = \#\text{minFVS}(G^*, k^*) = \#\text{minFVS}(G, k)$,
- $|V(G')| = \mathcal{O}(k^3) + \mathcal{O}(k^3 \cdot \log(2^k)^2) = \mathcal{O}(k^5)$, and
- $k' = \mathcal{O}(k + k^3 \cdot \log(2^k)^2) = \mathcal{O}(k^5)$. ◀

4 Conclusion

We introduced a new model of kernelization for counting problems: a polynomial-time preprocessing algorithm that either outputs the desired count, or reduces to a provably small instance with the same answer. We showed that for counting the number of minimum solutions of size at most k , a reduction to a graph of size $\text{poly}(k)$ exists for two classic problems.

We believe that the new viewpoint on counting kernelization facilitates a general theory that can be explored for many problems beyond the ones considered here. By following the textbook proof [9, Lemma 2.2] that a decidable parameterized decision problem is fixed-parameter tractable if and only if it admits a kernel (of potentially exponential size), it is easy to show the following equivalence between fixed-parameter tractability of counting problems and our notion of counting kernelization.

► **Lemma 4.1.** *Let $\mathcal{P}: \Sigma^* \times \mathbb{N} \rightarrow \mathbb{N}$ be a computable function for some finite alphabet Σ . Then the following two statements are equivalent:*

1. *There is a computable function $f: \mathbb{N} \rightarrow \mathbb{N}$ and an algorithm that, given an input $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs $\mathcal{P}(x, k)$ in time $f(k) \cdot |x|^{\mathcal{O}(1)}$.*
2. *There is a computable function $f: \mathbb{N} \rightarrow \mathbb{N}$ and a polynomial-time algorithm that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, either:*
 - a. *outputs $\mathcal{P}(x, k)$, or*
 - b. *outputs $(x', k') \in \Sigma^* \times \mathbb{N}$ satisfying $|x'|, k' \leq f(k)$ and $\mathcal{P}(x, k) = \mathcal{P}(x', k')$.*

Hence our view of counting kernelization is generic enough to capture all fixed-parameter tractable counting problems. Determining which counting problems have a *polynomial-size* kernel remains an interesting challenge.

In this work, we focused on counting *minimum-size* solutions (of size at most k). Apart from being of practical interest in several applications, this facilitates several steps in the design and analysis of our preprocessing algorithms. At present, we do not know whether the two considered problems have polynomial-size kernels when counting the number of inclusion-minimal solutions of size exactly k , or the number of (not necessarily minimal or minimum) solutions of size exactly k ; we leave this investigation to future work. To see the importance of the distinction, observe that the number of *minimum* vertex covers of size

at most k in a graph is bounded by 2^k (since the standard 2-way branching discovers all of them) and the Buss kernel preserves their count. But the total number of vertex covers of size at most k cannot be bounded in terms of k in general.

For both problems we investigated, our preprocessing step effectively consists of reducing to an equivalent instance composed of a small core along with $\text{poly}(k)$ simply structured parts, followed by replacing each such part by a small problem-specific gadget. In the world of decision problems, the theory of protrusion replacement [4, 13] gives a generic way of replacing such simply-structured parts by gadgets. Similarly, the condenser-extractor framework [18, 26] can be applied to generic problems as long as they can be captured in a certain type of logic. This leads to the question of whether, in our model of counting kernelization, the design of the gadgets can be automated. Can a notion of meta-kernelization be developed for counting problems?

References

- 1 Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, 2004. doi:10.1145/990308.990309.
- 2 Reuven Bar-Yehuda, Dan Geiger, Joseph Naor, and Ron M. Roth. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and bayesian inference. *SIAM J. Comput.*, 27(4):942–959, 1998.
- 3 Bart M. P. Jansen. The power of preprocessing: Gems in kernelization. <https://www.win.tue.nl/~bjansen/talks/BonnGemsInKernelization.pptx>, 2016.
- 4 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (meta) kernelization. *J. ACM*, 63(5):44:1–44:69, 2016. doi:10.1145/2973749.
- 5 Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. *SIAM J. Comput.*, 22(3):560–572, 1993. doi:10.1137/0222038.
- 6 Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set: New measure and new structures. *Algorithmica*, 73(1):63–86, 2015.
- 7 Radu Curticapean. The simple, little and slow things count: on parameterized counting complexity. *Bull. EATCS*, 120, 2016. URL: <http://eatcs.org/beatcs/index.php/beatcs/article/view/445>.
- 8 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 210–223. ACM, 2017. doi:10.1145/3055399.3055502.
- 9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 10 Henning Fernau. *Parameterized algorithmics: A graph-theoretic approach*. PhD thesis, Habilitationsschrift, Universität Tübingen, Germany, 2005.
- 11 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004. doi:10.1137/S0097539703427203.
- 12 Fedor Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.
- 13 Fedor V. Fomin. Protrusions in graphs and their applications. In Venkatesh Raman and Saket Saurabh, editors, *Parameterized and Exact Computation - 5th International Symposium, IPEC 2010, Chennai, India, December 13-15, 2010. Proceedings*, volume 6478 of *Lecture Notes in Computer Science*, page 3. Springer, 2010. doi:10.1007/978-3-642-17493-3_2.
- 14 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. *SIAM J. Comput.*, 49(6):1397–1422, 2020. doi:10.1137/16M1080264.

- 15 Yoichi Iwata. Linear-time kernelization for feedback vertex set. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 68:1–68:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.68.
- 16 Bart M. P. Jansen and Bart van der Steenhoven. Kernelization for counting problems on graphs: Preserving the number of minimum solutions. *CoRR*, abs/2310.04303, 2023. arXiv:2310.04303.
- 17 Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM J. Comput.*, 22(5):1087–1116, 1993.
- 18 Eun Jung Kim, Maria J. Serna, and Dimitrios M. Thilikos. Data-compression for parametrized counting problems on sparse graphs. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao, editors, *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, volume 123 of *LIPICs*, pages 20:1–20:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ISAAC.2018.20.
- 19 Daniel Lokshtanov, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Kernelization of counting problems. *CoRR*, abs/2308.02188, 2023. doi:10.48550/arXiv.2308.02188.
- 20 Michael Luby and Eric Vigoda. Fast convergence of the Glauber dynamics for sampling independent sets. *Random Struct. Algorithms*, 15(3-4):229–241, 1999.
- 21 Catherine McCartin. Parameterized counting problems. *Annals of Pure and Applied Logic*, 138(1):147–182, 2006. doi:10.1016/j.apal.2005.06.010.
- 22 Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: Simple building blocks of complex networks. *Science (New York, N. Y.)*, 298:824–7, November 2002. doi:10.1126/science.298.5594.824.
- 23 Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. Parameterized counting algorithms for general graph covering problems. In Frank K. H. A. Dehne, Alejandro López-Ortiz, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures, 9th International Workshop, WADS 2005, Waterloo, Canada, August 15-17, 2005, Proceedings*, volume 3608 of *Lecture Notes in Computer Science*, pages 99–109. Springer, 2005. doi:10.1007/11534273_10.
- 24 Pekka Orponen. Dempster’s rule of combination is #p-complete. *Artif. Intell.*, 44(1-2):245–253, 1990.
- 25 Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996.
- 26 Dimitrios M. Thilikos. Compactors for parameterized counting problems. *Comput. Sci. Rev.*, 39:100344, 2021. doi:10.1016/j.cosrev.2020.100344.
- 27 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, 2010. doi:10.1145/1721837.1721848.
- 28 Marc Thurley. Kernelizations for parameterized counting problems. In Jin-yi Cai, S. Barry Cooper, and Hong Zhu, editors, *Theory and Applications of Models of Computation, 4th International Conference, TAMC 2007, Shanghai, China, May 22-25, 2007, Proceedings*, volume 4484 of *Lecture Notes in Computer Science*, pages 703–714. Springer, 2007. doi:10.1007/978-3-540-72504-6_64.
- 29 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. doi:10.1137/0220053.
- 30 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979. doi:10.1016/0304-3975(79)90044-6.
- 31 D. J. A. Welsh. Graph theory and theoretical physics. *The Mathematical Gazette*, 54(390):432–433, 1970. doi:10.2307/3613919.

A An FPT algorithm for #minFVS

► **Lemma 3.7.** *There is an algorithm that, given a graph G and integer k , computes #minFVS(G, k) in $2^{\mathcal{O}(k)} \cdot \text{poly}(n)$ time.*

Proof. The pseudocode of an algorithm to solve the disjoint minimum feedback vertex set counting problem is given in Algorithm 2 and is based on the algorithm by Cao et al [6]. In fact, our algorithm solves a slightly more general version of the problem, where the vertices of the graph G are weighted by a function $w: V(G) \rightarrow \mathbb{N}$. In case G has no FVS of size at most k that is disjoint from W , then #DJ-FVS(G, w, W, k) will output a pair (a, b) with $a = \infty$ and $b = 0$. Otherwise, a will be the size of a minimum FVS disjoint of W and

$$b = \sum_{\substack{|S|=a, \\ S \in \text{FVS}(G), \\ S \cap W = \emptyset}} \prod_{v \in S} w(v).$$

We refer to this value as the *weighted disjoint minimum FVS sum* of G . The weight of a vertex v essentially models the number of distinct alternatives there are for a vertex v that, in the context of choosing them for a minimum FVS of G , achieve the same result. When we assign a weight of one to each vertex of a graph G , then the weighted minimum FVS sum of G is equal to the number of minimum feedback vertex sets of G .

In the pseudocode we make use of binary operator \oplus , which is defined to work on pairs as follows:

$$(a_1, b_1) \oplus (a_2, b_2) = \begin{cases} (a_1, b_1) & \text{if } a_1 < a_2 \\ (a_2, b_2) & \text{if } a_1 > a_2 \\ (a_1, b_1 + b_2) & \text{if } a_1 = a_2 \end{cases}$$

For the operators $+$ and \cdot we assume element-wise functionality when applied to pairs, i.e. $(a_1, b_1) + (a_2, b_2) = (a_1 + a_2, b_1 + b_2)$. Furthermore, for a weight function w of G and $X \subseteq V(G)$, we use $w|_X$ to denote the restriction of w to X .

We shall now explain how the #DJ-FVS algorithm works by going over the pseudocode in Algorithm 2. It makes use of a branching strategy with measure function $\ell + k$, where ℓ is the number of connected components of $G[W]$.

Lines 1-3 are the base cases of the algorithm. In lines 4-5 we remove vertices of degree at most one, which corresponds to (R2). In lines 6-7 we contract the chains of G existing in $G - W$ one edge at a time. Note for line 8 that H is a forest since W is an FVS of G . For lines 9-10, if a vertex v would form a cycle with W it should be contained in all solutions so we recurse on this choice. In lines 11-14, vertex v has at least two neighbors in W , but since $G[W \cup \{v\}]$ does not form a cycle these neighbors must be in different connected components of $G[W]$. Thus we branch on v not being in a solution, which corresponds with adding v to W , and on v being in a solution by removing it from the graph. In the former branch ℓ decreases, while in the latter k decreases.

In line 15, we choose a vertex $v \in V(H)$ that is not a leaf of tree H such that at most one of its neighbors in H is not a leaf of H . Note that such a vertex always exists for a tree that does not consist of only leaves. Furthermore, at this point of the algorithm no tree of H can consist of only leaves. If a tree of H consists of a single leaf, then either it has degree one in G , but then lines 4-5 would have gotten rid of it, or it has degree at least two, in which case the if condition on line 9 or the if condition on line 11 would have been satisfied. If a

tree of H consists of two leaves, then either both have degree two in which case the edge between them would have been contracted by lines 6-7, or one of the if statements of line 9 or line 11 would have been applicable to one of the two leaves.

First consider the case where v has one neighbor in W (lines 16-22). In that case we pick c in line 16 to be a child of v . For these two vertices, we branch over all possible combinations of whether or not they should be in a solution, while realizing that a minimum FVS that contains v can never contain c . Note that if $G[W \cup \{v, c\}]$ does not form a cycle, both v and c must have a neighbor in a different connected component of W so the branch in line 19 decreases ℓ . The branches on line 20 and 21 decrease k while not increasing ℓ .

Finally, we have the case that v has no neighbors in W (lines 23-31). That must mean that v has at least two children, which are all leaves by how we chose v . If v would have had only one child, then the edge between v and the child would have been contracted in lines 6-7. Thus we let c_1 and c_2 be two distinct children of v . Again, we branch over all viable combinations of how these three vertices can be part of solutions. The logic here is similar to that of the previous case. Here as well, in all branches, the measure $k + \ell$ decreases.

Since the algorithm branches in at most five directions and the time per iteration is polynomial in n , the runtime of the disjoint algorithm becomes $5^{k+\ell} \cdot n^{O(1)}$. We can use Algorithm 2 to compute $\# \text{minFVS}(G, k)$ by taking an FVS Z of G and running the disjoint algorithm for all subsets of Z , simulating the ways an FVS can intersect Z . The pseudocode for this compression algorithm can be seen in Algorithm 1. To get an FVS of G of size at most k , we can simply run one of the existing algorithms designed for this, for example the one by Cao et al. [6] which runs in time $\mathcal{O}(3.83^k) \cdot \text{poly}(n)$. If this reports that no such FVS exists, we output $\# \text{minFVS}(G, k) = 0$. Otherwise, we use the found FVS for the compression algorithm. Using the fact that the FVS used by the compression algorithm is of size at most k , we can bound the runtime of the complete algorithm to compute $\# \text{minFVS}(G, k)$ at $26^k \cdot n^{O(1)}$. ◀

■ **Algorithm 1** $\# \text{FVS-COMPRESSION}(G, k, Z)$.

Input: Graph G , integer k , FVS Z of G of size at most k
Output: A pair (a, b) with a being the feedback vertex number of G and $b = \# \text{minFVS}(G, k)$

- 1: $s = (\infty, 0)$
- 2: **for** $X_Z \subseteq Z$ **do**
- 3: $s' = (|X_Z|, 0) + \# \text{DJ-FVS}(G - X_Z, w, Z \setminus X_Z, k - |X_Z|)$ with $\forall v \in V(G - X_Z) :$
 $w(v) = 1$
- 4: $s = s \oplus s'$
- 5: **return** s

Algorithm 2 #DJ-FVS(G, w, W, k).

Input: Graph G , weight function $w: V(G) \rightarrow \mathbb{N}$, FVS W of G , integer k

Parameter: $k + \ell$, with $\ell = \#$ components of $G[W]$




Output: A pair (a, b) where a is the size of a minimum FVS S of G such that $S \cap W = \emptyset$ and b is the weighted disjoint minimum FVS sum of G . If no such FVS of size at most k exists, then $(a, b) = (\infty, 0)$.

- 1: **if** $k < 0$ **then return** $(\infty, 0)$
 - 2: **if** $G[W]$ has a cycle **then return** $(\infty, 0)$
 - 3: **if** $G - W$ is empty **then return** $(0, 1)$
 - 4: **if** $\exists v \in V(G - W): \deg_G(v) \leq 1$ **then**
 - 5: **return** #DJ-FVS($G - v, w|_{V(G) \setminus \{v\}}, W, k$)
 - 6: **if** $\exists \{v, u\} \in E(G): \deg_G(v) = \deg_G(u) = 2$ and $v, u \notin W$ **then**
 - 7: **return** #DJ-FVS(G', w', W, k), where G' is G with edge $\{v, u\}$ contracted to a single vertex s and w' is the weight function $w|_{V(G')}$ with $w'(s) = w(v) + w(u)$.
 - 8: Let H be forest $G - W$.
 - 9: **if** $\exists v \in V(H): G[W \cup \{v\}]$ has a cycle **then**
 - 10: **return** $(1, 0) + (1, w(v)) \cdot \#DJ-FVS(G - v, w|_{V(G) \setminus \{v\}}, W, k - 1)$
 - 11: **if** $\exists v \in V(H): |N_G(v) \cap W| \geq 2$ **then**
 - 12: $X_0 = \#DJ-FVS(G, w, W \cup \{v\}, k)$
 - 13: $X_1 = (1, 0) + (1, w(v)) \cdot \#DJ-FVS(G - v, w|_{V(G) \setminus \{v\}}, W, k - 1)$
 - 14: **return** $X_0 \oplus X_1$
 - 15: Let $v \in V(H)$ be a vertex that is not a leaf of H such that at most one vertex in $N_H(v)$ is not a leaf of H .
 - 16: **if** $|N_G(v) \cap W| = 1$ **then**
 - 17: Pick $c \in V(H)$ such that $N_G(c) = \{v, x\}$ for some $x \in W$
 - 18: **if** $G[W \cup \{v, c\}]$ forms a cycle **then** $X_{00} = (\infty, 0)$
 - 19: **else** $X_{00} = \#DJ-FVS(G, w, W \cup \{v, c\}, k)$
 - 20: $X_{10} = (1, 0) + (1, w(v)) \cdot \#DJ-FVS(G - v, w|_{V(G) \setminus \{v\}}, W, k - 1)$
 - 21: $X_{01} = (1, 0) + (1, w(c)) \cdot \#DJ-FVS(G - c, w|_{V(G) \setminus \{c\}}, W \cup \{v\}, k - 1)$
 - 22: **return** $X_{00} \oplus X_{10} \oplus X_{01}$
 - 23: **if** $|N_G(v) \cap W| = 0$ **then**
 - 24: Pick $c_1, c_2 \in V(H), c_1 \neq c_2$ such that $N_G(c_1) = \{v, x\}$ and $N_G(c_2) = \{v, y\}$ for some $x, y \in W$
 - 25: **if** $G[W \cup \{v, c_1, c_2\}]$ forms a cycle **then** $X_{000} = (\infty, 0)$
 - 26: **else** $X_{000} = \#DJ-FVS(G, w, W \cup \{v, c_1, c_2\}, k)$
 - 27: $X_{100} = (1, 0) + (1, w(v)) \cdot \#DJ-FVS(G - v, w|_{V(G) \setminus \{v\}}, W, k - 1)$
 - 28: $X_{010} = (1, 0) + (1, w(c_1)) \cdot \#DJ-FVS(G - c_1, w|_{V(G) \setminus \{c_1\}}, W \cup \{v, c_2\}, k - 1)$
 - 29: $X_{001} = (1, 0) + (1, w(c_2)) \cdot \#DJ-FVS(G - c_2, w|_{V(G) \setminus \{c_2\}}, W \cup \{v, c_1\}, k - 1)$
 - 30: $X_{011} = (2, 0) + (1, w(c_1) \cdot w(c_2)) \cdot \#DJ-FVS(G - \{c_1, c_2\}, w|_{V(G) \setminus \{c_1, c_2\}}, W \cup \{v\}, k - 2)$
 - 31: **return** $X_{000} \oplus X_{100} \oplus X_{010} \oplus X_{001} \oplus X_{011}$
-

On the Parameterized Complexity of Multiway Near-Separator

Bart M. P. Jansen   

Eindhoven University of Technology, The Netherlands

Shivesh K. Roy   

Eindhoven University of Technology, The Netherlands

Abstract

We study a new graph separation problem called MULTIWAY NEAR-SEPARATOR. Given an undirected graph G , integer k , and terminal set $T \subseteq V(G)$, it asks whether there is a vertex set $S \subseteq V(G) \setminus T$ of size at most k such that in graph $G - S$, no pair of distinct terminals can be connected by two pairwise internally vertex-disjoint paths. Hence each terminal pair can be separated in $G - S$ by removing at most one vertex. The problem is therefore a generalization of (NODE) MULTIWAY CUT, which asks for a vertex set for which each terminal is in a different component of $G - S$. We develop a fixed-parameter tractable algorithm for MULTIWAY NEAR-SEPARATOR running in time $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$. Our algorithm is based on a new pushing lemma for solutions with respect to important separators, along with two problem-specific ingredients. The first is a polynomial-time subroutine to reduce the number of terminals in the instance to a polynomial in the solution size k plus the size of a given suboptimal solution. The second is a polynomial-time algorithm that, given a graph G and terminal set $T \subseteq V(G)$ along with a single vertex $x \in V(G)$ that forms a multiway near-separator, computes a 14-approximation for the problem of finding a multiway near-separator not containing x .

2012 ACM Subject Classification Mathematics of computing \rightarrow Graph algorithms; Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Approximation algorithms analysis

Keywords and phrases fixed-parameter tractability, multiway cut, near-separator

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.28

Related Version *Full Version:* <https://arxiv.org/abs/2310.04332>

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803421, ReduceSearch).



1 Introduction

Graph separation problems play an important role in the study of graph algorithms. While the problem of finding a minimum vertex set whose removal separates two terminals s and t can be solved in polynomial-time via the Ford-Fulkerson algorithm [13], many variations of the problem are NP-complete. They form a fruitful subject of investigation in the study of parameterized algorithmics, where a typical goal is to develop an algorithm that finds a suitable separator of size k in an n -vertex input graph in time $f(k) \cdot n^{\mathcal{O}(1)}$, or concludes that no such solution exists. Landmark results in this area include the FPT algorithms for MULTIWAY CUT [5, 8, 15, 20, 27] (in which the goal is to find a vertex set which separates any pair of terminals from a given set T) and MULTICUT [3, 22] (in which only a specified subset of the terminal pairs must be separated) in undirected graphs.



© Bart M. P. Jansen and Shivesh K. Roy;

licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 28; pp. 28:1–28:18

Leibniz International Proceedings in Informatics



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

After the parameterized complexity of the most fundamental separation problems in this area were settled, researchers started considering variations on the theme of graph separation, including STEINER MULTICUT [4] (given a sequence of subsets of terminals T_1, \dots, T_ℓ , find a vertex set separating at least one pair $\{t_i \in T_i, t_j \in T_j\}$ for each $i \neq j$), MULTIWAY CUT-UNCUT [6, 19] (given an equivalence relation \mathcal{R} on a set of terminal vertices, find a vertex set whose removal leaves terminals t_i, t_j in the same connected component if and only if $t_i \equiv_{\mathcal{R}} t_j$), and STABLE MULTIWAY CUT [21] (find a multiway cut that is independent).

In this paper we start to explore a new variation of the separation theme from the parameterized perspective. Rather than asking for a vertex set which fully separates some given terminal pairs, we are interested in *nearly* separating terminals: in the remaining graph, there should not be *two or more* internally vertex-disjoint paths connecting a terminal pair. We therefore study the following problem, which we believe is a natural extension in the well-studied area of graph separation problems.

MULTIWAY NEAR-SEPARATOR (MWNS) **Parameter:** k
Input: An undirected graph G , terminal set $T \subseteq V(G)$, and a positive integer k .
Question: Is there a set $S \subseteq V(G) \setminus T$ with $|S| \leq k$ such that there does not exist a pair of distinct terminals $t_i, t_j \in T$ with *two* internally vertex-disjoint t_i - t_j paths in $G - S$?

Note that, by Menger's theorem, the requirement on solutions S to MWNS is equivalent to the requirement that in the graph $G - S$, any terminal pair can be separated by removing a non-terminal vertex. One could therefore imagine applications of this problem in the study of disrupting communications between nodes in a network. While the standard MULTIWAY CUT problem captures the setting that all potential for communication between terminals has to be broken, the size of a solution to the near-separation problem can be arbitrarily much smaller while it still ensures the following property: the communication between each terminal pair is either broken by the solution, or there is at least one non-terminal vertex through which all communications of the pair must pass, so that it may be intercepted at that point. A related problem of reducing connectivity between nodes by removing edges or vertices was studied by Barman and Chawla [1], who presented various approximation algorithms.

Using the (perhaps non-standard) view that a direct edge between two vertices t_i, t_j means there are two internally vertex-disjoint paths between them (the intersection of the set of interior vertices is empty), the requirement that in $G - S$ there do not exist two internally vertex-disjoint paths between any pair of distinct terminals, can alternatively be shown to be equivalent to demanding that T is an independent set and there is no simple cycle containing at least two vertices from T (see Proposition 2.7). Simple cycles containing two vertices from T , henceforth called T -cycles, therefore play an important role in our arguments. Based on this alternative characterization, the near-separator problem is related to the SUBSET FEEDBACK VERTEX SET problem, which asks for a minimum vertex set intersecting all cycles that contain at least one terminal [9], although there the solution is allowed to contain terminal vertices.

A simple reduction (Lemma B.1) shows that MULTIWAY NEAR-SEPARATOR is NP-complete. It forms a generalization of the (NODE) MULTIWAY CUT problem with undeletable terminals: an instance $(G, T = \{t_1, \dots, t_\ell\}, k)$ of the latter problem can be reduced to an equivalent instance (G', T, k) of MWNS by inserting $|T| - 1$ new non-terminal vertices $w_1, \dots, w_{|T|-1}$ with $N(w_i) = \{t_i, t_{i+1}\}$.

The MULTIWAY NEAR-SEPARATOR problem can be shown to be non-uniformly fixed-parameter tractable parameterized by k using the technique of recursive understanding [19], as the problem can be formulated in Monadic Second-Order Logic and can be shown to become

fixed-parameter tractable on $(s(k), k + 1)$ -unbreakable graphs for some function $s: \mathbb{N} \rightarrow \mathbb{N}$ by branching on small connected vertex sets with small neighborhoods. This only serves as a complexity classification, however, as the resulting algorithms are non-uniform and have a large (and unknown) parameter dependence $f(k)$. In this paper, our goal is to understand the structure of MULTIWAY NEAR-SEPARATOR and develop an efficient (and uniform) parameterized FPT algorithm for the problem.

Our results

The main result of our paper is the following theorem, showing that MWNS has a uniform FPT algorithm with parameter dependence $2^{\mathcal{O}(k \log k)}$.

► **Theorem 1.1** (★). *MULTIWAY NEAR-SEPARATOR can be solved in time $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$.*

The starting point for the algorithm is the approach for solving MULTIWAY CUT via important separators. The *pushing lemma* due to Marx [20] [7, Lemma 8.53] states that for any instance (G, T, k) of MULTIWAY CUT and any choice of terminal $t \in T$, there is an optimal solution which contains an *important* $(t, T \setminus \{t\})$ -separator. As the number of important separators of size at most k is bounded by 4^k , we can construct a solution by picking an arbitrary terminal which is not yet fully separated from the remaining terminals and branching on all choices of including a $(t, T \setminus \{t\})$ -separator in the solution.

Adapting this strategy directly for MULTIWAY NEAR-SEPARATOR fails for several reasons. Most importantly, since terminal pairs are allowed to remain in the same connected component, it is possible that in an instance with a solution S of size k , there exists a terminal $t \in T$ for which there are no $(t, T \setminus \{t\})$ -separators of size $f(k)$ for any function f . This happens when such a terminal t is located in a “central” block in the block-cut tree of $G - S$. However, there always exists a terminal $t' \in T$ for which there does exist a $(t', T \setminus \{t'\})$ -separator of size at most $k + 1$, and for which furthermore a variation of the pushing lemma can be proven: there is an optimal solution which contains all-but-one vertex of an important $(t', T \setminus \{t'\})$ -separator of size at most $k + 1$. Intuitively, such a terminal t' can be found in a leaf block of the block-cut tree of $G - S$. Hence there *exists* a terminal for which branching on important separators can make progress in identifying the solution, but not all terminals have this property and a priori it is not clear which is the right one.

To resolve this issue, we will effectively have the algorithm try all choices for the terminal t which is near-separated from all other terminals by an important separator. To ensure the branching factor of the resulting algorithm is bounded in terms of the parameter k , while the number of terminals T may initially be arbitrarily large compared to k , we therefore have to reduce the number of terminals to $k^{\mathcal{O}(1)}$ in a preprocessing phase.

For the standard MULTIWAY CUT problem, a preprocessing step based on the linear-programming relaxation of the problem can be used to reduce the number of terminals to $2k$ [8] (cf. [23]). For the near-separation variant we consider, it seems unlikely that the linear-programming relaxation has the same nice properties (such as half-integrality) as for the original problem, let alone that the resulting fractional solutions are useful for a reduction in the number of terminals. As one of our main technical ingredients, we therefore develop a combinatorial preprocessing algorithm to reduce the number of terminals to a polynomial in the solution size k plus the size of a given (suboptimal) near-separator S , which will be available via the technique of iterative compression [24] [7, §4]. The preprocessing step is based on concrete reduction rules operating in the graph.

► **Theorem 1.2 (★).** *There is a polynomial-time algorithm that, given an instance (G, T, k) of MWNS and a multiway near-separator \hat{S} for terminal set T in G , outputs an equivalent instance (G', T', k') such that:*

1. G' is an induced subgraph of G ,
2. T' is a subset of T of size $\mathcal{O}(k^5 \cdot |\hat{S}|^4)$, and
3. $k' \leq k$.

Moreover, there is a polynomial-time algorithm that, given a solution S' for (G', T', k') , outputs a solution S of (G, T, k) .

Note that the algorithm even runs in *polynomial-time*, and may therefore be a useful ingredient to build a polynomial kernelization for this problem or variations thereof. To obtain this terminal-reduction algorithm, it turns out to be useful to know whether the role of a vertex x in a suboptimal near-separator S can be taken over by $\mathcal{O}(k)$ alternative vertices. If not, then this immediately leads to the conclusion that such a vertex x belongs to any optimal solution to the problem. On the other hand, knowing a small vertex set S_x for which $(S \setminus \{x\}) \cup S_x$ is also a near-separator reveals a lot of structure in the instance which can be exploited by the reduction rule. This usage is similar as the use of the *blocker* [7, §9.1.3] in Thomassé's kernelization algorithm for FEEDBACK VERTEX SET [26].

Given a suboptimal near-separator S , we are therefore interested in determining, for a given $x \in S$, whether it is possible to obtain a near-separator S' by replacing x by a set of $\mathcal{O}(k)$ vertices. This is equivalent to finding a near-separator of size $\mathcal{O}(k)$ which avoids the use of vertex x in the graph $G' := G - (S \setminus \{x\})$. Hence this task effectively reduces to finding a solution not containing x in the graph G' for which $\{x\}$ forms a near-separator. We give a polynomial-time 14-approximation for this problem.

► **Theorem 1.3 (★).** *There is a polynomial-time algorithm that, given a graph G , terminal set $T \subseteq V(G)$, and a vertex $x \in V(G)$ such that $\{x\}$ is a multiway near-separator for terminal set T in G , outputs a multiway near-separator $S_x \subseteq V(G) \setminus \{x\}$ for T in G such that $|S_x| \leq 14|S_x^*|$, where $S_x^* \not\ni x$ is a smallest multiway near-separator for T in G that avoids x .*

Theorem 1.3 can be compared to a result for the CHORDAL DELETION problem, where the goal is to delete a minimum number of vertices to break all induced cycles of length at least four (holes). A key step in the polynomial kernelization algorithm for the problem due to Jansen and Pilipczuk is a subroutine ([16, Lemma 1.3]) which, given a graph G and vertex x for which $G - \{x\}$ is chordal, outputs a set of some $\ell \geq 0$ holes pairwise intersecting only in x , together with a vertex set S of size at most 12ℓ not containing x whose removal makes G chordal. Hence S is a 12-approximation for the problem of finding a chordal deletion set which avoids x .

Related work

Apart from the aforementioned work on graph separation problems, the work of Golovach and Thilikos [14] is related to our setting. They consider the problem of removing at most k edges from a graph to split it into exactly t connected components C_1, \dots, C_t such that C_i has edge-connectivity at least λ_i for a given sequence $(\lambda_1, \dots, \lambda_t)$. Besides recursive understanding, which only leads to non-uniform FPT classifications, another generic tool for deriving fixed-parameter tractability of separation problems is the treewidth reduction technique by Marx and Razgon [22]. To be able to apply the technique, the number of terminals must be bounded in terms of the parameter k , which is not the case in general.

Even if one uses Theorem 1.2 to bound the number of terminals first, it is not clear if the technique can be applied since the solutions to be preserved are not minimal separators in the graph. Furthermore, successful application of the technique would give double-exponential algorithms at best, due to having to perform dynamic programming on a tree decomposition of width $2^{\Omega(k)}$.

The problem of computing a near-separator avoiding a vertex x is related to the problem of computing an r -fault tolerant solution to a vertex deletion problem, which is a solution from which any r vertices may be omitted without invalidating the solution. The computation of r -fault tolerant solutions has been studied for the FEEDBACK VERTEX SET problem [2], which has a polynomial-time $\mathcal{O}(r)$ -approximation.

The FPT algorithm for SUBSET FEEDBACK VERTEX SET in undirected graphs due to Cygan et al. [9] bears some similarity to ours, in that it also uses reduction rules to bound the number of terminals followed by an algorithm which is exponential in the solution size and the number of terminals. However, SUBSET FEEDBACK VERTEX SET behaves differently from the problem we consider, since in the latter the structures to be hit always involve *pairs* of terminals to be near-separated. On the other hand, a solution to SUBSET FEEDBACK VERTEX SET will reduce the connectivity in the graph to the extent that there will no longer be two internally vertex-disjoint paths between any terminal pair, but also has to ensure that there are no cycles through a single terminal. This leads to significant differences in the approach.

Organization

We begin with short preliminaries with the crucial definitions. We prove our main Theorem 1.1 in Section 3 by assuming Theorem 1.2. Next, in Section 4, we prove Theorem 1.3 by giving a polynomial-time construction of a near-separator avoiding a specific vertex. The proof of Theorem 1.2 is given in Section 5. The proofs of statements marked with (★) are located in the full version [17].

2 Preliminaries

Graphs. We use standard graph-theoretic notation, and we refer the reader to Diestel [11] for any undefined terms. We consider simple unweighted undirected graphs. A graph G has vertex set $V(G)$ and edge set $E(G)$. We use shorthand $n = |V(G)|$ and $m = |E(G)|$. The set $\{1, \dots, \ell\}$ is denoted by $[\ell]$. The open neighborhood of $v \in V(G)$ is $N_G(v) := \{u \mid \{u, v\} \in E(G)\}$, where we omit the subscript G if it is clear from context. For a vertex set $S \subseteq V(G)$ the open neighborhood of S , denoted $N_G(S)$, is defined as $S := \bigcup_{v \in S} N_G(v) \setminus S$. For $S \subseteq V(G)$, the graph induced by S is denoted by $G[S]$. For two vertices x, y in a graph G , an x - y path is a sequence $(x = v_1, \dots, v_k = y)$ of vertices such that $\{v_i, v_{i+1}\} \in E(G)$ for all $i \in [k - 1]$. Furthermore, the vertices v_2, \dots, v_{k-1} are called the *internal vertices* of the x - y path. Given a path $P = (v_1, \dots, v_k)$ and indices $i, j \in [k]$, with $j \geq i$, we use $P[v_i, v_j]$ to denote the subpath of the path P which starts from v_i and ends at v_j . Moreover, we use shorthand $P(v_i, v_j) = P[v_i, v_j] - \{v_i\}$, $P(v_i, v_j) = P[v_i, v_j] - \{v_j\}$, and $P(v_i, v_j) = P[v_i, v_j] - \{v_i, v_j\}$. Given a p_1 - p_k path $P = (p_1, \dots, p_k)$ and a q_1 - q_ℓ path $Q = (q_1, \dots, q_\ell)$ with $p_k = q_1$ such that P and Q are internally vertex-disjoint, we use $P \cdot Q$ to denote the p_1 - q_ℓ path $(p_1, \dots, p_k, q_2, \dots, q_\ell)$ obtained by first traversing P and then Q . We say that a path P in G *intersects* a vertex $v_i \in V(G)$ if $v_i \in V(P)$, similarly, for a set $S \subseteq V(G)$, we say that path P intersects S if $V(P) \cap S \neq \emptyset$. For $S \subseteq V(G)$, an x - y path in G is called an S -path if $x, y \in S$. For $S \subseteq V(G)$, cycles C_1, C_2 in G are said to be S -disjoint if $V(C_1) \cap V(C_2) \subseteq S$.

We now define a few basic notations about block-cut graphs (for completeness we define the notion of block-cut graph in Definition A.2) that we will use henceforth. Given a graph G with connected components C_1, \dots, C_m , a *rooted block-cut forest* \mathcal{F} of G is a block-cut forest containing block-cut trees $\mathcal{T}_1, \dots, \mathcal{T}_m$ such that for each $i \in [m]$, the tree \mathcal{T}_i is a block-cut tree of C_i that is rooted at an arbitrary block of \mathcal{T}_i . Given a rooted forest \mathcal{F} and a vertex $v \in V(\mathcal{F})$, we use $\text{parent}_{\mathcal{F}}(v)$ to denote the parent of v (if v is a root then $\text{parent}_{\mathcal{F}}(v) = \emptyset$) and $\text{child}_{\mathcal{F}}(v)$ to denote the set containing all children of v (if v is a leaf then $\text{child}_{\mathcal{F}}(v) = \emptyset$). Given a rooted block-cut forest \mathcal{F} of G , and a node d of \mathcal{F} , we use $V_G(\mathcal{F}_d)$ to denote the vertices of G occurring in blocks of the subtree rooted at d . Furthermore, we use G_d to denote the graph induced by the vertex set $V_G(\mathcal{F}_d)$, i.e., $G_d := G[V_G(\mathcal{F}_d)]$. We also need the following observations.

► **Observation 2.1.** *Let G be a graph, and let B_1, B_2 be two distinct blocks of G such that $V(B_1) \cap V(B_2) = \{v\}$. In the block-cut graph G' of G , it holds that the distance between blocks B_1 and B_2 is two with v as an intermediate vertex.*

► **Observation 2.2.** *Consider a graph G , terminal set $T \subseteq V(G)$, and a MWNS $S \subseteq V(G)$ of (G, T) . Then each block B of $G - S$ contains at most one terminal.*

Two MWNS instances (G, T, k) and (G', T', k') are said to be *equivalent* if it holds that (G, T, k) is a YES-instance of MWNS if and only if (G', T', k') is a YES-instance of MWNS. An instance (G, T, k) of MWNS is said to be *non-trivial* if \emptyset is not a solution of (G, T, k) . A terminal $t \in T$ is said to be *nearly-separated* in G if there does not exist another terminal $t' \in T \setminus \{t\}$ such that there are 2 internally vertex-disjoint t - t' paths in G .

Throughout this manuscript we use MULTIWAY NEAR-SEPARATOR (MWNS) to denote the parameterized version of the multiway near-separator problem, whereas given a graph G and terminal set T we use multiway near-separator (MWNS) to refer to the graph-theoretic concept of nearly-separating a terminal set T in G . Formally, it is defined as follows.

► **Definition 2.3** (Multiway near-separator (MWNS)). *Given a graph G and terminal set $T \subseteq V(G)$, a set $S \subseteq V(G)$ is called a *multiway near-separator* (MWNS) of (G, T) if $S \cap T = \emptyset$ and there does not exist a pair of distinct terminals $t_i, t_j \in T$ such that $G - S$ contains two internally vertex-disjoint t_i - t_j paths.*

► **Definition 2.4** (r -redundant MWNS). *Given a graph G and terminal set $T \subseteq V(G)$, a set $S^* \subseteq V(G) \setminus T$ is an r -redundant MWNS of (G, T) if for all $R \subseteq S^*$ with $|R| \leq r$, the set $S^* \setminus R$ is a MWNS of (G, T) .*

► **Definition 2.5** (x -avoiding MWNS). *Given a graph G , terminal set $T \subseteq V(G)$, and a vertex $x \in V(G)$, a set $S_x \subseteq V(G) \setminus T$ is called an x -avoiding MWNS of (G, T) if $x \notin S_x$ and S_x is a MWNS of (G, T) . Among all x -avoiding MWNS of (G, T) , one with the minimum cardinality is called a *minimum x -avoiding MWNS* of (G, T) .*

Next, we define T -cycle and give a characterization of MWNS in terms of hitting T -cycles.

► **Definition 2.6** (T -cycle and T -cycle on x). *Given a graph G and terminal set $T \subseteq V(G)$, a cycle C in G is called a T -cycle if $|V(C) \cap T| \geq 2$. Moreover, if C also contains a vertex $x \in V(G)$, then C is called a T -cycle on x .*

We now show that several ways of looking at a near-separator are equivalent.

► **Proposition 2.7 (★).** *Given a graph G , terminal set $T \subseteq V(G)$, and a non-empty set $S \subseteq V(G) \setminus T$, the following conditions are equivalent:*

1. *For each pair of distinct terminals $t_i, t_j \in T$, the graph $G - S$ does not contain t_i - t_j paths P_1, P_2 which are pairwise internally vertex-disjoint. (Note that P_1 may be identical to P_2 if there are no internal vertices.)*
2. *For each pair of distinct terminals $t_i, t_j \in T$, there is a vertex $v \in V(G) \setminus T$ such that t_i and t_j belong to different connected components of $G - (S \cup \{v\})$.*
3. *The set T is an independent set and $G - S$ does not contain a simple cycle C containing at least two terminals (i.e., a T -cycle).*

Due to space constraints we defer the remaining preliminaries about graphs (including block-cut graphs and important separators) and parameterized algorithms to Appendix A.

3 FPT algorithm for Multiway Near-Separator

In this section we prove Theorem 1.1 assuming Theorem 1.2, which we prove later in Section 5. We use the combination of bounded search trees and iterative compression [7, §3–4] to obtain the FPT algorithm. Towards this, we first present the following structural lemma for a MWNS $S \subseteq V(G)$ of (G, T) . It says that in $G - S$, there is a terminal that can simultaneously be separated from *all* other terminals by the removal of a single non-terminal v .

► **Lemma 3.1 (★).** *Let (G, T, k) be a non-trivial instance of MWNS, and let $S \subseteq V(G) \setminus T$ be a solution. Then there exists a terminal $t \in T$ and a non-terminal vertex $v \in V(G) \setminus T$ such that $S \cup \{v\}$ is a $(t, T \setminus \{t\})$ -separator.*

Marx [20] [7, Lemma 8.18] introduced a pushing lemma for MULTIWAY CUT to prove that MULTIWAY CUT is FPT. In the following lemma, we present a pushing lemma for MWNS.

► **Lemma 3.2 (Pushing lemma for MWNS (★)).** *Let (G, T, k) be a non-trivial instance of MWNS and let $S \subseteq V(G) \setminus T$ be a solution. Then there exists a terminal $t \in T$ and a solution $S^* \subseteq V(G) \setminus T$ with $|S^*| \leq |S|$ for which one of the following holds:*

1. *there is an important $(t, T \setminus \{t\})$ -separator S_t^* of size at most k such that $S_t^* \subseteq S^*$, or*
2. *there is an important $(t, T \setminus \{t\})$ -separator S_t of size at most $(k + 1)$, and there exists a vertex $v \in S_t$ such that $(S_t \setminus \{v\}) \subseteq S^*$.*

The following lemma forms the heart of the FPT algorithm (Theorem 1.1). It says that there exists an FPT algorithm that can compress a $k + 1$ -sized MWNS of (G, T) to a k -sized MWNS if (G, T, k) is a YES-instance of MWNS. This is effectively the compression step of the iterative compression technique.

► **Lemma 3.3 (★).** *There is an algorithm that, given an instance (G, T, k) of MWNS and a set $S_{k+1} \subseteq V(G) \setminus T$ of size $k + 1$ such that S_{k+1} is a MWNS of (G, T) , runs in time $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ and outputs a solution of (G, T, k) (of size k) if it exists.*

Given Lemma 3.3, the proof of Theorem 1.1 follows by applying the standard technique of iterative compression. The formal proof can be found in the full version [17].

4 Constructing a near-separator avoiding a specified vertex

In this section we prove Theorem 1.3. Throughout the algorithm, we use the perspective provided by Proposition 2.7 that a MWNS is a set intersecting all T -cycles. Note that since $\{x\}$ is a MWNS for (G, T) , the set T must be an independent set. Before presenting the algorithm, we define some notations which we will use during the algorithm.

► **Definition 4.1** ($\mathcal{C}(v)$ and $\mathcal{C}_{\geq 1}(v)$). Given a graph G , terminal set $T \subseteq V(G)$, and a MWNS $\{x\} \subseteq V(G)$ of (G, T) , let \mathcal{F} be a rooted block-cut forest of $G - \{x\}$. Let $v \in V(\mathcal{F})$ be a cutvertex. Then we use $\mathcal{C}(v)$ to denote all the grandchildren (cutvertices) of v in the subtree \mathcal{F}_v , i.e., $\mathcal{C}(v) := \bigcup_{y \in \text{child}_{\mathcal{F}}(v)} \text{child}_{\mathcal{F}}(y)$. If v does not have a grandchild then $\mathcal{C}(v) := \emptyset$. We use $\mathcal{C}_{\geq 1}(v) \subseteq \mathcal{C}(v)$ to denote the cutvertices of $\mathcal{C}(v)$ such that for each vertex $c \in \mathcal{C}_{\geq 1}(v)$, the graph $G_c = G[V_G(\mathcal{F}_c)]$ contains a vertex $p \in N_G(x)$ such that there is a c - p path P in G_c which contains at least one terminal, i.e., $|V(P) \cap T| \geq 1$.

During the construction of an approximate x -avoiding MWNS, we will often make use of Definition 4.1 to keep track of which cutvertices have a pending subgraph attached that can reach a neighbor of x by a simple path containing a terminal. Such subpaths can be combined to form T -cycles. We often use the fact that, in an undirected graph G , it is possible to test in polynomial time whether there is a simple p - q path through a specified vertex t ; for example, by constructing a vertex-capacitated flow network in which t has a capacity of 2 and all other vertices a capacity of 1, and testing for a flow from $\{p, q\}$ to $\{t\}$.

Next, we prove some properties about the sets $\mathcal{C}(t)$ and $\mathcal{C}_{\geq 1}(t)$ defined above. We need these properties during the analysis phase (Section 4.2) of the blocker algorithm.

► **Proposition 4.2.** Given a graph G , terminal set $T \subseteq V(G)$, and a MWNS $\{x\} \subseteq V(G)$ of (G, T) , let \mathcal{F} be a rooted block-cut forest of $G - \{x\}$. Let $t \in T$ be a cutvertex of \mathcal{F} and let $\mathcal{C}(t)$ be the set of grandchildren of t in the subtree \mathcal{F}_t as defined in Definition 4.1. Then we have $\mathcal{C}(t) \cap T = \emptyset$.

Proof. Assume for a contradiction that there exists a vertex $t' \in \mathcal{C}(t) \cap T$. First, note that $t' \neq t$, as a cutvertex is present exactly once in a block-cut forest. Thus, we have $t' \in T \setminus \{t\}$. Let $B := \text{parent}_{\mathcal{F}}(t')$. Since $\{t', B\} \in E(\mathcal{F})$, we have $t' \in V(B)$ by definition of block-cut forest. Moreover, as B is the parent of t' and t' is a grandchild of t , we have $\{t, B\} \in E(\mathcal{F})$ and hence $t \in V(B)$. Note that B is a block in $G - \{x\}$ which contains two distinct terminals t, t' , a contradiction to Observation 2.2. ◀

► **Proposition 4.3.** Given a graph G , terminal set $T \subseteq V(G)$, and a MWNS $\{x\} \subseteq V(G)$ of (G, T) , let \mathcal{F} be a rooted block-cut forest of $G - \{x\}$. Let $t \in T$ be a cutvertex of \mathcal{F} and let $\mathcal{C}_{\geq 1}(t)$ be the subset of grandchildren of t in \mathcal{F}_t defined in Definition 4.1. Let B be a node of \mathcal{F}_t such that the graph $G[V_G(\mathcal{F}_B) \cup \{x\}]$ does not contain a T -cycle. Then the number of cutvertices below B in \mathcal{F}_B which also belong to the set $\mathcal{C}_{\geq 1}(t)$ is at most one, i.e., we have $|V_G(\mathcal{F}_B) \cap \mathcal{C}_{\geq 1}(t)| \leq 1$.

Proof. First of all, note that if B belongs to $\mathcal{C}(t)$ (see Definition 4.1 for the definition of $\mathcal{C}(t)$) or below in the subtree \mathcal{F}_t then the claim trivially holds, because in that case we have $|V_G(\mathcal{F}_B) \cap \mathcal{C}_{\geq 1}(t)| \leq 1$. Hence consider the case when either $B \in \text{child}_{\mathcal{F}}(t)$ or $B = t$, and assume for a contradiction that $|V_G(\mathcal{F}_B) \cap \mathcal{C}_{\geq 1}(t)| \geq 2$. Let c_1, c_2 be two distinct cutvertices in $V_G(\mathcal{F}_B) \cap \mathcal{C}_{\geq 1}(t)$. By definition of the set $\mathcal{C}_{\geq 1}(t)$ and the fact that $c_1, c_2 \in \mathcal{C}_{\geq 1}(t)$, we know that for each $i \in [2]$, the graph G_{c_i} contains a vertex $p_i \in N_G(x)$ such that there is a c_i - p_i path P_i in G_{c_i} containing a terminal t_i . Moreover, since $c_1 \neq c_2$, the paths P_1 and P_2 are vertex disjoint. Next, we do a case distinction based on whether $B \in \text{child}_{\mathcal{F}}(t)$ or $B = t$.

Case 1. When $B \in \text{child}_{\mathcal{F}}(t)$. Since $B \in \text{child}_{\mathcal{F}}(t)$ and by definition (of $\mathcal{C}_{\geq 1}(t)$) c_1, c_2 are grandchildren of t , we have $c_1, c_2 \in \text{child}_{\mathcal{F}}(B)$. Hence, we have $c_1, c_2 \in V_G(B)$. Next, we construct a cycle C in $G[V_G(\mathcal{F}_B) \cup \{x\}]$ as follows. Let $C := \{x, p_1\} \cdot P_1[p_1, c_1] \cdot R_{12}[c_1, c_2] \cdot P_2[c_2, p_2] \cdot \{p_2, x\}$, where R_{12} is a path between cutvertices $c_1, c_2 \in V_G(B)$ inside the block B . Note that the cycle C in $G[V_G(\mathcal{F}_B) \cup \{x\}]$ is simple and contains two distinct terminals $t_1, t_2 \in T$, a contradiction to the fact that there is no T -cycle in $G[V_G(\mathcal{F}_B) \cup \{x\}]$.

Case 2. When $B = t$. For $i \in [2]$, let B_i be the parent of c_i . Note that if $B_1 = B_2$ then similarly to Case 1, we can obtain a T -cycle in $G[V_G(\mathcal{F}_{B_1}) \cup \{x\}]$, which is also a T -cycle in the supergraph $G[V_G(\mathcal{F}_B) \cup \{x\}]$, again a contradiction to the fact that there is no T -cycle in $G[V_G(\mathcal{F}_B) \cup \{x\}]$. Hence assume that $B_1 \neq B_2$. Next, we show that even in this case we can obtain a T -cycle C in $G[V_G(\mathcal{F}_B) \cup \{x\}]$, yielding a contradiction. Indeed, we can use $C := \{x, p_1\} \cdot P_1[p_1, c_1] \cdot R_1[c_1, B] \cdot R_2[B, c_2] \cdot P_2[c_2, p_2] \cdot \{p_2, x\}$, where for $i \in [2]$, the path R_i is a path between vertices $c_i, B \in V_G(B_i)$ inside the block B_i .

Since both the above cases lead to a contradiction, this concludes the proof of Proposition 4.3. \blacktriangleleft

4.1 Algorithm

In this section we present a recursive algorithm $\text{Blocker}(G, T, x)$ to construct a set $S_x \subseteq V(G) \setminus (T \cup \{x\})$ such that S_x is a MWNS of (G, T) and $|S_x| \leq 14 \text{OPT}_x(G, T)$, where $\text{OPT}_x(G, T)$ is the cardinality of a minimum x -avoiding MWNS of (G, T) . The algorithm effectively takes a graph G , terminal set T , a vertex $x \in V(G) \setminus T$ (such that $\{x\}$ is a MWNS of (G, T)) as input, and computes a vertex set $Z \subseteq V(G) \setminus (T \cup \{x\})$ to hit certain types of T -cycles in G . It combines Z with the result of recursively computing a solution for $\text{Blocker}(G - Z, T, x)$. For ease of understanding, we present the algorithm step by step with interleaved comments in italic font, whenever required.

1. If the graph G does not contain a T -cycle on x then return $Z = \emptyset$.
2. Construct a rooted block-cut forest \mathcal{F} of $G - \{x\}$.
3. Choose a deepest node d in the block cut forest \mathcal{F} such that the graph $G[V_G(\mathcal{F}_d) \cup \{x\}]$ contains a T -cycle.
Note that such a vertex d exists as there is a T -cycle on x in the graph G while a simple cycle in G visits vertices from at most one tree $\mathcal{T} \in \mathcal{F}$.
4. Consider the following cases.
 - a. **If d is a cutvertex and $d \notin T$.** Let $Z := \{d\}$. Then return $(Z \cup \text{Blocker}(G - Z, T, x))$.
In this case, it is easy to observe that the set $Z = \{d\}$ is a MWNS of $(G[V_G(\mathcal{F}_d) \cup \{x\}], T)$ because d is a deepest node satisfying the conditions of Step 3.
 - b. **If d is a cutvertex and $d \in T$.** Let $\mathcal{C}_{\geq 1}(d)$ be the subset of grandchildren of d defined using Definition 4.1. Let $Z := \mathcal{C}_{\geq 1}(d)$ and return $(Z \cup \text{Blocker}(G - Z, T, x))$.
The set Z is a MWNS of $(G[V_G(\mathcal{F}_d) \cup \{x\}], T)$ by our choice of d , definition of the set $\mathcal{C}_{\geq 1}(d)$, the fact that a T -cycle contains at least 2 terminals, and for each block $B \in \text{child}_{\mathcal{F}}(d)$ we have $V(B) \cap (T \setminus \{d\}) = \emptyset$ due to Observation 2.2.
 - c. **If d is a block.**
 - Let $D_T := d - T$, i.e., $D_T := G[V(d) \setminus T]$. Note that the block d of $G - \{x\}$ contains at most 1 terminal due to Observation 2.2. In the case when $V(d) \cap T = \emptyset$, we have $D_T = d = G[V(d)]$. Let $\mathcal{C}^d := \text{child}_{\mathcal{F}}(d) \setminus T$ and partition \mathcal{C}^d as follows.
 - Let $\mathcal{C}_{\geq 2}^d \subseteq \mathcal{C}^d$ be the set such that for each (cut)vertex $c \in \mathcal{C}_{\geq 2}^d$ the graph $G_c := G[V_G(\mathcal{F}_c)]$ contains a vertex $p \in N_G(x)$ such that there is a c - p path P in G_c which contains at least 2 terminals, i.e., $|V(P) \cap T| \geq 2$.
 - Let $\mathcal{C}_1^d \subseteq \mathcal{C}^d \setminus \mathcal{C}_{\geq 2}^d$ be the subset of remaining vertices of \mathcal{C}^d such that for each vertex $c \in \mathcal{C}_1^d$ the graph G_c contains a vertex $p \in N_G(x)$ such that there is a c - p path P in G_c which contains 1 terminal, i.e., $|V(P) \cap T| = 1$.
 - Let $\mathcal{C}_0^d \subseteq \mathcal{C}^d \setminus (\mathcal{C}_{\geq 2}^d \cup \mathcal{C}_1^d)$ be the subset of remaining vertices of \mathcal{C}^d such that for each vertex $c \in \mathcal{C}_0^d$ the graph G_c contains a vertex $p \in N_G(x)$ such that there is a c - p path P in G_c .
 - Let $\mathcal{C}_\emptyset^d := \mathcal{C}^d \setminus (\mathcal{C}_{\geq 2}^d \cup \mathcal{C}_1^d \cup \mathcal{C}_0^d)$ be the remaining elements of \mathcal{C}^d .

28:10 On the Parameterized Complexity of Multiway Near-Separator

- Apply Gallai’s theorem ([7, Thm 9.2, Lemma 9.3], cf. [25, Thm 73.1]) on the graph D_T with $Q = \mathcal{C}_{\geq 2}^d \cup \mathcal{C}_1^d$ to obtain a maximum-cardinality family \mathcal{P}_Q of pairwise vertex disjoint Q -paths in D_T , along with a vertex set $Z_1 \subseteq V(D_T)$ of size at most $2|\mathcal{P}_Q|$ such that the graph $D_T - Z_1$ has no Q -path.
- Let $A := \mathcal{C}_{\geq 2}^d$ and $B := \mathcal{C}_0^d \cup (N_G(x) \cap V(d))$. Next, compute a minimum (A, B) -separator in the graph D_T using Edmonds-Karp algorithm [12] which outputs a vertex set $Z_2 \subseteq V(D_T)$.

Next, we do a case distinction based on whether or not the block d contains a terminal. Note that in the case when d does not contain a terminal then the set $(Z_1 \cup Z_2)$ hits all T -cycles of $G[V_G(\mathcal{F}_d) \cup \{x\}]$. On the other hand, when d contains a terminal there could still be a T -cycle in the graph $G[V_G(\mathcal{F}_d) \cup \{x\}] - (Z_1 \cup Z_2)$ (see the third figure of Figure 1). So our next steps are aimed at hitting those T -cycles (if any) that are not hit by the set $(Z_1 \cup Z_2)$.

- If $V(d) \cap T = \emptyset$, then let $Z_3, Z_4 := \emptyset$.
- Otherwise, there is a unique terminal in block $d \subseteq G - x$ since x is a MWNS for (G, T) . Let t be the terminal that belongs to the block d .
 - Let \mathcal{D} be the set containing connected components of $D_T - (Z_1 \cup Z_2)$ that contain at least one neighbor of t , i.e., for each connected component $D \in \mathcal{D}$, we have $V(D) \cap N(t) \neq \emptyset$.
 - Let $\mathcal{D}^* \subseteq \mathcal{D}$ be the set such that for each $D^* \in \mathcal{D}^*$, the connected component D^* contains a vertex from $Q = (\mathcal{C}_{\geq 2}^d \cup \mathcal{C}_1^d)$. More precisely, we have $|V(D^*) \cap Q| = 1$ for each D^* as the set Z_1 is hitting all Q -paths.
 - Let $V(\mathcal{D}^*) := \bigcup_{D^* \in \mathcal{D}^*} V(D^*)$ and define $Z_3 := (\mathcal{C}_{\geq 2}^d \cup \mathcal{C}_1^d) \cap V(\mathcal{D}^*)$.
Note that in the case when d contains a terminal t and $t \in \text{child}_{\mathcal{F}}(d)$, the way we have defined \mathcal{C}^d , it does not contain t . Hence, when $t \in \text{child}_{\mathcal{F}}(d)$ and the graph G_t has a vertex $p \in N_G(x)$ such that there is a t - p path P in G_t that contains at least one terminal other than t , i.e., $|V(P) \cap (T \setminus \{t\})| \geq 1$, there could still be T -cycles in $G[V_G(\mathcal{F}_d)] - \bigcup_{i=1}^3 Z_i$ (see the last figure of Figure 1). So our next step is to hit all T -cycles (if any) containing the t - p path P . Recall $\mathcal{C}_{\geq 1}(t)$ from Definition 4.1.
 - * If $t \in \text{child}_{\mathcal{F}}(d)$ and the graph G_t has a vertex $p \in N_G(x)$ such that there is a t - p path P in G_t with $|V(P) \cap T| \geq 2$, then let $Z_4 := \mathcal{C}_{\geq 1}(t)$.
 - * Otherwise, define $Z_4 := \emptyset$.

Finally, we try to break any interaction between vertices of $V_G(\mathcal{F}_d)$ and vertices of $V_G(\mathcal{F}) \setminus V_G(\mathcal{F}_d)$ by adding the parent of d into the hitting set. But note that in the case when $\text{parent}_{\mathcal{F}}(d) \in T$, we can not add it to the hitting set $Z \subseteq V(G) \setminus (T \cup \{x\})$.

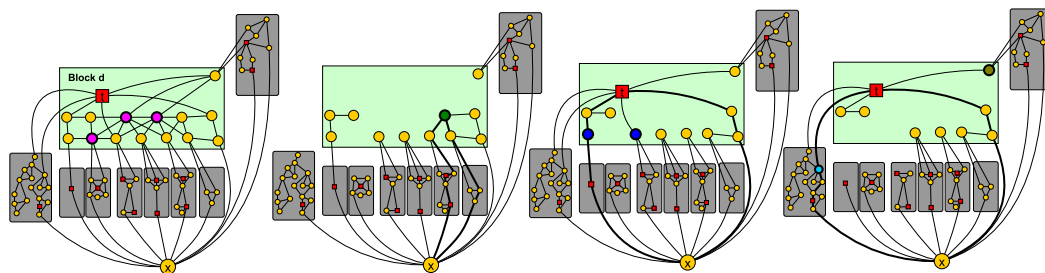
- If $\text{parent}_{\mathcal{F}}(d) \in T$, then let $Z_5 := \emptyset$.
- Otherwise, $Z_5 := \text{parent}_{\mathcal{F}}(d)$.

Let $Z := \bigcup_{i=1}^5 Z_i$ and return($Z \cup \text{Blocker}(G - Z, T, x)$).

This concludes the description of the algorithm. Summarizing, its main structure is to define a vertex set $Z \subseteq V(G) \setminus (T \cup \{x\})$ to break T -cycles which are *lowest* in the block-cut tree, include that set Z in the approximate solution, and complete the solution by recursively solving the problem on $G - Z$.

4.2 Analysis

The following lemma forms the heart of Theorem 1.3. It says that if the above procedure adds a set Z during the construction of the approximate solution S_x , then the optimum value of the remaining instance decreases by at least $\frac{|Z|}{14}$.



■ **Figure 1** Illustration of Step 4c. The leftmost figure shows the original graph G where terminals are represented by red squares and the pink vertices of block d represent the vertices of Z_1 . The second figure shows the construction of the set Z_2 in the graph $G - (Z_1 \cup \{t\})$ where the green vertex represents the vertex of Z_2 . It also shows a T -cycle represented by thick edges. The third figure shows a T -cycle (represented by thick edges) which appears after putting the terminal t back and the blue vertices represent the vertices of Z_3 . The last figure illustrates the construction of the set Z_4 and Z_5 where the cyan vertex and olive vertex represent the vertex of Z_4 and Z_5 , respectively.

► **Lemma 4.4.** *If a single iteration of the algorithm on input (G, T, x) yields the set Z , then $\text{OPT}_x(G - Z, T) \leq \text{OPT}_x(G, T) - \frac{|Z|}{14}$, where $\text{OPT}_x(G, T)$ and $\text{OPT}_x(G - Z, T)$ are the cardinalities of a minimum x -avoiding MWNS of (G, T) and $(G - Z, T)$, respectively.*

Proof. Let $S_x^* \subseteq V(G)$ be a minimum x -avoiding MWNS of (G, T) . If the algorithm stops before Step (3) then $Z = \emptyset$. Hence the inequality of the lemma trivially holds. Therefore assume that the algorithm reaches Step 4. Let d be the deepest node selected by the algorithm in Step (3) to compute Z . Let $\hat{S} := S_x^* \setminus V_G(\mathcal{F}_d)$. Note that $\hat{S} \cap (T \cup \{x\}) = \emptyset$. Next, we observe the following properties about the set Z computed in Step 4.

▷ **Claim 4.5 (★).** Suppose the algorithm reaches Step 4 and computes the set $Z \subseteq V_G(\mathcal{F}_d) \setminus T$. Then Z is a MWNS of $(G_d + x, T)$, where $G_d + x = G[V_G(\mathcal{F}_d) \cup \{x\}]$.

Proof sketch. By choice of d , each T -cycle of $G_d + x$ contains a vertex from block d , or the cutvertex d itself. Since x is a MWNS for (G, T) , each T -cycle also contains x . The sets Z_1, \dots, Z_4 added to Z in the algorithm ensure different types of T -cycles of $G_d + x$ are broken: Z_1 covers all T -cycles consisting of two paths between x and d , each containing at least one terminal; Z_2 covers T -cycles consisting of one path between x and d containing two or more terminals, and another such path containing no terminals; the sets Z_3 and Z_4 cover T -cycles that go through a terminal in d (if there is one), as explained in the algorithm. ◁

▷ **Claim 4.6 (★).** Suppose the algorithm reaches Step 4 and computes the set $Z \subseteq V_G(\mathcal{F}_d) \setminus T$. Then any T -cycle in $G - (Z \cup \hat{S})$ contains a vertex of $V_G(\mathcal{F}_d)$ (i.e., a vertex from G_d) and a vertex from $V_G(\mathcal{F}) \setminus (V_G(\mathcal{F}_d))$ (i.e., a vertex outside $G_d + x$).

Proof sketch. Since \hat{S} contains all vertices of the solution S_x^* except those occurring in a block of the subtree \mathcal{F}_d of the block-cut tree, any T -cycle disjoint from \mathcal{S} was intersected by S_x^* in a vertex of $V_G(\mathcal{F}_d)$ and therefore uses a vertex of the latter set. On the other hand, since the previous claim shows that Z hits all the T -cycles which live in $G_d + x$, a T -cycle disjoint from Z has to use a vertex outside $V_G(\mathcal{F}_d)$. ◁

▷ **Claim 4.7 (★).** Suppose the algorithm reaches Step 4 and computes the set $Z \subseteq V_G(\mathcal{F}_d) \setminus T$. Then any minimum x -avoiding MWNS S_x^* of (G, T) satisfies $|S_x^* \cap V_G(\mathcal{F}_d)| \geq \max\{1, \frac{|Z|-2}{6}\}$.

28:12 On the Parameterized Complexity of Multiway Near-Separator

Proof sketch. Any x -avoiding MWNS S_x^* contains a vertex from $V_G(\mathcal{F}_d)$ because there is a T -cycle in $G_d + x$, by choice of d , which explains why the intersection is nonempty. The fact that S_x^* contains at least $\frac{|Z|-2}{6}$ vertices from $V_G(\mathcal{F}_d)$ can be seen as follows. We have $|Z_4|, |Z_5| \leq 1$ by definition, so the largest Z_i of Z_1, Z_2, Z_3 has at least $\frac{|Z|-2}{3}$ vertices. Any solution contains at least $|Z_i|/2$ vertices from $V_G(\mathcal{F}_d)$ because the covering/packing duality for the three types of separators used to define Z_1, Z_2, Z_3 ensures, for each of these sets, that to hit all the T -cycles of the corresponding form, at least $|Z_i|/2$ vertices are needed. For example, each Q -path P in the family \mathcal{P}_Q obtained during the construction of Z_1 yields a T -cycle when combined with paths from endpoints of P (which are cutvertices in Q) to neighbors of x in two different subtrees of the block-cut forest \mathcal{F} , showing that $S_x^* \cap V_G(\mathcal{F}_d) \geq |\mathcal{P}_Q| \geq |Z_1|/2$. \triangleleft

Next, we show that if a minimum x -avoiding MWNS S_x^* of (G, T) contains exactly 1 vertex from the subtree \mathcal{F}_d then the set $\hat{S} := S_x^* \setminus V_G(\mathcal{F}_d)$ is a MWNS of $(G - Z, T)$.

\triangleright **Claim 4.8 (★).** If $|S_x^* \cap V_G(\mathcal{F}_d)| = 1$ then $\hat{S} = S_x^* \setminus V_G(\mathcal{F}_d)$ is a MWNS of $(G - Z, T)$.

Proof sketch. If $|S_x^* \cap V_G(\mathcal{F}_d)| = 1$, then from the T -cycle in $G_d + x$ which we know to exist, the set S_x^* contains at most one vertex. In the most crucial case that the d is a block whose parent is a terminal, this means that S_x^* cannot break all paths from x through blocks in the subtree \mathcal{F}_d to the parent of d . The only types of connections that the set Z does not break, and which can be part of T -cycles in G , can be shown to be precisely paths from a neighbor of x to d in G_d that do not contain a terminal. But if $|S_x^* \cap V_G(\mathcal{F}_d)| = 1$, then S_x^* does not break all such paths either. By a rerouting argument, this allows us to show that updating S^* by replacing $S_x^* \cap V_G(\mathcal{F}_d)$ with Z gives a valid x -avoiding MWNS, which is equivalent to saying that \hat{S} is a MWNS of $(G - Z, T)$. \triangleleft

The following claim shows that if the set \hat{S} is not a MWNS of $(G - Z, T)$, then we can find a vertex \hat{c} such that the set obtained after adding \hat{c} to the set \hat{S} is a MWNS of $(G - Z, T)$.

\triangleright **Claim 4.9 (★).** If \hat{S} is not a MWNS of $(G - Z, T)$ then there exists a vertex $\hat{c} \in V(G) \setminus (T \cup \{x\})$ such that $\hat{S} \cup \{\hat{c}\}$ is a MWNS of $(G - Z, T)$.

Proof sketch. The proof consists of a delicate argument which essentially says that this situation only happens when d is a block whose parent is a terminal, and there is a cutvertex \hat{c} close to block d in the block-cut forest which combines with Z to break all paths in $G - \{x\}$ from $N_G(x) \cap V_G(\mathcal{F}_d)$ to vertices of $N_G(x) \setminus V_G(\mathcal{F}_d)$, and therefore breaks all T -cycles intersecting $V_G(\mathcal{F}_d)$. \triangleleft

The proof of Lemma 4.4 follows from the preceding statements by formula manipulation: whenever the approximation algorithm chooses a set Z , an optimal solution chooses at least $|Z|/14$ vertices from $V_G(\mathcal{F}_d)$. The remainder of the proof is given in the full version [17]. \blacktriangleleft

Using Lemma 4.4, an easy induction shows that the algorithm indeed computes a 14-approximation. As each iteration can be implemented in polynomial time, this leads to a proof of Theorem 1.3. The details are given in the full version [17].

5 Bounding the number of terminals

Our main goal in this section is to prove Theorem 1.2. Intuitively, there are two distinct ways in which a connected component admitting a small solution can contain a large number of terminals: either there can be a star-like structure of blocks containing a terminal joined at a single cutvertex, or there exists a long path in the block-cut tree containing many blocks with a terminal. After applying reduction rules to attack both kinds of situations, we give a final reduction rule to bound the number of relevant connected components that contain a terminal, which will lead to the desired bound on the number of terminals. Due to space constraints, the safeness of the reduction rules we present here is deferred to the full version [17].

► **Reduction Rule 1.** *Let (G, T, k) be an instance of MWNS, and let $t \in T$ be a terminal such that for any other terminal $t' \in T \setminus \{t\}$, there do not exist 2 internally vertex-disjoint t - t' paths in G , i.e., the terminal t is nearly-separated. Then remove t from the set T . The new instance is $(G, T \setminus \{t\}, k)$.*

Next, we have the following general reduction rule.

► **Reduction Rule 2.** *Let (G, T, k) be an instance of MWNS. Suppose there exist 2 non-terminal vertices $x, y \in V(G) \setminus T$ such that $G - \{x, y\}$ has a connected component D satisfying the following conditions:*

- (a) $|V(D) \cap T| \geq 3$,
- (b) the graph $G[V(D) \cup \{x, y\}]$ has no T -cycle, and
- (c) there is an x - y path in $G[V(D) \cup \{x, y\}]$ containing distinct terminals $t_1, t_2 \in T$.

Then turn any terminal of D which is not t_1, t_2 into a non-terminal. Formally, let $t \in V(D) \cap (T \setminus \{t_1, t_2\})$, then $(G, T \setminus \{t\}, k)$ is a new instance of MWNS.

Next, we show that given an instance (G, T, k) of MWNS and a 1-redundant MWNS $S^* \subseteq V(G) \setminus T$ of (G, T) , in polynomial-time we can obtain an equivalent instance (G, T', k) such that the number of connected components of $G - S^*$ which contain at least one terminal from T' is bounded by $\mathcal{O}(|S^*|^2 \cdot k)$. Towards this, we apply the following marking scheme with respect to the set S^* to mark $\mathcal{O}(|S^*|^2 \cdot k)$ connected components of $G - S^*$.

► **Marking Scheme 1.** *Let (G, T, k) be an instance of MWNS, and let $S^* \subseteq V(G)$ be a 1-redundant MWNS of (G, T) . For each pair of distinct vertices $x, y \in S^*$, we greedily mark $(k + 2)$ connected components $C_{i_1}^{x,y}, \dots, C_{i_{k+2}}^{x,y}$ of $G - S^*$ such that for each $m \in [k + 2]$, the connected component $C_{i_m}^{x,y}$ contains two vertices (not necessarily distinct) $p_m \in N_G(x)$ and $q_m \in N_G(y)$ such that there is a p_m - q_m path P_m (possibly of length 0) inside the connected component $C_{i_m}^{x,y}$ which contains at least one terminal, i.e., we have $|V(P_m) \cap T| \geq 1$. If there are fewer than $k + 2$ such components, we simply mark all of them.*

We have the following reduction rule based on the above marking scheme. It requires a 1-redundant MWNS to be known. In the context of Theorem 1.2, where we are given a MWNS S of (G, T, k) , we can exploit Theorem 1.3 to obtain a 1-redundant MWNS of size $\mathcal{O}(|S| \cdot k)$ in polynomial time: for each $x \in S$, if an x -avoiding MWNS S_x in $G - (S \setminus \{x\})$ has size more than $14k$ then all solutions of (G, T, k) contain x and we may safely remove x and decrease k ; otherwise, we can add S_x to S to ensure all T -cycles intersecting x are hit twice, without blowing up the size of S too much. The details are given in the full version [17].

► **Reduction Rule 3.** Let (G, T, k) be an instance of MWNS, and let $S^* \subseteq V(G) \setminus T$ be a 1-redundant MWNS of (G, T) . Let \mathcal{C}_M be the set of connected components of $G - S^*$ marked by Marking Scheme 1 with respect to the set S^* . Let $T' := T \cap (\bigcup_{C \in \mathcal{C}_M} V(C))$. If $T \setminus T' \neq \emptyset$ then we convert the terminals of $T \setminus T'$ to non-terminals. The new instance is (G, T', k) .

A delicate analysis shows that applying these reduction rules indeed leads to an equivalent instance with the desired bound on the number of terminals, which leads to a proof of Theorem 1.2. The details are given in the full version [17].

6 Conclusions

In this paper we initiated the study of the MULTIWAY NEAR-SEPARATOR problem, a generalization of MULTIWAY CUT focused on reducing the connectivity between each pair of terminals. We developed reduction rules to reduce the number of terminals in an instance, aided by a constant-factor approximation algorithm for the problem of finding a near-separator avoiding a given vertex x in an instance for which $\{x\}$ is a near-separator. Our work leads to several follow-up questions. First of all, one could consider extending the notion of near-separation to allow for larger (but bounded) connectivity between terminal pairs in the resulting instance, for example by requiring that in the graph $G - S$, for each pair of distinct terminals t_i, t_j there is a vertex set of size at most c whose removal separates t_i and t_j . The setting we considered here is that of $c = 1$, which allows us to understand the structure of the problem based on the block-cut forest of $G - S$. For $c = 2$ it may be feasible to do an analysis based on the decomposition of $G - S$ into triconnected components, but for larger values of c the structure may become significantly more complicated. One could also consider a generic setting (see [1]) where for each pair of terminals t_i, t_j , some threshold $f(t_i, t_j) = f(t_j, t_i)$ is specified such that in $G - S$ there should be a set of at most $f(t_i, t_j)$ nonterminals whose removal separates t_i from t_j . Note that if the values of f are allowed to be arbitrarily large, this generalizes MULTICUT. Is the resulting problem fixed-parameter tractable parameterized by k ?

Another direction for future work lies in the development of a polynomial kernel. For the MULTIWAY CUT problem with deletable terminals, as well as the setting with undeletable but constantly many terminals, a polynomial kernel is known based on matroid techniques [18]. Does the variant of MULTIWAY NEAR-SEPARATOR with deletable terminals admit a polynomial kernel?

References

- 1 Siddharth Barman and Shuchi Chawla. Region growing for multi-route cuts. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 404–418. SIAM, 2010. doi:10.1137/1.9781611973075.34.
- 2 Václav Blazej, Pratibha Choudhary, Dusan Knop, Jan Matyás Kristan, Ondrej Suchý, and Tomás Valla. Constant factor approximation for tracking paths and fault tolerant feedback vertex set. In Jochen Könemann and Britta Peis, editors, *Approximation and Online Algorithms - 19th International Workshop, WAOA 2021, Lisbon, Portugal, September 6-10, 2021, Revised Selected Papers*, volume 12982 of *Lecture Notes in Computer Science*, pages 23–38. Springer, 2021. doi:10.1007/978-3-030-92702-8_2.
- 3 Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. *SIAM J. Comput.*, 47(1):166–207, 2018. doi:10.1137/140961808.

- 4 Karl Bringmann, Danny Hermelin, Matthias Mnich, and Erik Jan van Leeuwen. Parameterized complexity dichotomy for steiner multicut. *J. Comput. Syst. Sci.*, 82(6):1020–1043, 2016. doi:10.1016/j.jcss.2016.03.003.
- 5 Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009. doi:10.1007/s00453-007-9130-6.
- 6 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM J. Comput.*, 45(4):1171–1229, 2016. doi:10.1137/15M1032077.
- 7 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 8 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *ACM Trans. Comput. Theory*, 5(1):3:1–3:11, 2013. doi:10.1145/2462896.2462899.
- 9 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Subset feedback vertex set is fixed-parameter tractable. *SIAM J. Discret. Math.*, 27(1):290–309, 2013. doi:10.1137/110843071.
- 10 E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiway cuts (extended abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 241–251, New York, NY, USA, 1992. Association for Computing Machinery. doi:10.1145/129712.129736.
- 11 Reinhard Diestel. *Graph Theory, 5th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2017. doi:10.1007/978-3-662-53622-3.
- 12 Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, April 1972. doi:10.1145/321694.321699.
- 13 L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.
- 14 Petr A. Golovach and Dimitrios M. Thilikos. Clustering to given connectivities. In Bart M. P. Jansen and Jan Arne Telle, editors, *14th International Symposium on Parameterized and Exact Computation, IPEC 2019, September 11-13, 2019, Munich, Germany*, volume 148 of *LIPICs*, pages 18:1–18:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.IPEC.2019.18.
- 15 Sylvain Guillemot. FPT algorithms for path-transversal and cycle-transversal problems. *Discret. Optim.*, 8(1):61–71, 2011. doi:10.1016/j.disopt.2010.05.003.
- 16 Bart M. P. Jansen and Marcin Pilipczuk. Approximation and kernelization for chordal vertex deletion. *SIAM J. Discret. Math.*, 32(3):2258–2301, 2018. doi:10.1137/17M112035X.
- 17 Bart M. P. Jansen and Shivesh K. Roy. On the parameterized complexity of multiway near-separator, 2023. arXiv:2310.04332.
- 18 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.
- 19 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Reducing CMSO model checking to highly connected graphs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPICs*, pages 135:1–135:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICALP.2018.135.
- 20 Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006. doi:10.1016/j.tcs.2005.10.007.
- 21 Dániel Marx, Barry O’Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Trans. Algorithms*, 9(4):30:1–30:35, 2013. doi:10.1145/2500119.

- 22 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM J. Comput.*, 43(2):355–388, 2014. doi:10.1137/110855247.
- 23 Igor Razgon. Large isolating cuts shrink the multiway cut. *CoRR*, abs/1104.5361, 2011. arXiv:1104.5361.
- 24 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. doi:10.1016/j.orl.2003.10.009.
- 25 A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.
- 26 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, 2010. doi:10.1145/1721837.1721848.
- 27 Mingyu Xiao. Simple and improved parameterized algorithms for multiterminal cuts. *Theory Comput. Syst.*, 46(4):723–736, 2010. doi:10.1007/s00224-009-9215-5.

A Additional preliminaries

Graphs. An x - y path P and a q - r path Q are said to be internally vertex-disjoint if they do not share a common internal-vertex, i.e., we have $(V(P) \setminus \{x, y\}) \cap (V(Q) \setminus \{q, r\}) = \emptyset$. Given $X, Y \subseteq V(G)$, a path (v_1, \dots, v_k) in G is called an $X - Y$ path if $v_1 \in X$ and $v_k \in Y$.

► **Theorem A.1** (Menger's theorem, [11], Theorem 3.3.1). *Let G be a graph and $X, Y \subseteq V(G)$ be subsets of vertices such that $X \cap Y = \emptyset$ and there does not exist an edge $\{x, y\} \in E(G)$ for any $x \in X$ and $y \in Y$. Then the minimum number of vertices separating X from Y is equal to the maximum number of vertex-disjoint $X - Y$ paths in G .*

A cutvertex in a graph is a vertex v whose removal increases the number of connected components. A graph is 2-connected if it has at least three vertices and does not contain any cutvertex. A *block* of a graph G is a maximal connected subgraph B of G such that B does not have a cutvertex. Each block of G is either a 2-connected subgraph of G , a single edge, or an isolated vertex.

► **Definition A.2** (Block-cut graph, [11], §3.1). *Given a graph G , let A be the set of cutvertices of G , and let \mathcal{B} be the set of its blocks. The block-cut graph G' of G is the bipartite graph with partite sets A and \mathcal{B} , and for each cut-vertex $a \in A$, for each block $B \in \mathcal{B}$, there is an edge $\{a, B\} \in E(G')$ if $a \in V(B)$.*

We also need the following simple but useful properties of block-cut graphs.

► **Lemma A.3** ([11], Lemma 3.1.4). *The block-cut graph of a connected graph is a tree.*

► **Observation A.4.** *Consider an edge e of the block-cut tree \mathcal{T} of a connected graph G , let v be the unique cutvertex incident on e , let $\mathcal{T}_1, \mathcal{T}_2$ be the two trees of $\mathcal{T} - \{e\}$, and let $Y_i := V_G(\mathcal{T}_i)$ be the vertices of G occurring in blocks of \mathcal{T}_i , for $i \in [2]$. Then all paths from a vertex of $Y_1 \setminus \{v\}$ to a vertex of $Y_2 \setminus \{v\}$ in G intersect v .*

► **Observation A.5.** *Let G be a graph and $T \subseteq V(G)$ be a set of terminals such that $V(G) \setminus T \neq \emptyset$. If no block of G contains two or more terminals, then for each pair $t_i, t_j \in T$ of distinct terminals there exists a vertex $v \in V(G) \setminus T$ such that v_i and v_j belong to different connected components of $G - \{v\}$.*

► **Observation A.6.** *For any positive integer ℓ , if there are ℓ cycles on x which are $(T \cup \{x\})$ -disjoint in graph G , then any x -avoiding MWNS of (G, T) contains at least ℓ vertices: at least one distinct non-terminal from each cycle.*

► **Proposition A.7.** *Let G , be a graph, B a block in G , and consider three distinct vertices $p, q, t \in V(B)$. There is a p - t path P and q - t path Q inside block B such that $V(P) \cap V(Q) = \{t\}$.*

Proof. Since B is a block containing at least three vertices, it is a 2-connected graph. We first add a new vertex $v_{p,q}$ and edges $\{v_{p,q}, p\}$, $\{v_{p,q}, q\}$ to block B to obtain a new graph B' . Observe that B' is still a 2-connected graph, as the above modification can be seen as adding a new B -path $v, v_{p,q}, q$ to B and adding a B -path to a 2-connected graph results in a 2-connected graph (see [11, Proposition 3.1.1]). Next, we apply Menger's theorem with $X = \{v_{p,q}\}$ and $Y = \{t\}$ in the (2-connected) graph B' , which ensures that there are 2 internally vertex-disjoint $v_{p,q}$ - t paths P' and Q' in B' . By construction of B' and the fact that P' and Q' are internally vertex-disjoint $v_{p,q}$ - t paths in B' , we know that one of the $v_{p,q}$ - t paths (assume w.l.o.g. P') contains vertex p of block B , whereas the other $v_{p,q}$ - t path Q' contains vertex q of block B . Hence the paths $P = P'[p, t]$ and $Q = Q'[q, t]$ are the desired paths with $V(P) \cap V(Q) = \{t\}$. ◀

This following observation follows from Proposition A.7, as explained below.

► **Observation A.8.** *Let \mathcal{F} be the block-cut forest of a graph G . Suppose there is an x - y path \mathcal{P} in \mathcal{F} between cutvertices x, y of \mathcal{F} . Then for any distinct vertices $v_1, v_2 \in V(G)$ from 2 distinct blocks on \mathcal{P} , the graph G has an x - y path P_{12} through v_1, v_2 , i.e., $v_1, v_2 \in V(P_{12})$.*

We can construct the desired path P_{12} by concatenating paths inside each block B on the x - y path in \mathcal{F} , where each path connects the cutvertex p connecting to the previous block, with the cutvertex q connecting through the next block. If a forced v_i is chosen from block B , we can use Proposition A.7 to obtain a p - q path through $t = v_i$.

Parameterized algorithms. A *parameterized problem* L is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is called *fixed parameter tractable* (FPT) if there exists an algorithm which for every input $(x, k) \in \Sigma^* \times \mathbb{N}$ correctly decides whether $(x, k) \in L$ in $f(k) \cdot |x|^{\mathcal{O}(1)}$ time, where $f: \mathbb{N} \rightarrow \mathbb{N}$ is a computable function. We refer to [7] for more background on parameterized algorithms.

Given a graph G and $X, Y \subseteq V(G)$, a set $S \subseteq V(G)$ is called an (X, Y) -separator if there is no x - y path in $G - S$ for any $x \in X \setminus S$ and $y \in Y \setminus S$. The notion of important separator was defined by Marx [20] to obtain an FPT algorithm for MULTIWAY CUT. He also derived several useful properties.

► **Definition A.9** (Important separator, [7], Definition 8.49). *Let G be an undirected graph, let $X, Y \subseteq V(G)$ be two sets of vertices, and let $V^\infty \subseteq V(G)$ be a set of undeletable vertices. Let $S \subseteq V(G) \setminus V^\infty(G)$ be an (X, Y) -separator and let R be the set of vertices reachable from $X \setminus S$ in $G - S$. We say that S is an important (X, Y) -separator if it is inclusion-wise minimal and there is no (X, Y) -separator $S' \subseteq V(G) \setminus V^\infty(G)$ with $|S'| \leq |S|$ such that $R \subset R'$, where R' is the set of vertices reachable from $X \setminus S'$ in $G - S'$.*

► **Proposition A.10** ([7], Proposition 8.50). *Let G be an undirected graph and $X, Y \subseteq V(G)$ be two sets of vertices, and let $V^\infty \subseteq V(G)$ be a set of undeletable vertices. Let $\hat{S} \subseteq V(G) \setminus V^\infty(G)$ be an (X, Y) -separator and let \hat{R} be the set of vertices reachable from $X \setminus \hat{S}$ in $G - \hat{S}$. Then there is an important (X, Y) -separator $S' = N_G(\hat{R}) \subseteq V(G) \setminus V^\infty(G)$ such that $|S'| \leq |\hat{S}|$ and $\hat{R} \subseteq R'$.*

► **Theorem A.11** ([7], Theorem 8.51). *Let $X, Y \subseteq V(G)$ be two sets of vertices in an undirected graph G , let $k \geq 0$ be an integer, and let \mathcal{S}_k be the set of all (X, Y) -important separators of size at most k . Then $|\mathcal{S}_k| \leq 4^k$ and \mathcal{S}_k can be constructed in time $\mathcal{O}(|\mathcal{S}_k| \cdot k^2 \cdot (n + m))$.*

B Hardness proof for Multiway Near-Separator

► **Lemma B.1.** *MULTIWAY NEAR-SEPARATOR is NP-hard.*

Proof. We give a polynomial-time reduction from MULTIWAY SEPARATOR to MWNS. The MULTIWAY SEPARATOR problem is formally defined as follows.

MULTIWAY SEPARATOR (MWS)	Parameter: k
Input: An undirected graph G , terminal set $T \subseteq V(G)$, and a positive integer k .	
Question: Is there a set $S \subseteq V(G) \setminus T$ with $ S \leq k$ such that there does not exist a pair of distinct terminals $t_i, t_j \in T$ for which there is a t_i - t_j path in $G - S$?	

MULTIWAY SEPARATOR is NP-hard [10]. Given an instance (G, T, k) of MWS we describe the construction of an instance (G', T, k) of MWNS; it will be easy to see that it can be carried out in polynomial time. Consider an arbitrary ordering $t_1, t_2, \dots, t_{|T|}$ of the set T . We construct the graph G' such that (G', T, k) is a YES-instance of MWNS if and only if (G, T, k) is a YES-instance of MWS.

We begin with $G' := G$. Then for each $i \in [|T| - 1]$, we create a vertex w_i in G' , and insert edges $\{t_i, w_i\}$ and $\{w_i, t_{i+1}\}$ in G' . This completes the construction of G' .

Next, we prove that (G, T, k) is a YES-instance of MWS if and only if (G', T, k) is a YES-instance of MWNS. In the forward direction, assume that (G, T, k) is a YES-instance of MWS, and let $S \subseteq V(G) \setminus T$ be a MWS of (G, T, k) . Note that by definition of MWS, for each pair of distinct terminals $t_i, t_j \in T$, there is no t_i - t_j path in $G - S$. Since the transformation into G' consists of adding degree-2 vertices connecting consecutive terminals, any pair of distinct terminals $t_i, t_j \in T$ with $i < j$ can be separated in $G' - S$ by removing the vertex w_i . Hence the same set $S \subseteq V(G') \setminus T$ is also a MWNS of (G', T, k) .




In the reverse direction, assume that (G', T, k) is a YES-instance of MWNS and let $S' \subseteq V(G') \setminus T$ be a solution. Let $W := \bigcup_{i \in [|T| - 1]} \{w_i\}$ be the set of newly added vertices in G' . In the case when $S' \cap W = \emptyset$, it is easy to see that the same set S' is also a MWS of (G, T, k) : because if S is not a MWS of (G, T, k) then it implies that there is a pair of distinct terminals $t_i, t_j \in T$ connected by a t_i - t_j path say P in $G - S$. By construction of G' , there is also a unique t_i - t_j path say P' in $G'[W \cup T]$. Note that the paths P and P' are T -disjoint, and neither P nor P' intersects the set S' , a contradiction to the fact that S' is a solution of (G', T, k) since the paths witness that t_i, t_j cannot be separated by removing a single non-terminal.

Hence we need to show how to prove the case when $S' \cap W \neq \emptyset$. In this case, note that due to Lemma 3.1, there is a terminal $t \in T$ and a non-terminal $x \in V(G') \setminus T$ such that the set $S' \cup \{x\}$ is a $(t, T \setminus \{t\})$ -separator. So by applying Lemma 3.1 at most $|T| - 1$ times we obtain a set $S^* \subseteq V(G') \setminus T$ such that the set S^* is a MWS of (G', T) . Moreover, note that $|S^*| \leq |S'| + (|T| - 1)$ because in each step of Lemma 3.1 we add at most 1 additional vertex to separate a terminal. Next, we obtain a new set $\hat{S} := S^* \setminus W$. Now, it is easy to observe that the set $\hat{S} \subseteq V(G') \setminus T$ is a solution of (G', T, k) such that $\hat{S} \cap W = \emptyset$ thus (like in the previous case) we have the property that the set \hat{S} is also a MWS of (G, T, k) . Hence, (G, T, k) is a YES-instance of MWS. ◀

Sunflowers Meet Sparsity: A Linear-Vertex Kernel for Weighted Clique-Packing on Sparse Graphs

Bart M. P. Jansen   

Eindhoven University of Technology, The Netherlands

Shivesh K. Roy   

Eindhoven University of Technology, The Netherlands

Abstract

We study the kernelization complexity of the WEIGHTED H -PACKING problem on sparse graphs. For a fixed connected graph H , in the WEIGHTED H -PACKING problem the input is a graph G , a vertex-weight function $w: V(G) \rightarrow \mathbb{N}$, and positive integers k, t . The question is whether there exist k vertex-disjoint subgraphs H_1, \dots, H_k of G such that H_i is isomorphic to H for each $i \in [k]$ and the total weight of these $k \cdot |V(H)|$ vertices is at least t . It is known that the (unweighted) H -PACKING problem admits a kernel with $\mathcal{O}(k^{|V(H)|-1})$ vertices on general graphs, and a linear kernel on planar graphs and graphs of bounded genus. In this work, we focus on case that H is a clique on $h \geq 3$ vertices (which captures TRIANGLE PACKING) and present a linear-vertex kernel for WEIGHTED K_h -PACKING on graphs of bounded expansion, along with a kernel with $\mathcal{O}(k^{1+\varepsilon})$ vertices on nowhere-dense graphs for all $\varepsilon > 0$. To obtain these results, we combine two powerful ingredients in a novel way: the Erdős-Rado Sunflower lemma and the theory of sparsity.

2012 ACM Subject Classification Mathematics of computing \rightarrow Graph algorithms; Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Packing and covering problems

Keywords and phrases kernelization, weighted problems, graph packing, sunflower lemma, bounded expansion, nowhere dense

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.29

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803421, ReduceSearch).



1 Introduction

Packing and covering problems form an important area in the study of (algorithmic) graph theory [12, 25, 32, 35, 39, 47, 48, 56]. These problems have also been actively studied from the kernelization viewpoint [1, 3, 9, 16, 24, 43, 44, 52]. Roughly speaking, kernelization is a formalization of polynomial-time preprocessing aimed at compressing the instance size in terms of a complexity parameter (see Definition 4 for a formal definition). It is well-known that a decidable parameterized problem has a kernelization algorithm if and only if it is fixed-parameter tractable (FPT) [15]. Having an FPT algorithm for the problem implies that there exists a kernel, but the size of the kernel can be exponential in the parameter. Hence, finding a polynomial (or even linear) kernel is an active area of research in parameterized complexity [1, 2, 5, 6, 10, 11, 13, 16, 26, 27, 37, 45, 46, 53, 55].

For a fixed graph H , the H -PACKING problem asks, given a graph G and a positive integer k , whether there are k vertex-disjoint subgraphs H_1, \dots, H_k of G such that H_i is isomorphic to H for each $i \in [k]$. It is known that H -PACKING is NP-hard whenever H has a connected component on at least three vertices [42]. For $H = K_3$ the problem is equivalent to the well-known TRIANGLE PACKING problem. An application of the sunflower lemma due



© Bart M. P. Jansen and Shivesh K. Roy;

licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 29; pp. 29:1–29:13

Leibniz International Proceedings in Informatics



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

to Erdős and Rado [22] gives a $\mathcal{O}(k^{|V(H)|})$ vertex-kernel for H -PACKING. This bound was improved to $\mathcal{O}(k^{|V(H)|-1})$ by Abu-Khazam [1]. For restricted graph classes, such as planar graphs and graphs of bounded genus, H -PACKING is known to admit a linear kernel [11, §8.4]. Very recently, the problem was shown to have kernels with $\mathcal{O}(k^{1+\varepsilon})$ vertices and edges (for every $\varepsilon > 0$) on every class of *nowhere dense* graphs [4, Theorem 4.1].

When taking $d := |V(H)|$, the H -PACKING problem is a special case of d -SET PACKING. The latter problem asks, given a family of size- d subsets of a universe U and integer k , whether the family contains k pairwise disjoint sets. Dell and Marx [16] showed that for $d \geq 3$, there does not exist a kernel for d -SET PACKING with bit-size $\mathcal{O}(k^{d-\varepsilon})$ for any $\varepsilon > 0$, under the assumption that $\text{NP} \not\subseteq \text{coNP/poly}$. It is a long-standing open problem whether d -SET PACKING (or the related d -HITTING SET) admits a kernel with $\mathcal{O}(k)$ universe elements. Even for special cases such as TRIANGLE PACKING, no kernels with $\mathcal{O}(k)$ vertices are known on general graphs despite intensive research into linear-vertex kernels for packing problems [9, 24].

In this work, our focus is on the *weighted* variant of H -PACKING, which is defined as follows for a fixed graph H .

WEIGHTED H -PACKING **Parameter:** k

Input: An undirected graph G , a vertex-weight function $w: V(G) \rightarrow \mathbb{N}$, and positive integers k, t .

Question: Do there exist k vertex-disjoint subgraphs H_1, \dots, H_k of G such that H_i is isomorphic to H for each $i \in [k]$ and $\sum_{i \in [k]} \sum_{v \in V(H_i)} w(v) \geq t$?

The use of weights in the problem definition allows the problem to model a larger set of applications, since the weights can be used to capture different profits associated to a solution. Extending tractability horizons to weighted versions of problems is a natural and often challenging direction of research [14, 23, 36, 38, 40, 41].

It is not difficult to extend the sunflower-based kernel with $\mathcal{O}(k^{|V(H)|})$ vertices and edges to work in the weighted setting as well. However, the techniques used in previous papers to obtain (almost) linear kernels for sparse graph classes seem incompatible with the use of weights. For example, the meta-kernelization framework [11] does not apply to weighted problems since they do not have the *finite integer index* property.

Our results

In this work, we focus on the important special case that H is a clique on $h \geq 3$ vertices, which captures the TRIANGLE PACKING problem. We prove that WEIGHTED K_h -PACKING has a linear-vertex kernel on graph classes of bounded expansion. Classes of bounded expansion generalize planar graphs, bounded-degree graphs, and graphs excluding any fixed graph H as a minor or topological minor. Roughly speaking, a graph class \mathcal{G} has bounded expansion if there exists a function $f: \mathbb{N} \rightarrow \mathbb{N}$ such that for each $G \in \mathcal{G}$, the edge density of each graph G' that can be obtained from a subgraph of G by contracting disjoint connected vertex sets of diameter at most r , is bounded by $f(r)$. See [51, §5.5] for formal definitions. Our main result reads as follows.

► **Theorem 1.** *For each graph class \mathcal{G} of bounded expansion, for each integer $h \geq 3$, WEIGHTED K_h -PACKING admits a linear-vertex kernel on graphs from \mathcal{G} .*

Our approach extends to provide almost-linear kernels for every nowhere-dense graph class.

► **Theorem 2.** *For each nowhere-dense graph class \mathcal{G} , integer $h \geq 3$, and $\varepsilon > 0$, WEIGHTED K_h -PACKING admits a kernel with $\mathcal{O}(k^{1+\varepsilon})$ vertices on graphs from \mathcal{G} .*

The main idea behind our kernel for bounded-expansion graph classes is as follows. We start with a greedy phase that repeatedly extracts a maximum-weight K_h -subgraph. After having collected $h \cdot k$ such subgraphs, we show that if a solution exists, there is a solution in which each copy of K_h intersects one of the greedily identified subgraphs. This yields an $\mathcal{O}(k)$ -sized vertex set P_0 for which we can assume any solution intersects P_0 . Then we apply tools of sparsity to enrich P_0 into a slightly larger vertex set P of size $\mathcal{O}(k)$, such that the remaining vertices in G can be partitioned into $\mathcal{O}(k)$ equivalence classes in such a way that all K_h -subgraphs intersecting a class interact with the same set of $\mathcal{O}(1)$ vertices in P . Having this constant bound allows us to apply the sunflower lemma separately on each family of K_h -subgraphs intersecting a given equivalence class, in such a way that having a sunflower of *constant* size suffices to guarantee that one of the corresponding K_h -subgraphs can be avoided when making a solution. In this way, we can shrink each of the $\mathcal{O}(k)$ equivalence classes to $\mathcal{O}(1)$ vertices, giving a linear-vertex kernel. Hence our approach exploits the structural properties of sparse graphs to allow more efficient usage of the sunflower lemma. It easily generalizes to the setting of nowhere dense graph classes by utilizing a different lemma to compute the enriched set $P \supseteq P_0$, giving a bound of $\mathcal{O}(k^\varepsilon)$ rather than $\mathcal{O}(1)$ on the number of vertices from P that can interact with the copies of K_h in a given equivalence class.

We consider the conceptual simplicity of our kernelization algorithm an appealing feature. Unlike the meta-kernelization framework [11], it does not rely on treewidth-based argumentation and the corresponding notion of protrusion replacement. (The latter yields proofs that a kernelization algorithm exists, without explicitly showing what the algorithm is.) Compared to previous (kernelization) results on sparse graphs [4, 18, 54] we only require a few tools from the sparsity theory based on neighborhood complexity and the closure lemma, and avoid the use of the technical notion of *uniform quasi-wideness*.

In our argumentation, we focus on reducing the number of vertices in the instance to $\mathcal{O}(k)$. Strictly speaking this does not ensure the total encoding size becomes bounded in k , as the weights can be arbitrarily large. However, once the number of vertices is small, the weight-compression technique of Etscheid et al. [23] can be used to get to bound the maximum weight.

Related work

The study of kernelization on restricted graph classes began with the seminal result of Alber et al. [5], who proved that DOMINATING SET on planar graphs admits a linear kernel. It was later extended to larger graph classes [7, 28, 29, 30, 34]. The tools from sparsity have been extensively studied in the last decades. Dvorák, Král, and Thomas gave an FPT algorithm for deciding first-order properties in classes of graphs with bounded expansion [19], which was later extended to nowhere dense graph classes by Grohe, Kreutzer, and Siebertz [33]. It was also shown in [19] that if a graph class \mathcal{G} is not nowhere dense (is somewhere dense) and is closed under taking subgraphs, then model checking First Order formulae on \mathcal{G} is not FPT parameterized by the length of the formula unless $\text{FPT} = W[1]$.

In terms of kernelization, the first systematic study using the modern sparsity framework was started by Drange et al. [18]. They showed that for every fixed positive integer r , the r -DOMINATING SET problem admits a linear kernel on bounded expansion graphs. They also gave an almost-linear kernel for the standard DOMINATING SET problem on nowhere dense

graphs. Later, Eickmeyer et al. [20] showed that r -DOMINATING SET admits an almost-linear kernel on nowhere dense graphs. Pilipczuk and Siebertz [54] proved that the r -INDEPENDENT SET problem admits an almost-linear kernel on every nowhere dense graph class. The above kernelization results were recently unified by Einarson and Reidl [21] and Ahn et al. [4]. Apart from that, the tools from sparsity have also been used by Demaine et al. [17] in real-world graphs.

2 Preliminaries

We use standard notation for graphs and parameterized algorithms. We refer the reader to a textbook [15] for any undefined terms. For positive integers n we define $[n] := \{1, \dots, n\}$. We consider simple undirected graphs. A graph G has vertex set $V(G)$ and edge set $E(G)$. The open neighborhood of $v \in V(G)$ is $N_G(v) := \{u \mid \{u, v\} \in E(G)\}$, where we omit the subscript G if it is clear from context. For a vertex set $S \subseteq V(G)$ the open neighborhood of S , denoted $N_G(S)$, is defined as $N_G(S) := \bigcup_{v \in S} N_G(v) \setminus S$. For $S \subseteq V(G)$, the graph induced by S is denoted by $G[S]$. For two vertices x, y in a graph G , an $x - y$ path is a sequence $(x = v_1, \dots, v_k = y)$ of vertices such that $\{v_i, v_{i+1}\} \in E(G)$ for all $i \in [k - 1]$. Furthermore, the vertices v_2, \dots, v_{k-1} are called the *internal vertices* of the $x - y$ path. We say that a subgraph H of G *intersects* a vertex set $S \subseteq V(G)$ if $V(H) \cap S \neq \emptyset$.

We next state the following lemma due to Erdős-Rado [22]. Before presenting the lemma we define the terminology used in the lemma.

A *sunflower* \mathcal{S} with k sets and *core* X is a collection of sets S_1, \dots, S_k such that $S_i \cap S_j = X$ for all $i \neq j$, and such that $S_i \setminus X \neq \emptyset$ for all $i \in [k]$. The sets $S_i \setminus X$ are *petals* of the sunflower \mathcal{S} .

► **Theorem 3** (Sunflower lemma, [15, Theorem 2.25]). *Let \mathcal{A} be a family of sets (without duplicates) over a universe U , such that each set in \mathcal{A} has cardinality exactly d . If $|\mathcal{A}| > d!(k - 1)^d$, then \mathcal{A} contains a sunflower with k petals and such a sunflower can be computed in time polynomial in $|\mathcal{A}|, |U|$, and k .*

For completeness, we now give the formal definition of a kernelization. A *parameterized problem* Q is a subset of $\Sigma^* \times \mathbb{N}_+$, where Σ is a finite alphabet.

► **Definition 4** (Kernel). *Let $Q, Q' \subseteq \Sigma^* \times \mathbb{N}_+$ be parameterized problems and let $h: \mathbb{N}_+ \rightarrow \mathbb{N}_+$ be a computable function. A generalized kernel for Q into Q' of size $h(k)$ is an algorithm that, on input $(x, k) \in \Sigma^* \times \mathbb{N}_+$, takes time polynomial in $|x| + k$ and outputs an instance (x', k') such that:*

1. $|x'|$ and k' are bounded by $h(k)$, and
2. $(x', k') \in Q'$ if and only if $(x, k) \in Q$.

The algorithm is a kernel for Q if $Q = Q'$. It is a polynomial (generalized) kernel if $h(k)$ is a polynomial.

Sparsity. The theory of sparsity was introduced by Nešetřil and Ossona de Mendez [49, 50] using the notions of bounded expansion and nowhere denseness. Many important sparse graphs, like classes of bounded treewidth, planar graphs, graphs with bounded genus, apex-minor-free graphs, (topological)-minor free graphs, and graphs of bounded degree have bounded expansion. We refer the reader to the book [51] for a detailed introduction to the topic.

We will need some basic notation and tools for sparse graphs from earlier work [18, 20, 54] to prove our results. Let G be a graph and $X \subseteq V(G)$ be a subset of vertices. For a vertex $v \in V(G) \setminus X$ and a positive integer r , we define the r -*projection* of v onto X as

the set of all the vertices $w \in X$, for which there is a $v - w$ path in G of length at most r whose internal vertices do not belong to X . The r -projection of v onto a set X is denoted by $M_r(v, X)$.

Now we are ready to state the lemmas. The following lemma says that any vertex set $X \subseteq V(G)$ of a bounded expansion graph G can be “closed” to a set \widehat{X} whose size is asymptotically the same as $|X|$, such that the r -projection of any vertex outside \widehat{X} onto \widehat{X} has constant size.

► **Lemma 5** (Closure lemma, [18] Lemma 2.2). *Let \mathcal{G} be a class of bounded expansion. There exists a polynomial-time algorithm that, given a graph $G \in \mathcal{G}$, a non-negative integer r , and a set $X \subseteq V(G)$, computes a vertex-set \widehat{X} with the following properties.*

1. $X \subseteq \widehat{X} \subseteq V(G)$,
2. $|\widehat{X}| = \mathcal{O}(|X|)$, and
3. $|M_r(v, \widehat{X})| \leq \alpha \in \mathcal{O}(1)$, for each $v \in V(G) \setminus \widehat{X}$.

We note that in the above, the $\mathcal{O}(\cdot)$ notation also hides the factors depending on r and the graph class \mathcal{G} .

We also cite the corresponding closure lemma for nowhere dense graphs due to Eickmeyer et al. [20].

► **Lemma 6** (Closure lemma for nowhere dense graphs, [20, 54]). *Let \mathcal{G} be a nowhere dense class of graphs. There are a function $f_{cl}: \mathbb{N} \times \mathbb{R} \rightarrow \mathbb{N}$ and a polynomial-time algorithm that, given a graph $G \in \mathcal{G}$, a non-negative integer r , a set $X \subseteq V(G)$, and $\varepsilon > 0$, computes a vertex-set \widehat{X} with the following properties.*

1. $X \subseteq \widehat{X} \subseteq V(G)$,
2. $|\widehat{X}| = f_{cl}(r, \varepsilon) \cdot |X|^{1+\varepsilon}$, and
3. $|M_r(v, \widehat{X})| \leq f_{cl}(r, \varepsilon) \cdot |X|^\varepsilon$, for each $v \in V(G) \setminus \widehat{X}$.

In [18], Drange et al. proved that the number of distinct r -projections on a vertex set $X \subseteq V(G)$ of a bounded expansion graph G is linear in the cardinality of X .

► **Lemma 7** ([18, Lemma 2.3]). *Let \mathcal{G} be a class of bounded expansion and let r be a non-negative integer. Let $G \in \mathcal{G}$ be a graph and $X \subseteq V(G)$. Then*

$$|\{Y : Y = M_r(v, X) \text{ for some } v \in V(G) \setminus X\}| \leq c \cdot |X|,$$

for some constant c depending only on r and the graph class \mathcal{G} .

A similar bound exists for nowhere dense graphs.

► **Lemma 8** ([20, Theorem 3]). *Let \mathcal{G} be a nowhere dense class of graphs. There is a function $f_{nbr}: \mathbb{N} \times \mathbb{R} \rightarrow \mathbb{N}$ such that for every non-negative integer r , real $\varepsilon > 0$, graph $G \in \mathcal{G}$, and vertex set $X \subseteq V(G)$, we have*

$$|\{Y : Y = M_r(v, X) \text{ for some } v \in V(G) \setminus X\}| \leq f_{nbr}(r, \varepsilon) \cdot |X|^{1+\varepsilon},$$

for some constant c depending only on r and the graph class \mathcal{G} .

3 Kernelization for Weighted K_h -Packing on Sparse Graphs

In this section we present our kernels for WEIGHTED K_h -PACKING. We start by introducing some problem-specific terminology that will be useful to streamline our arguments.

A *solution* to an instance (G, w, k, t) of WEIGHTED K_h -PACKING is a sequence of vertex-disjoint subgraphs H_1, \dots, H_k of G such that H_i is isomorphic to K_h for each $i \in [k]$ and $\sum_{i \in [k]} \sum_{v \in V(H_i)} w(v) \geq t$.

► **Definition 9** (*P*-bound solution and solution confined to \mathcal{H}). Let (G, w, k, t) be an instance of WEIGHTED K_h -PACKING. For a vertex set $P \subseteq V(G)$, a solution H_1, \dots, H_k of (G, w, k, t) is *P*-bound if $V(H_i) \cap P \neq \emptyset$ for all $i \in [k]$.

For a collection \mathcal{H} of subgraphs isomorphic to K_h in G , a solution H_1, \dots, H_k of (G, w, k, t) is said to be confined to \mathcal{H} if $H_i \in \mathcal{H}$ for all $i \in [k]$.

We now show how the sunflower lemma can be combined with the theory of sparsity to get a linear-vertex kernel for WEIGHTED K_h -PACKING on bounded expansion graph classes.

► **Theorem 1.** For each graph class \mathcal{G} of bounded expansion, for each integer $h \geq 3$, WEIGHTED K_h -PACKING admits a linear-vertex kernel on graphs from \mathcal{G} .

Proof. Let (G, w, k, t) be an instance of WEIGHTED K_h -PACKING with $G \in \mathcal{G}$. We refer to a subgraph H_i of G isomorphic to K_h as a *copy* of K_h . In the following proof, we will treat such H_i both as a subgraph of G and as a vertex subset of G , depending on which is more convenient. Our kernelization algorithm performs the following steps.

Algorithm.

1. Compute a greedy packing \mathcal{P} of up to hk vertex-disjoint copies H_1, \dots, H_{hk} of K_h in G such that H_1 is a maximum-weighted copy of K_h in G , and for each $i \in \{2, \dots, hk\}$, the copy H_i is a maximum-weighted copy of K_h in the graph $G - (\bigcup_{j=1}^{i-1} V(H_j))$. While following the above greedy procedure, if it is not possible to pack hk disjoint copies of K_h then we obtain a maximal packing.
 2. Let $P_0 := V(\mathcal{P})$. Invoke the algorithm of Lemma 5 with G , $r = 2$, and $P_0 \subseteq V(G)$ to obtain a vertex set P such that:
 - a. $P_0 \subseteq P \subseteq V(G)$,
 - b. $|P| = \mathcal{O}(|P_0|) = \mathcal{O}(k)$, and
 - c. $|M_2(v, P)| \leq \alpha \in \mathcal{O}(1)$ for each $v \in V(G) \setminus P$,
 where α is a constant depending on r and the graph class \mathcal{G} .
 3. Partition the vertices of $V(G) \setminus P$ into equivalence classes C_1, \dots, C_m based on their 2-projection onto the set P , i.e., for every equivalence class C_i , and for every pair of distinct vertices $x, y \in C_i$, we have $M_2(x, P) = M_2(y, P)$. (Due to Lemma 7, we have $m = \mathcal{O}(|P|) = \mathcal{O}(k)$.)
 4. Let \mathcal{H} be the set family containing the (vertex sets of) all copies of K_h in G .
 - For each equivalence class C_i of Step 3, do the following.
 - Let $\mathcal{H}_i \subseteq \mathcal{H}$ be the (vertex sets of) copies of K_h in G that contain a vertex of C_i .
 - while** $|\mathcal{H}_i| > h!(h\alpha + 1)^h$ **do**
 - Apply Theorem 3 to obtain a sunflower $\mathcal{S} \subseteq \mathcal{H}_i$ with $h\alpha + 2$ copies of K_h .
 - Let $S_r \in \mathcal{S}$ be a copy of K_h with minimum weight. Remove S_r from \mathcal{H} and \mathcal{H}_i .
 - end while**
 5. Let $\mathcal{H}' \subseteq \mathcal{H}$ be the (reduced) set obtained after Step 4. Define $G' := G[V(\mathcal{H}')]$ and output (G', w, k, t) as the result of the kernelization.
- (Note that in principle, the encoding size of the weight function can be unbounded in terms of k , which can be resolved by a standard application of the weight reduction technique by Frank and Tardos [31] as explained by Etscheid et al. [23].)

This concludes the description of the algorithm.

Analysis. It is easy to observe that since we treat h as a constant, the above algorithm takes polynomial time: each step of the algorithm takes polynomial time and each step is applied a polynomial number of times. We now prove the correctness of the algorithm. Towards this, we first prove the following claim which says that any solution of (G, w, k, t) which is not already P -bound can be converted to a solution where the number of copies of K_h intersecting with P is strictly larger.

▷ **Claim 10.** If \mathcal{H}^* is a solution of (G, w, k, t) and $S_f \in \mathcal{H}^*$ such that $V(S_f) \cap P = \emptyset$, then there exists a set $S_j \in \mathcal{P}$ such that $(\mathcal{H}^* \setminus \{S_f\}) \cup \{S_j\}$ is a solution of (G, w, k, t) .

Proof. First note that, by the construction of the packing \mathcal{P} (in Step 1 of the algorithm), if $|\mathcal{P}| < hk$ then the constructed set \mathcal{P} is an inclusion-maximal packing, implying that all copies of H in G intersect $V(\mathcal{P})$ and therefore P . Under the stated assumptions, as S_f is a copy of K_h which is completely contained in the graph $G - P$, we have $|\mathcal{P}| = hk$. Moreover, for every $S_p \in \mathcal{P}$, it holds that $w(S_p) \geq w(S_f)$ since the copy S_f was available to choose in the iteration when the algorithm selected S_f , while the algorithm selects a maximum-weight copy at every step.

Since \mathcal{H}^* is a packing of k copies of K_h , $S_f \in \mathcal{H}^*$, and $V(S_f) \cap P = \emptyset$, at most $h(k-1)$ vertices of \mathcal{H}^* can intersect with the hk copies of K_h from the packing \mathcal{P} . Hence, there is at least one copy of K_h say $S_j \in \mathcal{P}$ such that $V(S_j) \cap V(\mathcal{H}^*) = \emptyset$. Moreover, we have $w(S_j) \geq w(S_f)$ as $S_j \in \mathcal{P}$. Thus the set $(\mathcal{H}^* \setminus \{S_f\}) \cup \{S_j\}$ is a solution of (G, w, k, t) . ◁

Next, using the above claim we prove that if there is a solution to (G, w, k, t) then there is a P -bound solution.

▷ **Claim 11.** If (G, w, k, t) has a solution, then (G, w, k, t) has a P -bound solution.

Proof. Let \mathcal{H}^* be a solution of (G, w, k, t) . If $V(S_f) \cap P \neq \emptyset$ for all $S_f \in \mathcal{H}^*$ then by Definition 9, the set \mathcal{H}^* is a P -bound solution. Otherwise, while there exists a set $S_f \in \mathcal{H}^*$ such that $V(S_f) \cap P = \emptyset$, we use Claim 10 to obtain a set $S_j \in \mathcal{P}$ such that $(\mathcal{H}^* \setminus \{S_f\}) \cup \{S_j\}$ is a solution of (G, w, k, t) with strictly fewer copies of K_h (than in \mathcal{H}^*) which are disjoint from the set P . ◁

Finally, in the following claim we prove that if (G, w, k, t) has a P -bound solution (which is guaranteed due to the above claim) then removal of the set S_r in Step 4 of the algorithm is safe.

▷ **Claim 12.** Suppose Step 4 of the above algorithm removes the set S_r from \mathcal{H} . If (G, w, k, t) has a P -bound solution which is confined to \mathcal{H} , then (G, w, k, t) has a P -bound solution which is confined to $\mathcal{H} \setminus \{S_r\}$.

Proof. Let $\mathcal{H}^* \subseteq \mathcal{H}$ be a P -bound solution of (G, w, k, t) which is confined to \mathcal{H} . If $S_r \notin \mathcal{H}^*$, then \mathcal{H}^* is also a P -bound solution which is confined to $\mathcal{H} \setminus \{S_r\}$. Therefore assume that $S_r \in \mathcal{H}^*$. Let $\mathcal{S} := \{S_1, \dots, S_{h\alpha+2}\}$ be the sunflower found in Step 4 of the above algorithm when it removes $S_r \in \mathcal{S}$ from \mathcal{H} , let X be its core, and let C_i be the equivalence class it considered when it found the sunflower. We now show that there exists a “free” set $S_f \in \mathcal{S} \setminus \{S_r\}$ such that the set $(\mathcal{H}^* \setminus \{S_r\}) \cup \{S_f\}$ is a solution of (G, w, k, t) which is confined to $\mathcal{H} \setminus \{S_r\}$. Towards this, we first derive the following: some set S_f of the sunflower $\mathcal{S} \setminus \{S_r\}$ is disjoint from $V(\mathcal{H}^* \setminus \{S_r\})$.

$$\exists S_f \in \mathcal{S} \setminus \{S_r\}: V(S_f) \cap V(\mathcal{H}^* \setminus \{S_r\}) = \emptyset. \quad (1)$$

Assume for a contradiction that there does not exist such a set S_f . First, note that since $S_r \in \mathcal{H}^*$, the core X of the sunflower \mathcal{S} is contained in S_r , and \mathcal{H}^* is a collection of vertex-disjoint copies of K_h , we have $X \cap V(\mathcal{H}^* \setminus \{S_r\}) = \emptyset$. Hence the copies of K_h in the set $\mathcal{H}^* \setminus \{S_r\}$ only intersect with petals of the sunflower $\mathcal{S} \setminus \{S_r\}$. Moreover, as the petals of a sunflower are pairwise disjoint, each copy of K_h from the set $\mathcal{H}^* \setminus \{S_r\}$ can intersect with at most h petals of the sunflower $\mathcal{S} \setminus \{S_r\}$.

Since all the $h\alpha + 1$ petals of the sunflower $\mathcal{S} \setminus \{S_r\}$ intersect with the set $V(\mathcal{H}^* \setminus \{S_r\})$ and a single copy of K_h from the set $\mathcal{H}^* \setminus \{S_r\}$ can hit at most h petals, the number of copies of K_h from $\mathcal{H}^* \setminus \{S_r\}$ intersecting with the petals of sunflower $\mathcal{S} \setminus \{S_r\}$ is at least $\alpha + 1$. Let $\mathcal{H}_S^* := \{H_{i_1}, \dots, H_{i_\ell}\} \subseteq \mathcal{H}^*$ be the set containing copies of K_h from the set $\mathcal{H}^* \setminus \{S_r\}$ which intersect with a petal of sunflower $\mathcal{S} \setminus \{S_r\}$. We have $\ell \geq \alpha + 1$.

We will prove that for each $q \in [\ell]$, there is a path P_q in G of length at most 2 that starts in a vertex of equivalence class C_i , ends in a vertex of $P \cap V(H_{i_q})$, and does not intersect any other vertex of P . Towards this end, let p_q be an arbitrary vertex of $V(H_{i_q}) \cap P$, which exists since the solution \mathcal{H}^* is P -bound. By choice of \mathcal{H}_S^* , the set $V(H_{i_q})$ intersects some petal $S_z \setminus X$ for $z \neq r$ of the sunflower $\mathcal{S} \setminus \{S_r\}$; let $x_q \in V(H_{i_q}) \cap (S_z \setminus X)$ be a vertex at which the sets intersect, and note that $\{x_q, p_q\} \in E(G)$ since they are both contained in the common clique H_{i_q} . Each set S_z contains a vertex from equivalence class C_i by the specification of Step 4, so there is a vertex $c_z \in S_z \cap C_i$. We have $\{c_z, x_q\} \in E(G)$ since these vertices are contained in the common clique S_z . Observe that $c_z \notin P$ since the equivalence classes partition the vertex set $V(G) \setminus P$. Now, if $x_q \notin P$ then the path (c_z, x_q, p_z) is the desired path P_q ; if $x_q \in P$ then we take (c_z, x_q) as the path P_q . Note that in both cases, P_q is indeed a path in G on at most 2 edges starting in C_i and ending in a vertex of $P \cap V(H_{i_q})$.

Hence for each $H_{i_q} \in \mathcal{H}_S^*$, there exists a vertex in the equivalence class C_i that can reach a vertex of $P \cap V(H_{i_q})$ by a path of length at most 2 whose internal vertices do not belong to P . By definition of the equivalence classes C_i , if one vertex in C_i has such a path to P , then *all* vertices of C_i have such a path. As $\ell \geq \alpha + 1$ and the copies in \mathcal{H}_S^* are disjoint, for any $v \in C_i$ we have $|M_2(v, P)| \geq \ell \geq \alpha + 1$. This contradicts that $|M_2(v, P)| \leq \alpha$ which was ensured by Step 2 of the above algorithm. Hence we establish (1).

Now we continue with the remaining proof of Claim 12. As there is a set $S_f \in \mathcal{S} \setminus \{S_r\}$ of the sunflower $\mathcal{S} \setminus \{S_r\}$ with $V(S_f) \cap V(\mathcal{H}^* \setminus \{S_r\}) = \emptyset$ and $w(S_f) \geq w(S_r)$ by our choice of S_r in Step 4 of the above algorithm, the set $\tilde{\mathcal{H}} := (\mathcal{H}^* \setminus \{S_r\}) \cup \{S_f\}$ is a solution of (G, w, k, t) . Note that if $V(S_f) \cap P \neq \emptyset$ then the set $\tilde{\mathcal{H}}$ is also a P -bound solution. Otherwise we invoke Claim 10 (with the set $\tilde{\mathcal{H}}$, and $S_f \in \tilde{\mathcal{H}}$) to obtain a K_h -copy $S_j \in \mathcal{P}$ of the packing \mathcal{P} such that the set $(\tilde{\mathcal{H}} \setminus \{S_f\}) \cup \{S_j\}$ is a P -bound solution of (G, w, k, t) . Note that the latter solution is also confined to $\mathcal{H} \setminus \{S_r\}$, since the copy $S_j \in \mathcal{P}$ was added to the set \mathcal{H} at the initialization and can never be removed: it does not occur in any \mathcal{H}_i since its vertex set is fully contained in P ; it does not intersect any equivalence class of $V(G) \setminus P$. This concludes the proof of Claim 12. \triangleleft

It follows from the preceding two claims that the output instance (G', t, k, w) is equivalent to the input (G, w, k, t) . Since the output is an induced subgraph of the input, one direction is trivial. For the other direction, if the input instance has a solution, it has a P -bound solution by Claim 11. Then by Claim 12 and induction, there is a solution confined to \mathcal{H}' , the final state of the variable \mathcal{H} . Since G' contains all copies of K_h contained in \mathcal{H}' , this proves the output instance also has a solution.

We conclude the proof of Theorem 1 by giving a bound on the number vertices of the reduced graph G' .

▷ Claim 13. $|V(G')| = \mathcal{O}(k)$.

Proof. Note that $|P| = \mathcal{O}(k)$ by Step 2. It follows that G' contains at most $\mathcal{O}(k)$ vertices which belong to P . To prove the claim, we show that the number of vertices of $V(G') \setminus P$ is also bounded by $\mathcal{O}(k)$.

For each equivalence class C_i of $V(G) \setminus P$, let \mathcal{H}'_i denote the contents of \mathcal{H}_i upon termination of the algorithm. The while-loop of Step 4 ensures that $|\mathcal{H}'_i| \leq h!(h\alpha + 1)^h$.

For each $v \in V(G') \setminus P$, by definition of $G' = G[V(\mathcal{H}')]$ there exists an equivalence class C_i of $V(G) \setminus P$ and a copy $H_j \in \mathcal{H}'_i$, such that $v \in V(H_j)$. Hence $V(G') \setminus P$ is contained in $\bigcup_i \bigcup_{H_j \in \mathcal{H}'_i} V(H_j)$. Since there are $\mathcal{O}(k)$ choices for i by Lemma 7, while $|\mathcal{H}'_i| \leq h!(h\alpha + 1)^h = \mathcal{O}(1)$, while each copy H_j also consists of $\mathcal{O}(1)$ vertices, it follows that $|V(G') \setminus P| = \mathcal{O}(k)$. This concludes the proof. ◀

This concludes the proof of Theorem 1. ◀

The argument for nowhere-dense graphs is almost identical.

► **Theorem 2.** *For each nowhere-dense graph class \mathcal{G} , integer $h \geq 3$, and $\varepsilon > 0$, WEIGHTED K_h -PACKING admits a kernel with $\mathcal{O}(k^{1+\varepsilon})$ vertices on graphs from \mathcal{G} .*

Proof. For WEIGHTED K_h -PACKING on a nowhere dense graph class \mathcal{C} , one can use the same approach. Let $\varepsilon' := \frac{\varepsilon}{h+1}$. In the algorithm, we use Lemma 6 for value ε' instead of Lemma 5; this means that the closure set P has size $\mathcal{O}(k^{1+\varepsilon'})$ rather than $\mathcal{O}(k)$, and that the bound α on $|M_2(v, P)|$ becomes $\mathcal{O}(k^{\varepsilon'})$ rather than $\mathcal{O}(1)$. For the analysis, we use Lemma 8 for value ε' instead of Lemma 7, which means the number of equivalence classes of $V(G) \setminus P$ becomes $\mathcal{O}(k^{1+\varepsilon'})$ rather than $\mathcal{O}(k)$. The rest of the algorithm and its correctness proof is identical.

As in the proof of Claim 13, we can bound the number of vertices in the resulting graph G' using the insight that every vertex of G' is either contained in P , or belongs to some copy H_j of K_h that remains in a set \mathcal{H}'_i for some equivalence class C_i of $V(G) \setminus P$. The key insight is again that $|\mathcal{H}'_i| \leq h!(h\alpha + 1)^h$ due to the application of the Sunflower lemma.

Hence the number of vertices in the reduced instance G' is bounded as follows:

$$\begin{aligned} |V(G')| &\leq |P| + \left| \bigcup_i \bigcup_{H_j \in \mathcal{H}'_i} V(H_j) \right| \\ &\leq \mathcal{O}(k^{1+\varepsilon'}) + \mathcal{O}(k^{1+\varepsilon'} \cdot h!(h\alpha + 1)^h \cdot h) \\ &\leq \mathcal{O}(k^{1+\varepsilon'}) + \mathcal{O}(k^{1+\varepsilon'} \cdot h!2^h h^h k^{\varepsilon' \cdot h} \cdot h) && \text{since } h\alpha + 1 \leq 2h\alpha \\ &\leq \mathcal{O}(k^{1+\varepsilon'+\varepsilon' \cdot h}) = \mathcal{O}(k^{1+\varepsilon'(h+1)}). && \text{since } h \in \mathcal{O}(1) \end{aligned}$$

Since we chose $\varepsilon' = \frac{\varepsilon}{h+1}$, the number of vertices in the kernel is indeed bounded by $\mathcal{O}(k^{1+\varepsilon})$, as required. ◀

4 Conclusions

We have shown that for a fixed complete graph K_h , the WEIGHTED K_h -PACKING problem admits a linear-vertex kernel on bounded-expansion graphs and an almost-linear kernel on nowhere-dense graphs. Whether there is a linear-vertex kernel for the associated WEIGHTED K_h -HITTING problem is an interesting problem for further study. In this problem, the input consists of a graph G , weight function $w: V(G) \rightarrow \mathbb{N}$, and integers k, t ; the question is whether there is a vertex set of size at most k and weight at most t that intersects all

K_h -subgraphs of G . In the unweighted setting, the kernelization complexity of packing problems typically matches that of the related hitting problem [8, 24]. In the weighted setting, the situation seems different and we do not know how to extend our techniques to WEIGHTED K_h -HITTING.

To illustrate the difficulty of hitting over packing in the presence of weights, observe the following. If $C_i \subseteq V(G)$ is a vertex subset such that all copies of K_h which intersect C_i also intersect a vertex set P_i of size $\mathcal{O}(1)$, then it effectively means that any packing of disjoint copies of K_h uses $\mathcal{O}(1)$ vertices of C_i , so that only a limited redundancy is needed in terms of which vertices of C_i are preserved in the kernel. But note that in the same scenario, a solution to WEIGHTED K_h -HITTING can contain up to k vertices from C_i : even though the K_h -subgraphs through C_i can be intersected by the vertex set P_i of size $\mathcal{O}(1)$, the weight of these vertices may be much larger than the weight of k vertices from C_i hitting the same subgraphs. Hence solutions to the hitting problem may select more than a constant number of vertices from C_i , which leads to having to store more vertices in the kernel.

References

- 1 Faisal N. Abu-Khzam. An improved kernelization algorithm for r -set packing. *Inf. Process. Lett.*, 110(16):621–624, 2010. doi:10.1016/j.ipl.2010.04.020.
- 2 Faisal N. Abu-Khzam. A kernelization algorithm for d -hitting set. *J. Comput. Syst. Sci.*, 76(7):524–531, 2010. doi:10.1016/j.jcss.2009.09.002.
- 3 Akanksha Agrawal, Daniel Lokshantov, Diptapriyo Majumdar, Amer E. Mouawad, and Saket Saurabh. Kernelization of cycle packing with relaxed disjointness constraints. *SIAM J. Discret. Math.*, 32(3):1619–1643, 2018. doi:10.1137/17M1136614.
- 4 Jungho Ahn, Jinha Kim, and O-joung Kwon. Unified almost linear kernels for generalized covering and packing problems on nowhere dense classes. *CoRR*, abs/2207.06660, 2022. doi:10.48550/arXiv.2207.06660.
- 5 Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, 2004. doi:10.1145/990308.990309.
- 6 Noga Alon, Gregory Z. Gutin, Eun Jung Kim, Stefan Szeider, and Anders Yeo. Solving max- r -sat above a tight lower bound. *Algorithmica*, 61(3):638–655, 2011. doi:10.1007/s00453-010-9428-7.
- 7 Noga Alon and Shai Gutner. Kernels for the dominating set problem on graphs with an excluded minor. *Electron. Colloquium Comput. Complex.*, TR08-066, 2008. arXiv:TR08-066.
- 8 Stéphane Bessy, Marin Bougeret, Dimitrios M. Thilikos, and Sebastian Wiederrecht. Kernelization for graph packing problems via rainbow matching. *CoRR*, abs/2207.06874, 2022. doi:10.48550/arXiv.2207.06874.
- 9 Stéphane Bessy, Marin Bougeret, Dimitrios M. Thilikos, and Sebastian Wiederrecht. Kernelization for graph packing problems via rainbow matching. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3654–3663. SIAM, 2023. doi:10.1137/1.9781611977554.ch139.
- 10 Daniel Binkele-Raible, Henning Fernau, Fedor V. Fomin, Daniel Lokshantov, Saket Saurabh, and Yngve Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. *ACM Trans. Algorithms*, 8(4):38:1–38:19, 2012. doi:10.1145/2344422.2344428.
- 11 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshantov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (meta) kernelization. *J. ACM*, 63(5):44:1–44:69, 2016. doi:10.1145/2973749.
- 12 Marthe Bonamy, Edouard Bonnet, Hugues Déprés, Louis Esperet, Colin Geniet, Claire Hilaire, Stéphane Thomassé, and Alexandra Wesolek. Sparse graphs with bounded induced cycle packing number have logarithmic treewidth. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3006–3028. SIAM, 2023. doi:10.1137/1.9781611977554.ch116.

- 13 Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001. doi:10.1006/jagm.2001.1186.
- 14 Miroslav Chlebík and Janka Chlebíková. Crown reductions for the minimum weighted vertex cover problem. *Discret. Appl. Math.*, 156(3):292–312, 2008. doi:10.1016/j.dam.2007.03.026.
- 15 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.
- 16 Holger Dell and Dániel Marx. Kernelization of packing problems. 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 68–81. SIAM, 2012. doi:10.1137/1.9781611973099.6.
- 17 Erik D. Demaine, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, Somnath Sikdar, and Blair D. Sullivan. Structural sparsity of complex networks: Bounded expansion in random models and real-world graphs. *J. Comput. Syst. Sci.*, 105:199–241, 2019. doi:10.1016/j.jcss.2019.05.004.
- 18 Pål Grønås Drange, Markus Sortland Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, Felix Reidl, Fernando Sánchez Villaamil, Saket Saurabh, Sebastian Siebertz, and Somnath Sikdar. Kernelization and sparseness: the case of dominating set. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPICs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.STACS.2016.31.
- 19 Zdenek Dvorák, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36:1–36:24, 2013. doi:10.1145/2499483.
- 20 Kord Eickmeyer, Archontia C. Giannopoulou, Stephan Kreutzer, O-joung Kwon, Michal Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. Neighborhood complexity and kernelization for nowhere dense classes of graphs. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 63:1–63:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.63.
- 21 Carl Einarson and Felix Reidl. A general kernelization technique for domination and independence problems in sparse classes. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPICs*, pages 11:1–11:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.IPEC.2020.11.
- 22 P. Erdős and R. Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, s1-35(1):85–90, 1960. doi:10.1112/jlms/s1-35.1.85.
- 23 Michael Etscheid, Stefan Kratsch, Matthias Mnich, and Heiko Röglin. Polynomial kernels for weighted problems. *J. Comput. Syst. Sci.*, 84:1–10, 2017. doi:10.1016/j.jcss.2016.06.004.
- 24 Fedor V. Fomin, Tien-Nam Le, Daniel Lokshtanov, Saket Saurabh, Stéphan Thomassé, and Meirav Zehavi. Subquadratic kernels for implicit 3-hitting set and 3-set packing problems. *ACM Trans. Algorithms*, 15(1):13:1–13:44, 2019. doi:10.1145/3293466.
- 25 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Quadratic upper bounds on the erdős-pósa property for a generalization of packing and covering cycles. *J. Graph Theory*, 74(4):417–424, 2013. doi:10.1002/jgt.21720.
- 26 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar f-deletion: Approximation, kernelization and optimal FPT algorithms. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 470–479. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.62.
- 27 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 503–510. SIAM, 2010. doi:10.1137/1.9781611973075.43.

- 28 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Linear kernels for (connected) dominating set on H -minor-free graphs. 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 82–93. SIAM, 2012. doi:10.1137/1.9781611973099.7.
- 29 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Linear kernels for (connected) dominating set on graphs with excluded topological subgraphs. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, volume 20 of *LIPICs*, pages 92–103. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPICs.STACS.2013.92.
- 30 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. *SIAM J. Comput.*, 49(6):1397–1422, 2020. doi:10.1137/16M1080264.
- 31 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Comb.*, 7(1):49–65, 1987. doi:10.1007/BF02579200.
- 32 Prachi Goyal, Neeldhara Misra, Fahad Panolan, and Meirav Zehavi. Deterministic algorithms for matching and packing problems based on representative sets. *SIAM J. Discret. Math.*, 29(4):1815–1836, 2015. doi:10.1137/140981290.
- 33 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.
- 34 Shai Gutner. Polynomial kernels and faster algorithms for the dominating set problem on graphs with an excluded minor. In Jianer Chen and Fedor V. Fomin, editors, *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, volume 5917 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2009. doi:10.1007/978-3-642-11269-0_20.
- 35 Penny E. Haxell, Alexandr V. Kostochka, and Stéphan Thomassé. Packing and covering triangles in K_4 -free planar graphs. *Graphs Comb.*, 28(5):653–662, 2012. doi:10.1007/s00373-011-1071-9.
- 36 Bart M. P. Jansen. Kernelization for maximum leaf spanning tree with positive vertex weights. *J. Graph Algorithms Appl.*, 16(4):811–846, 2012. doi:10.7155/jgaa.00279.
- 37 Bart M. P. Jansen. Turing kernelization for finding long paths and cycles in restricted graph classes. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 579–591. Springer, 2014. doi:10.1007/978-3-662-44777-2_48.
- 38 Bart M. P. Jansen, Shivesh Kumar Roy, and Michal Włodarczyk. On the hardness of compressing weights. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPICs*, pages 64:1–64:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.MFCS.2021.64.
- 39 Naonori Kakimura, Ken-ichi Kawarabayashi, and Dániel Marx. Packing cycles through prescribed vertices. *J. Comb. Theory, Ser. B*, 101(5):378–381, 2011. doi:10.1016/j.jctb.2011.03.004.
- 40 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Flow-augmentation III: complexity dichotomy for boolean csps parameterized by the number of unsatisfied constraints. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3218–3228. SIAM, 2023. doi:10.1137/1.9781611977554.ch122.
- 41 Eun Jung Kim, Marcin Pilipczuk, Roohani Sharma, and Magnus Wahlström. On weighted graph separation problems and flow-augmentation. *CoRR*, abs/2208.14841, 2022. doi:10.48550/arXiv.2208.14841.
- 42 David G. Kirkpatrick and Pavol Hell. On the complexity of general graph factor problems. *SIAM J. Comput.*, 12(3):601–609, 1983. doi:10.1137/0212040.

- 43 Stefan Kratsch. On polynomial kernels for integer linear programs: Covering, packing and feasibility. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, volume 8125 of *Lecture Notes in Computer Science*, pages 647–658. Springer, 2013. doi:10.1007/978-3-642-40450-4_55.
- 44 Stefan Kratsch and Vuong Anh Quyen. On kernels for covering and packing ilps with small coefficients. In Marek Cygan and Pinar Heggernes, editors, *Parameterized and Exact Computation - 9th International Symposium, IPEC 2014, Wroclaw, Poland, September 10-12, 2014. Revised Selected Papers*, volume 8894 of *Lecture Notes in Computer Science*, pages 307–318. Springer, 2014. doi:10.1007/978-3-319-13524-3_26.
- 45 Stefan Kratsch and Magnus Wahlström. Compression via matroids: A randomized polynomial kernel for odd cycle transversal. *ACM Trans. Algorithms*, 10(4):20:1–20:15, 2014. doi:10.1145/2635810.
- 46 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.
- 47 Daniel Lokshtanov, Amer E. Mouawad, Saket Saurabh, and Meirav Zehavi. Packing cycles faster than erdos-posa. *SIAM J. Discret. Math.*, 33(3):1194–1215, 2019. doi:10.1137/17M1150037.
- 48 Dániel Marx. Chordless cycle packing is fixed-parameter tractable. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPICs*, pages 71:1–71:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ESA.2020.71.
- 49 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion i. decompositions. *Eur. J. Comb.*, 29(3):760–776, 2008. doi:10.1016/j.ejc.2006.07.013.
- 50 Jaroslav Nešetřil and Patrice Ossona de Mendez. On nowhere dense graphs. *Eur. J. Comb.*, 32(4):600–617, 2011. doi:10.1016/j.ejc.2011.01.006.
- 51 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.
- 52 Christophe Paul, Anthony Perez, and Stéphan Thomassé. Conflict packing yields linear vertex-kernels for k -fast, k -dense RTI and a related problem. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 497–507. Springer, 2011. doi:10.1007/978-3-642-22993-0_45.
- 53 Marcin Pilipczuk, Michał Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen. Network sparsification for steiner problems on planar and bounded-genus graphs. *ACM Trans. Algorithms*, 14(4):53:1–53:73, 2018. doi:10.1145/3239560.
- 54 Michał Pilipczuk and Sebastian Siebertz. Kernelization and approximation of distance- r independent sets on nowhere dense graphs. *Eur. J. Comb.*, 94:103309, 2021. doi:10.1016/j.ejc.2021.103309.
- 55 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, 2010. doi:10.1145/1721837.1721848.
- 56 Meirav Zehavi. Parameterized approximation algorithms for packing problems. *Theor. Comput. Sci.*, 648:40–55, 2016. doi:10.1016/j.tcs.2016.08.004.

How Can We Maximize Phylogenetic Diversity? Parameterized Approaches for Networks

Mark Jones  

TU Delft, The Netherlands

Jannik Schestag¹  

TU Delft, The Netherlands

Friedrich-Schiller-Universität Jena, Germany

Abstract

Phylogenetic Diversity (PD) is a measure of the overall biodiversity of a set of present-day species (taxa) within a phylogenetic tree. We consider an extension of PD to phylogenetic networks. Given a phylogenetic network with weighted edges and a subset S of leaves, the all-paths phylogenetic diversity of S is the summed weight of all edges on a path from the root to some leaf in S . The problem of finding a bounded-size set S that maximizes this measure is polynomial-time solvable on trees, but NP-hard on networks. We study the latter from a parameterized perspective.

While this problem is $W[2]$ -hard with respect to the size of S (and $W[1]$ -hard with respect to the size of the complement of S), we show that it is FPT with respect to several other parameters, including the phylogenetic diversity of S , the acceptable loss of phylogenetic diversity, the number of reticulations in the network, and the treewidth of the underlying graph.

2012 ACM Subject Classification Theory of computation \rightarrow Fixed parameter tractability; Theory of computation \rightarrow W hierarchy; Applied computing \rightarrow Bioinformatics

Keywords and phrases Phylogenetic Networks, Phylogenetic Diversity, Parameterized Complexity, W-hierarchy, FPT algorithms

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.30

Funding *Mark Jones*: Partially supported by Netherlands Organisation for Scientific Research (NWO) grant OCENW.KLEIN.125.

Jannik Schestag: Supported by the German Academic Exchange Service (DAAD), project 57556279.

1 Introduction

Phylogenetic diversity, first introduced in 1992 by Faith [8] is a measure of the amount of biodiversity in a set of species. It formalizes the intuitive notion that a set of species is likely to have a greater range of biological features when they are distantly related. Such a measure is of crucial importance in the field of biological conservation, where there are often insufficient resources available to save every threatened species, one must make hard decisions about which species to prioritize. Phylogenetic diversity forms the basis of the Fair Proportion Index and the Shapley Value [11, 12, 17], which are used to evaluate the individual contribution of individual species to overall biodiversity. These measures are used by conservation initiatives such as the IUCN's Phylogenetic Diversity Task Force (<https://www.pdtf.org/>) and the Zoological Society of London's EDGE of Existence program [14].

Let \mathcal{T} be a phylogenetic tree; that is, a rooted tree, with weights on the edges, and S a subset of leaves of \mathcal{T} (representing a subset of present-day species). Then the *phylogenetic diversity* $PD_{\mathcal{T}}(S)$, as defined by Faith, is the sum of all weights on a path from the root

¹ The research was carried out during an extended research visit of Jannik Schestag at TU Delft.



© Mark Jones and Jannik Schestag;

licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 30; pp. 30:1–30:12

Leibniz International Proceedings in Informatics



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

to one of the leaves in S . Here the weight of an edge corresponds to phylogenetic distance, which is taken to be proportional to the number of features of interest (e.g. biological characteristics) that emerge along that edge.

Phylogenetic Diversity as originally proposed by Faith is defined for phylogenetic trees. Consequently, it does not allow for models of evolutionary history with reticulation events (where a species inherits genetic data from two or more species), such as hybridization or lateral gene transfer. Such events are modeled in phylogenetic *networks* (directed acyclic graphs with a single source), which extend the class of phylogenetic trees [13]. There are a number of ways to extend phylogenetic diversity to phylogenetic networks. In this paper we consider one of the simplest, *all-paths phylogenetic diversity* (first introduced under the name “phylogenetic subnet diversity” in [21] and further studied in [2]). Under this measure, given a rooted phylogenetic network \mathcal{N} with edge weights and a subset of leaves S , the phylogenetic diversity of S is again the total weight of all edges on a (directed) path from the root to one of the leaves in S .

Assuming it is not possible to preserve all threatened species (e.g. due to limited resources), we would like to find a subset of species that can be preserved, for which the overall diversity is maximized. This gives rise to the maximum phylogenetic diversity problem: given a network \mathcal{N} and integer k , find a set of leaves S with $|S| \leq k$ such with maximum phylogenetic diversity score. Fortunately in the case of trees, this turns out to be a tractable problem - given as input a phylogenetic tree and number k , there is a polynomial-time greedy algorithm that outputs the set of k species with maximum phylogenetic diversity [19, 16]. Unfortunately this result does not extend to phylogenetic networks - the problem is NP-hard, and cannot be approximated in polynomial time with approximation ratio better than $1 - \frac{1}{e}$ unless $P = NP$ [2]. For this reason, we study the problem from the perspective of parameterized complexity.

Related Work

All-paths phylogenetic diversity as a measure on networks was first introduced in [21]. The computational complexity of MAPPD was first studied in [2], where the authors showed that the problem is NP-hard and cannot be approximated in polynomial time with approximation ratio better than $1 - \frac{1}{e}$ unless $P = NP$, but is polynomial-time solvable on the class of level-1 networks (in which the undirected cycles are pairwise vertex-disjoint).

Phylogenetic diversity forms the basis of the Shapley Value, a measure that describes how much a *single* species contributes to overall biodiversity. The definition of the Shapley Value involves the phylogenetic diversity of every possible subset of species, and so is difficult to calculate directly. However it was shown in [9] that (on phylogenetic trees) the Shapley Value is equivalent to the Fair Proportion Index [17], which can be calculated in polynomial time. In the case of phylogenetic networks, it was shown that this result also extends to Shapley Value based on all-paths phylogenetic diversity. This is in contrast to the NP-hardness result of [2] - while it is easy to determine the individual species that contributes the most phylogenetic diversity across all sets of species, it is hard to find a *set* of species for which the phylogenetic diversity is maximal.

The phylogenetic networks considered in this paper are *explicit* networks, in which each vertex represents a different species in evolutionary history and the edges represent the transfer of genetic information from one species to another. Phylogenetic diversity has also been studied on *split* networks. Such networks do not represent a single explicit evolutionary history, but can represent structural information from several sources (e.g. conflicting phylogenetic trees). See e.g. [3, 18].

Our contribution

We study several parameterizations of the problem MAX-ALL-PATHS-PD (MAPPD), in which the task is to find a set of at most k leaves maximizing the all-paths phylogenetic diversity in a network (see Section 2 for a formal definition). We first consider the problem parameterized by k . We show in Section 3 that this problem is W[2]-hard by reduction from SET COVER. Moreover, we establish an equivalence between this parameterization of MAPPD and a generalization of SET COVER called ITEM-WEIGHTED PARTIAL SET COVER. We also show via a similar method that MAPPD is W[1]-hard with respect to the “dual” of k , namely $\bar{k} := |X| - k$, where X is the set of all leaves in the network. On the positive side, we show in Section 4.1 that MAPPD is fixed-parameter tractable (FPT) with respect to D , the total phylogenetic diversity of the desired solution, and also with respect to the “dual” \bar{D} , i.e. the acceptable loss in phylogenetic diversity. Finally we turn to structural parameters. In Section 4.2 we give single-exponential fixed-parameter algorithms for MAPPD with respect to the number of reticulations in the network, and with respect to the treewidth of the underlying graph of the network. In the case of reticulations, this algorithm is asymptotically tight under the Strong Exponential Time Hypothesis.

2 Preliminaries

Mathematical Definitions

For an integer ℓ , by $[\ell]$ we denote the set $\{1, \dots, \ell\}$ and $[\ell]_0 := \{0\} \cup [\ell]$.

A *phylogenetic X-network* $\mathcal{N} = (V, E, \omega)$ is a directed acyclic graph with *edge-weight* function $\omega : E \rightarrow \mathbb{N}_{>0}$ and a single vertex of indegree 0 (the *root*), in which the vertices of outdegree 0 (the *leaves*) have in-degree 1 and are bijectively labeled with elements from a set X , and such that all vertices either have indegree at most 1 or outdegree at most 1. The vertices with indegree at least 2 and outdegree 1 are called *reticulations*; the other non-leaf vertices are called *tree vertices*. In biological applications, the set X is a set of *taxa*, the internal vertices of \mathcal{N} correspond to biological ancestors of these taxa and $\omega(e)$ describes the phylogenetic distance between the endpoints of e (as these endpoints correspond to distinct species, we may assume this distance is greater than 0). For brevity, we will usually refer to a phylogenetic X -network as an *X-network*, or more simply a *network* when the set X is not relevant.

For a vertex v , the *descendants* $\text{desc}(v)$ (*ancestors* $\text{anc}(v)$) of v is the set of vertices u for which there is a path from v to u (from u to v). The *offspring* $\text{off}(v)$ of v is the intersection of $\text{desc}(v)$ and X . Further for an edge $e = (v, w)$ we define $\text{anc}(e) = \text{anc}(v)$, $\text{desc}(e) = \text{desc}(w)$ and $\text{off}(e) = \text{off}(w)$. For a set of taxa Y , an edge e is *affected by Y* if $\text{off}(e) \cap Y \neq \emptyset$ and *strictly affected by Y* if $\text{off}(e) \subseteq Y$. The sets T_Y and E_Y are the strictly affected and affected edges by Y , respectively. For a set of taxa Y , the *all-paths phylogenetic diversity* $PD_{\mathcal{N}}(Y)$ of Y is

$$PD_{\mathcal{N}}(Y) := \sum_{e \in E_Y} \omega(e).$$

That is, $PD_{\mathcal{N}}(Y)$ is the total weight of all edges (u, v) in \mathcal{N} so that there is a path from v to a vertex in Y . In what follows we refer to $PD_{\mathcal{N}}(Y)$ simply as the *phylogenetic diversity* of Y .

For a detailed introduction to parameterized complexity refer to the standard monographs [5, 7].

Problem Definitions and Parameterizations

Our main object of study is the following problem, introduced in [2]:

MAX-ALL-PATHS-PD (MAPPD)

Input: A phylogenetic X -network \mathcal{N} and two integers k and D .

Question: Is there a subset $Y \subseteq X$ of taxa with size at most k and phylogenetic diversity at least D ? That is $|Y| \leq k$ and $PD_{\mathcal{N}}(Y) \geq D$.

In Section 3 we show that there is a strong connection between MAPPD and the problem ITEM-WEIGHTED PARTIAL SET COVER, which is defined as follows.

ITEM-WEIGHTED PARTIAL SET COVER (WPSC)

Input: A universe \mathcal{U} , a family \mathcal{F} of subsets over \mathcal{U} , an integer weight $\omega(u)$ for each item $u \in \mathcal{U}$ and two integers k and D .

Question: Are there sets $F_1, \dots, F_k \in \mathcal{F}$ such that sum of the weights of the elements in $L := \bigcup_{i=1}^k F_i$ is at least D ? That is $\sum_{u \in L} \omega(u) \geq D$.

SET COVER is the special case of WPSC with $D = |\mathcal{U}|$ and $\omega(u) = 1$ for each $u \in \mathcal{U}$.

We examine MAPPD within the framework of parameterized complexity. In addition to the parameters k and D which are the number of saved taxa and the preserved phylogenetic diversity, we also study the dual parameters which are the minimum number of species that will go extinct $\bar{k} := |X| - k$ and the acceptable loss of phylogenetic diversity $\bar{D} := PD_{\mathcal{N}}(X) - D$. By $\text{ret}_{\mathcal{N}}$ we denote the number of reticulations in \mathcal{N} , and by $\text{tw}_{\mathcal{N}}$ we denote the treewidth of the underlying undirected graph of \mathcal{N} (see, e.g. [5, Chapter 7] for an overview of treewidth). By \max_{ω} we denote the biggest weight of an edge.

Binary Networks

A phylogenetic X -network is called *binary* if each non-leaf, non-root vertex has degree 3, and the root has degree 2. We note that in this paper (with the exception of Lemma 4.3 and Theorem 4.4) we do not assume networks are binary; in particular, we allow tree vertices to have indegree and outdegree 1. Bordewich et al. [2], we have required that the given network \mathcal{N} is binary. In the following, we show that algorithmically, there is hardly any difference.

The proofs of theorems and lemmas marked with (\star) are deferred to a longer version of this paper.

► **Lemma 2.1** (\star) . *For every instance (\mathcal{N}, k, D) of MAPPD an equivalent instance (\mathcal{N}', k', D') of MAPPD with a binary network \mathcal{N}' , $\text{tw}_{\mathcal{N}'} = \text{tw}_{\mathcal{N}}$ and $|E'| \leq 2|E|$ can be computed in $\mathcal{O}(|E|)$ time.*

3 Relationship to ITEM-WEIGHTED PARTIAL SET COVER

In this section, we demonstrate a relationship between MAPPD and WPSC by presenting reductions in both directions. Bordewich et al. already proved a similar reduction from SET COVER to MAPPD [2].

► **Theorem 3.1.** *For every instance $\mathcal{I} = (\mathcal{U}, \mathcal{F}, \omega, k, D)$ of WPSC,*

1. *an equivalent instance $\mathcal{I}' = (\mathcal{N}, k', D')$ of MAPPD with $k' = k$ and $|X| = \text{ret}_{\mathcal{N}} = |\mathcal{F}|$ can be computed in time polynomial in $|\mathcal{U}| + |\mathcal{F}|$;*
2. *an equivalent instance $\mathcal{I}'_2 = (\mathcal{N} = (V, E, \omega'), k', D')$ of MAPPD in which $k' = k$ and each edge weights 1 can be computed in time polynomial in $|\mathcal{U}| + |\mathcal{F}| + \max_{\omega}$.*

This theorem has several applications for the complexity of MAPPD. Because SET COVER is W[2]-hard with respect to the size of the solution k , MAPPD is as well. This is in contrast to the fact that MAPPD can be solved in polynomial time when the network does not have reticulations and therefore is a phylogenetic tree [19].

► **Corollary 3.2.** *MAPPD is W[2]-hard when parameterized with k , even if $\max_{\omega} = 1$.*

In RED-BLUE NON-BLOCKER an undirected bipartite graph G with vertex bipartition $V = V_r \cup V_b$ and an integer k are given. The question is whether there is a set $S \subseteq V_r$ of size at least k such that each vertex v of V_b has a neighbor in $V_r \setminus S$. There is a standard reduction from RED-BLUE NON-BLOCKER to SET COVER: Let V_b be the universe, for each vertex $v \in V_r$ add a set $F_v := N(v)$ to \mathcal{F} and finally set $k' := |V_r| - k$. RED-BLUE NON-BLOCKER is W[1]-hard when parameterized by the size of the solution [6]. Hence, SET COVER is W[1]-hard with respect to $|\mathcal{F}| - k$ and with Theorem 3.1 we conclude as follows.

► **Theorem 3.3.** *MAPPD is W[1]-hard when parameterized with $\bar{k} = |X| - k$.*

MAPPD can be solved in $\mathcal{O}^*(2^{|X|})$ with a brute force algorithm that tries every possible subset of species as a solution. In Theorem 4.5 we will prove that MAPPD can be solved in $\mathcal{O}^*(2^{\text{ret}_{\mathcal{N}}})$ time. In order to prove that these algorithms can not be improved significantly, we apply the well-established **Strong Exponential Time Hypothesis (SETH)**.

Unless SETH fails, SET COVER can not be solved in $\mathcal{O}^*(2^{\epsilon \cdot |F|})$ time for any $\epsilon < 1$ [4, 15]. Thus, Theorem 3.1 shows that under SETH, not a lot of hope remains to find faster algorithms for MAPPD than these two algorithms. Thus, these two algorithms, with respect to the number of taxa $|X|$ and reticulations $\text{ret}_{\mathcal{N}}$, for MAPPD are tight with the lower bounds.

► **Corollary 3.4.** *MAPPD can not be solved in $\mathcal{O}(2^{\epsilon \cdot |X|}) \cdot \text{poly}(|\mathcal{I}|)$ time or in $\mathcal{O}(2^{\epsilon \cdot \text{ret}_{\mathcal{N}}}) \cdot \text{poly}(|\mathcal{I}|)$ time for any $\epsilon < 1$, unless SETH fails.*

So now, without further ado, we prove Theorem 3.1.

Proof of Theorem 3.1.

Reduction. Let $\mathcal{I} = (\mathcal{U}, \mathcal{F}, k, D)$ be an instance of WPSC. Let \mathcal{U} consist of the items u_1, \dots, u_n and let \mathcal{F} contain the sets F_1, \dots, F_m . We may assume that for each u_i there is a set F_j which contains u_i . We define an instance $\mathcal{I}' = (\mathcal{N}, k, D')$ of MAPPD as follows. Let k stay unchanged and define $D' := D \cdot Q + 1$ for $Q := m(n + 1)$. We define a network \mathcal{N} with leaves x_1, \dots, x_m , and further vertices $r, v_1, \dots, v_n, w_1, \dots, w_m$.

Let the set of edges consist of the edges (r, v_i) for $i \in [n]$, (w_j, x_j) for $j \in [m]$, and let (v_i, w_j) be an edge if and only if $u_i \in F_j$. We define the weight of (r, v_i) to be $\omega(u_i) \cdot Q$ for each $i \in [n]$ and 1 for each other edge. Figure 1 depicts an example of this reduction.

This completes the construction of instance \mathcal{I}' in case 2 of the theorem. We now describe how to construct an instance \mathcal{I}'_2 from \mathcal{I}' in which the maximum weight of an edge is 1, completing the construction for case 1. For each edge $e = (r, v_i)$ with $w(e) > 1$, make $\omega(e) - 1$ subdivisions and attach a new leaf as the child of each subdividing vertex. We call these newly-added leaves *false leaves*, and we call the other leaves of \mathcal{N} *true leaves*.

Correctness. The proof of the correctness is deferred to a longer version of this paper. ◀

In the proof of Theorem 3.1, we can see that in the root r , we model an operation that ensures that at least D of the children of r are selected and further, these tree vertices ensure that at least one of the reticulations below them are selected. It might appear that by adding

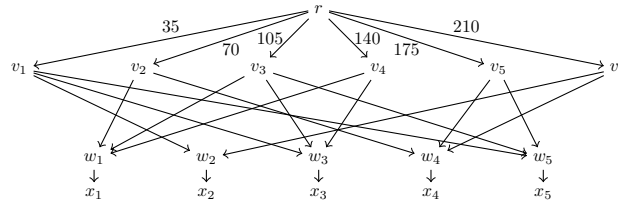


Figure 1 This figure depicts the network \mathcal{N} that we reduce to from the instance $(\mathcal{U} := \{u_1, \dots, u_6\}, \mathcal{F} := \{F_1, \dots, F_5\}, \omega, k, D)$ of WPSC with $\omega(u_i) = i$, $F_1 := \{u_2, u_3, u_4\}$, $F_2 := \{u_1, u_6\}$, $F_3 := \{u_1, u_3, u_4\}$, $F_4 := \{u_2, u_5, u_6\}$, $F_5 := \{u_1, u_3, u_5\}$. Unlabeled edges have a weight of 1. Here $n = 6, m = 5$ and $Q = 35$. The value of k' would be k and D' would be $35D + 1$.

more layers of reticulations and tree vertices to the construction of \mathcal{N} , one could reduce from problems even more complex than WPSC, and thereby show that MAPPD has an even higher position in the W-hierarchy. This however is unlikely, because of the reduction to WPSC that we are about to show.

► **Theorem 3.5.** *For every instance $\mathcal{I} = (\mathcal{N}, k, D)$ of MAPPD, we can compute an equivalent instance $(\mathcal{U}, \mathcal{F}, \omega, k', D')$ of WPSC with $k' = k, D' = D$ and $\max_{\omega'} = \max_{\omega}$ in time polynomial in $|\mathcal{I}|$.*

Proof.

Reduction. Let $\mathcal{I} = (\mathcal{N}, k, D)$ be an instance of MAPPD. We define an instance $\mathcal{I}' = (\mathcal{U}, \mathcal{F}, \omega', k, D)$ of WPSC as follows. Let k and D stay unchanged. For each edge e of \mathcal{N} , define an item u_e with weight $\omega'(u_e) = \omega(e)$ and let \mathcal{U} be the set of these u_e . For each taxon x , define a set F_x which contains item u_e if and only if e is affected by $\{x\}$. Let \mathcal{F} be the family of these F_x .

Correctness. Clearly, the reduction is computed in polynomial time. We show the equivalence of the two instances.

Let Y be a solution for the instance \mathcal{I} of MAPPD. Without loss of generality, assume $Y = \{x_1, \dots, x_\ell\}$ with $\ell \leq k$. We show that F_1, \dots, F_ℓ is a solution for \mathcal{I}' of WPSC. By definition, $\ell \leq k$. Let E_Y be the edges affected by Y . Observe that e is in E_Y if and only if u_e is in $F^+ := \bigcup_{i=1}^{\ell} F_i$. Then, $D \leq PD_{\mathcal{N}}(Y) = \sum_{e \in E_Y} \omega(e) = \sum_{u_e \in F^+} \omega'(u_e)$. Hence, F_1, \dots, F_ℓ is a solution for \mathcal{I}' of WPSC.

Now, without loss of generality, let F_1, \dots, F_ℓ be a solution for \mathcal{I}' of WPSC. Let u_{e_1}, \dots, u_{e_p} be the items in the union of F_1, \dots, F_ℓ . By the construction, the edges e_1, \dots, e_p are affected by $Y = \{x_1, \dots, x_\ell\}$. Then, $PD_{\mathcal{N}}(Y) \geq \sum_{i=1}^p \omega(e_i) = \sum_{i=1}^p \omega'(u_{e_i}) \geq D$. Because the size of Y is at most k , Y is a solution for \mathcal{I} of MAPPD. ◀

To the best of our knowledge, it is unknown if WPSC is W[2]-complete, like SET COVER. Nevertheless, we obtain the following connection between WPSC and MAPPD.

► **Corollary 3.6.** *MAPPD is W[t]-complete with respect to k if and only if WPSC is W[t]-complete with respect to k .*

4 Fixed-Parameter Tractability Results

4.1 Preserved and lost Diversity

In this subsection, we show that MAPPD is FPT with respect to the threshold of phylogenetic diversity D and the acceptable loss of phylogenetic diversity $\bar{D} := PD_{\mathcal{N}}(X) - D$.

Let \mathcal{I} be an instance of MAPPD. If there is an edge e with $\omega(e) \geq D$ and $k \geq 1$, then for each offspring x of e we have $PD_{\mathcal{N}}(\{x\}) \geq \omega(e) \geq D$, and so $\{x\}$ is a solution for \mathcal{I} . So, we may assume that $\max_{\omega} < D$. Therefore, each edge e can be subdivided $\omega(e) - 1$ times in $\mathcal{O}(D \cdot m)$ time such that $\omega'(e) = 1$ for each edge e of the new network \mathcal{N}' . Bläser showed that WPSC can be solved in $\mathcal{O}^*(2^{\mathcal{O}(D)})$ time when $\omega(u) = 1$ for each item $u \in \mathcal{U}$ [1]. Subsequently, with Theorem 3.5 and the result from Bläser we conclude the following.

► **Corollary 4.1.** *MAPPD can be solved in $\mathcal{O}^*(2^{\mathcal{O}(D)})$ time.*

As SET COVER is a special case of WPSC with $D = \sum_{u \in \mathcal{U}} \omega(u)$, WPSC is para-NP-hard with respect to the dual $\sum_{u \in \mathcal{U}} \omega(u) - D$. By contrast, we show in the following that MAPPD is FPT with respect to \bar{D} .

To this end, we use the technique of color coding. Recall that $\text{off}(e) = \text{off}(w)$ for each edge $e = (v, w)$ and the strictly affected edges T_Y for a set of taxa $Y \subseteq X$ is the set of edges e with $\text{off}(e) \subseteq Y$. We define an auxiliary problem.

COLORED-MAX-ALL-PATHS-PD (COLORED-MAPPD)

Input: A phylogenetic X -network \mathcal{N} , an edge-coloring $c : E \rightarrow \{\text{red}, \text{green}\}$ and integers k and D .

Question: Is there a subset $Y \subseteq X$ of taxa such that $|Y| \leq k$, $PD_{\mathcal{N}}(Y) \geq D$ and each edge in $T_{X \setminus Y}$ is colored red, while edges not in $T_{X \setminus Y}$ but adjacent to $T_{X \setminus Y}$ are colored green?

In order to solve COLORED-MAPPD we observe the following.

► **Lemma 4.2** (\star). $T_{Y_1 \cup \dots \cup Y_\ell} = T_{Y_1} \cup \dots \cup T_{Y_\ell}$ for any $Y_1, \dots, Y_\ell \subseteq X$ such that each vertex v of \mathcal{N} is incident with edges of at most one set of $T_{Y_1}, \dots, T_{Y_\ell}$.

► **Lemma 4.3.** *COLORED-MAPPD can be solved in $\mathcal{O}(\bar{D} \cdot m \cdot \log(\bar{k} + \max_{\omega}))$ time on binary networks.*

Proof.

Algorithm. Let $\mathcal{I} := (\mathcal{N} := (V, E, \omega), c, k, D)$ be an instance of COLORED-MAPPD. Compute the graph $G = (V, E')$, where E' is the subset of edges colored red.

For every weakly connected component $C = (V_C, E_C)$ of G proceed as follows. Compute the subset of leaves Y_C that are in V_C , and from this compute T_{Y_C} , the set of strictly affected edges in \mathcal{N} for Y_C . If $Y_C = \emptyset$ or $T_{Y_C} \neq E_C$ then continue with the next connected component. Otherwise, define an item I_C with *weight* $\omega(T_{Y_C})$ and *value* $|Y_C|$.

Let N be the set of these items. Now return yes if there is a subset of items in N whose total weight is at most \bar{D} and whose total value is at least $\bar{k} = |X| - k$, and no otherwise. Observe that this can be determined by solving an instance of KNAPSACK with set of items N , budget \bar{D} , and target value \bar{k} , which can be done in $\mathcal{O}(\bar{D} \cdot |N| \cdot \log(\bar{k})) = \mathcal{O}(\bar{D} \cdot |X| \cdot \log(\bar{k}))$ time [20, 10]. (The $\log(\bar{k})$ -factor of the running time comes from adding $\log(\bar{k})$ -digit numbers and is not mentioned in the original paper.)

Correctness. Assume that \mathcal{I} is a yes-instance of COLORED-MAPPD with solution $S \subseteq X$. Each edge e that is not affected by S is strictly affected by $X \setminus S$. Because S is a solution we conclude that the color of e is red and the connected component C_e of G that contains e contains a set of leaves Y_C of which $\text{off}(e)$ is a subset. Further, all edges of T_{Y_C} are colored red and the adjacent edges are colored green. Thus, C_e fulfills the conditions to be in N for each edge e that is not affected by S . Let C_1, \dots, C_t be the unique connected components

that contain the edges that is not affected by S . We conclude that $\omega(C_1 \cup \dots \cup C_t) \leq \bar{D}$ and $C_1 \cup \dots \cup C_t$ contain the leaves $X \setminus S$, which are at least \bar{k} . Hence, I_{C_1}, \dots, I_{C_t} is a solution for the KNAPSACK-instance and the algorithm returns yes.

Conversely, assume that the algorithm returns yes and let I_{C_1}, \dots, I_{C_t} be a solution for the KNAPSACK-instance. Let Y_i be the set of taxa such that $T_{Y_i} = E(C_i)$. We prove that $S := X \setminus \bigcup_{i=1}^t Y_i$ is a solution for the instance \mathcal{I} of COLORED-MAPPD. As the edges of each Y_i are colored red and the adjacent edges are green, we have that the edges of Y_i and Y_j are not adjacent for any $i \neq j$. Then by Lemma 4.2, $T_{X \setminus S} = T_{Y_1 \cup \dots \cup Y_t} = T_{Y_1} \cup \dots \cup T_{Y_t}$. We conclude that $T_{X \setminus S}$ is colored red and adjacent edges are green. Further, because $\sum_{i=1}^t \omega(T_{Y_i}) \leq \bar{D}$ the phylogenetic diversity of S is $PD_{\mathcal{N}}(S) = PD_{\mathcal{N}}(X) - \omega(T_{Y_1 \cup \dots \cup Y_t}) = PD_{\mathcal{N}}(X) - \sum_{i=1}^t \omega(T_{Y_i}) \geq PD_{\mathcal{N}}(X) - \bar{D} = D$. Likewise as $\sum_{i=1}^t |Y_i| \geq \bar{k}$, we conclude $|S| = |X| - \sum_{i=1}^t |Y_i| \leq |X| - \bar{k} = k$.

Running Time. The graph G and weakly connected components of G can be computed in $\mathcal{O}(m)$ time. For each component $C = (V_C, E_C)$ with leaves Y_C , the set T_Y can be computed in $\mathcal{O}(|T_{Y_C}|)$ time. It follows that we can determine whether $E_C = T_C$, and construct the set of items N , including their weights and values, in $\mathcal{O}(m \cdot \log(\max_{\omega}))$ time. As the instance of KNAPSACK can be solved in $\mathcal{O}(\bar{D} \cdot |X| \cdot \log(\bar{k}))$ time [10], we have an overall running time of $\mathcal{O}(m \cdot \log(\max_{\omega}) + \bar{D} \cdot |X| \cdot \log(\bar{k})) = \mathcal{O}(\bar{D} \cdot m \cdot \log(\bar{k} + \max_{\omega}))$. ◀

To show that MAPPD is FPT with respect to \bar{D} , we show that MAPPD can be reduced to COLORED-MAPPD using standard color coding techniques.

► **Theorem 4.4** (*). MAPPD can be solved in $\mathcal{O}(2^{3\bar{D} + \mathcal{O}(\log^2(\bar{D}))} \cdot m \log m \log(\bar{k} + \max_{\omega}))$ time on binary networks.

4.2 Proximity to a tree

MAPPD can be solved in polynomial time with Faith's Greedy-Algorithm, if the given network is a tree [19, 8]. Therefore, in this subsection, we examine MAPPD with respect to two parameters that classify the network's proximity to a tree, the number of reticulations $\text{ret}_{\mathcal{N}}$ and the smaller parameter treewidth $\text{tw}_{\mathcal{N}}$.

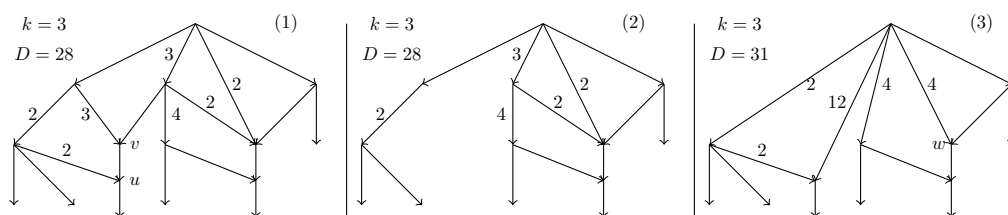
► **Theorem 4.5.** MAPPD can be solved in $\mathcal{O}(2^{\text{ret}_{\mathcal{N}}} \cdot k \cdot m \cdot \log(\max_{\omega}))$ time.

Observe that by Corollary 3.4, MAPPD can not be solved in $O^*(2^{\epsilon \cdot \text{ret}_{\mathcal{N}}})$ time for any $\epsilon < 1$, unless SETH fails. Therefore, the running time of the previous proof is tight, to some extent.

Proof.

Algorithm. For a reticulation v in a network \mathcal{N} with child u , let $E^{(\uparrow vu)}$ be the set of edges of \mathcal{N} that are between two vertices of $\text{anc}(v) \cup \{u\}$. Recall that $\text{off}(e) \subseteq X$ is the set of offspring of w for an edge $e = (v, w)$ and the strictly affected edges T_Y for a set of taxa $Y \subseteq X$ is the set of edges e with $\text{off}(e) \subseteq Y$. Define two operations, called **take** and **leave**, that for an instance $\mathcal{I} = (\mathcal{N}, k, D)$ and a reticulation v of \mathcal{N} return another instance of MAPPD. Every subset of taxa Y that does (does not, respectively) contain an offspring of v should be a solution for \mathcal{I} if and only if Y is a solution for **take**(\mathcal{I}, v) (**leave**(\mathcal{I}, v)).

We define **leave**(\mathcal{I}, v) to be the instance $\mathcal{I}' = (\mathcal{N}', k, D)$ of MAPPD, in which k and D are unchanged and \mathcal{N}' is the network that results from deleting the edges $T_{\text{off}(v)}$ and the resulting isolated vertices from \mathcal{N} . Recall that $\bar{D} = \sum_{e \in E} \omega(e) - D$. We define **take**(\mathcal{I}, v) to be the instance $\mathcal{I}' = (\mathcal{N}', k, D')$ of MAPPD with $D' = D + \bar{D}$ and k is unchanged. \mathcal{N}' is the network that results from \mathcal{N} by deleting the edges $E^{(\uparrow vu)}$, merging all the ancestors of v



■ **Figure 2** In this figure, an example for the usage of **leave** and **take** is given. A hypothetical instance \mathcal{I} is given in (1). Here, the value of \bar{D} is 3. In (2) the instance $\mathbf{leave}(\mathcal{I}, v)$, and in (3) the instance $\mathbf{take}(\mathcal{I}, v)$ is depicted. Unlabeled edges have a weight of 1. Observe in (3), the weight of the edge (r, w) is 4, as w has two edges from ancestors of v in \mathcal{I} which have a weight of 2 each. The weight of (r, u) is 12, as in \mathcal{I} the edges of $E^{\uparrow vu}$ have a combined weight of 9.

to a single vertex r , adding an edge (r, u) , and setting the weight $\omega'((r, u))$ to $\omega(E^{\uparrow vu}) + \bar{D}$. For each vertex $w \neq u$ that has $t \geq 1$ parents u_1, \dots, u_t , in $\text{anc}(v)$, we add an edge (r, w) that has weight $\sum_{i=1}^t \omega((u_i, w))$. Observe that $PD_{\mathcal{N}'}(X) = PD_{\mathcal{N}}(X) + \bar{D}$. Figure 2 depicts an example of the operations **take** and **leave**.

Now, we are at the position to define the branching algorithm. Let $\mathcal{I} = (\mathcal{N}, k, D)$ be an instance of MAPPD. If \mathcal{N} is a phylogenetic tree, solve the instance \mathcal{I} with Faith's Algorithm [19, 8]. Otherwise, let v be a reticulation of \mathcal{N} . Then, return yes if $\mathbf{take}(\mathcal{I}, v)$ or $\mathbf{leave}(\mathcal{I}, v)$ is a yes-instance of MAPPD and no otherwise.

Correctness. The correctness of the base case is given by the correctness of Faith's Algorithm. We show that if \mathcal{N} contains a reticulation v , then \mathcal{I} is a yes-instance of MAPPD if and only if $\mathbf{take}(\mathcal{I}, v)$ or $\mathbf{leave}(\mathcal{I}, v)$ is a yes-instance of MAPPD.

Consider any set of taxa $Y \subseteq X$. Firstly, we claim that if $Y \cap \text{off}(e) = \emptyset$, then $PD_{\mathcal{N}'}(Y) = PD_{\mathcal{N}}(Y)$, where \mathcal{N}' is the network in $\mathbf{leave}(\mathcal{I}, v)$. Indeed, \mathcal{N}' contains all the vertices and edges of \mathcal{N} that have an offspring outside of $\text{off}(v)$. Therefore, $PD_{\mathcal{N}'}(Y) = PD_{\mathcal{N}}(Y)$. Secondly, we claim that if $Y \cap \text{off}(v) \neq \emptyset$, $PD_{\mathcal{N}'}(Y) = PD_{\mathcal{N}}(Y) + \bar{D}$, where \mathcal{N}' is the network for $\mathbf{take}(\mathcal{I}, v)$. Recall that each edge $e = (u_1, u_2)$ with $u_1 \neq r$ of $E(\mathcal{N}')$ is also an edge of \mathcal{N} and $\omega'(e) = \omega(e)$. Further, for each edge $e = (r, u_2)$ with $u_2 \neq u$ of $E(\mathcal{N}')$ there are edges $e_1 = (u_{i_1}, u_2), \dots, e_t = (u_{i_t}, u_2)$ of $E(\mathcal{N})$ with $\omega'(e) = \sum_{i=1}^t \omega(e_i)$. Now, let $Q = Q_1 \cup Q_2 \cup \{(r, u)\}$ be the edges of \mathcal{N}' that have at least one offspring in Y , of which edges in Q_1 have both endpoints in $V(\mathcal{N}') \setminus \{r\}$, and Q_2 are outgoing edges of r . Further, let $P = P_1 \cup P_2 \cup E^{\uparrow vu}$ be the edges of \mathcal{N} that have at least one offspring in Y , of which edges in P_1 have both endpoints in $V(\mathcal{N}')$, and P_2 are edges with one endpoint in $\text{anc}(v) \setminus \{v\}$ and one endpoint in $V(\mathcal{N}') \setminus \{r\}$. Observe that, since any vertex in $V(\mathcal{N}')$ has the same offspring in \mathcal{N} as in \mathcal{N}' , $Q_1 = P_1$ and $\omega'(Q_1) = \omega(P_1)$. Further, $\omega'(Q_2) = \omega(P_2)$ as for each $u_2 \in V(\mathcal{N}') \setminus \{r\}$, the total weight of edges (u_1, u_2) with $u_1 \in \text{anc}(v) \setminus \{v\}$ in \mathcal{N} is equal to the weight of the edge (r, u_2) in \mathcal{N}' . It follows that $PD_{\mathcal{N}'}(Y) = \omega'(Q_1) + \omega'(Q_2) + \omega'(\{r, u\}) = \omega(P_1) + \omega(P_2) + \omega(E^{\uparrow vu}) + \bar{D} = PD_{\mathcal{N}}(Y) + \bar{D}$.

It follows from the above that if Y is a solution for \mathcal{I} (that is, $|Y| \leq k$ and $PD_{\mathcal{N}}(Y) \geq D$), then either Y is a solution for $\mathbf{leave}(\mathcal{I}, v)$ or Y is a solution for $\mathbf{take}(\mathcal{I}, v)$. Conversely, if Y is a solution for $\mathbf{leave}(\mathcal{I}, v)$ then $Y \cap \text{off}(e) = \emptyset$ and thus $PD_{\mathcal{N}}(Y) = PD_{\mathcal{N}'}(Y) \geq D$, so Y is also a solution for \mathcal{I} . Finally, if Y is a solution for $\mathbf{take}(\mathcal{I}, v)$ then $Y \cap \text{off}(e) \neq \emptyset$, as otherwise $PD_{\mathcal{N}'}(Y) \leq PD_{\mathcal{N}'}(X) - \omega'(\{r, y\}) = D + 2\bar{D} - (\omega(E^{\uparrow vu}) + \bar{D}) \leq D + \bar{D} - 1 < D'$. Then $PD_{\mathcal{N}'}(Y) = PD_{\mathcal{N}}(Y) + \bar{D}$, from which it follows that $PD_{\mathcal{N}}(Y) \geq D' - \bar{D} = D$ and Y is also a solution for \mathcal{I} .

30:10 How Can We Maximize Phylogenetic Diversity in Networks?

Running Time. Let \mathcal{I} be an instance of MAPPD that contains a reticulation v . The number of reticulations in \mathcal{I} is greater than the number of reticulations in $\mathbf{take}(\mathcal{I}, v)$ and $\mathbf{leave}(\mathcal{I}, v)$, because at least the reticulation v is removed and no new reticulations are added. Therefore, the search tree contains $\mathcal{O}(2^{\text{ret}_{\mathcal{N}}})$ nodes. It can be checked in $\mathcal{O}(m)$ time, if \mathcal{N} contains a reticulation. Faith's Algorithm takes $\mathcal{O}(k \cdot m \cdot \log(\max_{\omega}))$ [19].

The sets $\text{off}(v)$, $\text{anc}(v)$ for a vertex v , and T_Y for a set Y can be computed in $\mathcal{O}(m)$ time. Once $\text{anc}(v)$ is computed, we can iterate over E to find the edges that are outgoing from $\text{anc}(v)$ and compute the value for an edge (r, w) in \mathcal{N}' in $\mathcal{O}(m \cdot \log(\max_{\omega}))$ time, which is also the time needed to compute $\omega((r, u))$ which needs \overline{D} and the weight of $E^{(\uparrow vu)}$. Therefore, the instances $\mathbf{take}(\mathcal{I}, v)$ and $\mathbf{leave}(\mathcal{I}, v)$ can be computed in $\mathcal{O}(m \cdot \log(\max_{\omega}))$ time.

Thus, a solution for MAPPD can be computed in $\mathcal{O}(2^{\text{ret}_{\mathcal{N}}} \cdot k \cdot m \cdot \log(\max_{\omega}))$ time. ◀

Bordewich et al. showed that MAPPD can be solved in polynomial time on level-1 networks [2]. We extend this result by showing that MAPPD is fixed-parameter tractable with respect to treewidth.

► **Theorem 4.6** (\star). MAPPD can be solved in $\mathcal{O}(9^{\text{tw}_{\mathcal{N}}} \cdot \text{tw}_{\mathcal{N}} \cdot k^2 \cdot m)$ time.

The detailed proof is deferred to a longer version of this paper; we give a sketch of the main ideas here.

We aim to find a set of edges E' that have an overall weight of at least D and that are incident with at most k leaves. Further, for each edge $e = (u, v) \in E'$ we require that either $v \in X$ or there is an edge $(v, w) \in E'$. In the algorithm, which is a dynamic program over a nice tree decomposition, we index feasible partial solutions by a 3-coloring of the vertices. At a given node of the tree decomposition, a vertex v is colored:

- red, if it is still mandatory that we select an outgoing edge of v (because we have selected an incoming edge of v),
- green, if we can select incoming edges of v and do not need to select an outgoing edge of v (because v is a leaf or we have already selected an outgoing edge of v),
- black, if we have to not yet selected an edge incident with v (such that only the selection of an incoming edge of v makes the selection of an outgoing edge of v necessary).

We introduce each leaf as a green vertex and the other vertices as black vertices. In order to consider only feasible solutions, a vertex must be green or black when it is forgotten. The most important step of the algorithm is in the introduction of an edge, where colors may be adjusted depending on whether or not the new edge is included in E' .

5 Discussion

While we were able to show that MAPPD is $\mathbb{W}[2]$ -hard parameterized by k , it is unknown whether it is $\mathbb{W}[2]$ -complete. We were however able to show an equivalence between MAPPD parameterized by k and ITEM-WEIGHTED PARTIAL SET COVER parameterized by the size of the solution. Thus establishing the exact complexity class of ITEM-WEIGHTED PARTIAL SET COVER, which seems to be of interest, would also establish the exact complexity class of MAPPD.

The all-paths phylogenetic diversity measure $PD_{\mathcal{N}}$ considered in this paper is one of four measures considered in [2], where it is called *AllPaths-PD*. The second measure, which they call *Network-PD*, requires not only weights on each of the edges in the network, but also an *inheritance proportion* $p(e)$ on each edge $e = (u, v)$ leading into a reticulation. This value denotes the expected number of features that are expected to be passed from u to v . Network-PD is a generalization of AllPaths-PD, as the measures are equivalent when all inheritance

proportions are 1. The authors also consider two additional measures, *MinWeightTree-PD* and *MaxWeightTree-PD*, that, under certain restrictions, act as lower and upper bounds respectively on Network-PD.

It is natural to ask whether our parameterized complexity results for MAPPD extend to the corresponding maximization problems for Network-PD. We note that, since Network-PD generalizes AllPaths-PD, our hardness results for k and \bar{k} also carry across to Network-PD. For the FPT results, the main challenge is that to compute Network-PD for a network \mathcal{N} and a subset of leave S , one must compute for each edge e an expected proportion $\gamma(S, e)$ of features arising in e that will be passed down to an offspring in S . $\gamma(S, e)$ is computed recursively; for an edge $e = (u, v)$ the value of $\gamma(S, e)$ is a non-linear function of the value $\gamma(S, e')$ for all edges e' leaving wv . Taking these values into account is likely to complicate the FPT algorithms presented in this paper significantly.

References

- 1 Markus Bläser. Computing small Partial Coverings. *Information Processing Letters*, 85(6):327–331, 2003.
- 2 Magnus Bordewich, Charles Semple, and Kristina Wicke. On the Complexity of optimising variants of Phylogenetic Diversity on Phylogenetic Networks. *Theoretical Computer Science*, 917:66–80, 2022.
- 3 Olga Chernomor, Steffen Klaere, Arndt von Haeseler, and Bui Quang Minh. *Split Diversity: Measuring and Optimizing Biodiversity using Phylogenetic Split Networks*, pages 173–195. Springer International Publishing, 2016.
- 4 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, and et al. On Problems as hard as CNF-SAT. *ACM Transactions on Algorithms (TALG)*, 12(3):1–24, 2016.
- 5 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 6 Rodney G. Downey and Michael R. Fellows. Fixed-Parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141(1-2):109–131, 1995.
- 7 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- 8 Daniel P. Faith. Conservation evaluation and Phylogenetic Diversity. *Biological Conservation*, 61(1):1–10, 1992.
- 9 Michael Fuchs and Emma Yu Jin. Equality of Shapley value and fair proportion index in phylogenetic trees. *Journal of mathematical biology*, 71:1133–1147, 2015.
- 10 Frank Gurski, Carolin Rehs, and Jochen Rethmann. Knapsack Problems: A parameterized point of view. *Theoretical Computer Science*, 775:93–108, 2019.
- 11 Claus-Jochen Haake, Akemi Kashiwada, and Francis Edward Su. The Shapley value of Phylogenetic Trees. *Journal of mathematical biology*, 56(4):479–497, 2008.
- 12 Klaas Hartmann. The equivalence of two Phylogenetic Diodiversity measures: the Shapley value and Fair Proportion index. *Journal of Mathematical Biology*, 67:1163–1170, 2013.
- 13 Daniel H. Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic Networks: Concepts, Algorithms and Applications*. Cambridge University Press, 2010.
- 14 Nick J.B. Isaac, Samuel T. Turvey, Ben Collen, Carly Waterman, and Jonathan E.M. Baillie. Mammals on the EDGE: Conservation Priorities Based on Threat and Phylogeny. *PLOS ONE*, 2(3):1–7, 2007.
- 15 Bingkai Lin. A simple gap-producing Reduction for the parameterized Set Cover Problem. *arXiv preprint arXiv:1902.03702*, 2019.
- 16 Fabio Pardi and Nick Goldman. Species Choice for Comparative Genomics: Being Greedy Works. *PLoS Genetics*, 1, 2005.

30:12 How Can We Maximize Phylogenetic Diversity in Networks?

- 17 David W. Redding and Arne Ø. Mooers. Incorporating evolutionary measures into conservation prioritization. *Conservation Biology*, 20(6):1670–1678, 2006.
- 18 Andreas Spillner, Binh T. Nguyen, and Vincent Moulton. Computing Phylogenetic Diversity for Split Systems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(2):235–244, 2008.
- 19 Mike Steel. Phylogenetic Diversity and the greedy algorithm. *Systematic Biology*, 54(4):527–529, 2005.
- 20 H. Martin Weingartner. Capital budgeting of interrelated projects: survey and synthesis. *Management Science*, 12(7):485–516, 1966.
- 21 Kristina Wicke and Mareike Fischer. Phylogenetic Diversity and biodiversity indices on Phylogenetic Networks. *Mathematical Biosciences*, 298:80–90, 2018.

Sidestepping Barriers for Dominating Set in Parameterized Complexity

Ioannis Koutis  

New Jersey Institute of Technology, NJ, USA

Michał Włodarczyk  

University of Warsaw, Poland

Meirav Zehavi  

Ben-Gurion University of the Negev, Beerhseba, Israel

Abstract

We study the classic DOMINATING SET problem with respect to several prominent parameters. Specifically, we present algorithmic results that sidestep time complexity barriers by the incorporation of either approximation or larger parameterization. Our results span several parameterization regimes, including: (i,ii,iii) time/ratio-tradeoff for the parameters *treewidth*, *vertex modulator to constant treewidth* and *solution size*; (iv,v) FPT-algorithms for the parameters *vertex cover number* and *feedback edge set number*; and (vi) compression for the parameter *feedback edge set number*.

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

Keywords and phrases Dominating Set, Parameterized Complexity, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.31

Related Version *Full Version*: <https://arxiv.org/abs/2309.15645>

Funding Supported by BGU–NJIT Joint Seed Research Fund and ERC Starting Grant titled PARAPATH.

1 Introduction

The DOMINATING SET problem is one of the most central problems in Parameterized Complexity [8, 4]. The input to DOMINATING SET consists of an n -vertex graph G , and the objective is to output a minimum-size subset $U \subseteq V(G)$ that is a *dominating set* – that is, the closed neighborhood of U in G equals $V(G)$, or, in other words, every vertex in $V(G) \setminus U$ is adjacent in G to at least one vertex in U . When parameterized by the solution size and stated as a decision problem, the input also consists of a non-negative integer k , and the objective is to determine whether there exists a subset $U \subseteq V(G)$ of size at most k that is a dominating set.

From the perspective of Parameterized Complexity, DOMINATING SET parameterized by the sought solution size k is very hard. First, DOMINATING SET is W[2]-complete [8] (and, clearly, in XP).¹ In fact, DOMINATING SET and SET COVER are the two most well-studied W[2]-hard problems in Parameterized Complexity. Moreover, under the Strong Exponential Time Hypothesis (SETH), DOMINATING SET cannot be solved in $f(k) \cdot n^{k-\epsilon}$ time [19]. Still, for every integer $k \geq 7$, DOMINATING SET is solvable in $n^{k+o(1)}$ time [9].

¹ We refer to Section 2 for notations and concepts not defined in the introduction.



From the perspective of approximation (and parameterized approximation), DOMINATING SET is also very hard. Unless $P=NP$, DOMINATING SET does not admit a polynomial-time $(1 - \epsilon) \ln n$ -approximation algorithm for any fixed $\epsilon > 0$ [7] (see also [10]). However, it admits a polynomial-time $(\ln n - \ln \ln n + O(1))$ -approximation algorithm [23]. Moreover, under the SETH, DOMINATING SET does not even admit a $g(k)$ -approximation $f(k) \cdot n^{k-\epsilon}$ -time algorithm for any computable functions g and f of k and fixed $\epsilon > 0$ [20]. (Observe that this statement generalizes the one above by [19].) Similar (but weaker) results of hardness of approximation in the setting of Parameterized Complexity also exist under other assumptions, including the ETH, $W[1] \neq FPT$, and the k -SUM hypothesis [20].

Concerning structural parameters, the most well-studied parameters in Parameterized Complexity are treewidth and vertex cover number [8, 4]. Regarding DOMINATING SET, on the positive side, the problem is easily solvable in $O(3^{\text{tw}} \cdot n)$ time [21]. Here, tw is the treewidth of the input graph. However, under the SETH, DOMINATING SET cannot be solved in $(3 - \epsilon)^{\text{tw}} \cdot n^{O(1)}$ time for any fixed $\epsilon > 0$ [16]. Similarly to the case of the parameter solution size k , we again have (essentially) matching upper and lower bounds in terms of time complexity. Moreover, it is not hard to see that, under the any of the SETH and the Set Cover Conjecture, DOMINATING SET cannot be solved in $(2 - \epsilon)^{\text{vc}} \cdot n^{O(1)}$ time for any fixed $\epsilon > 0$ (see Section 5). Here, vc is the vertex cover number of the input graph.

Lastly, we note that the weighted version of DOMINATING SET is, similarly, approximable in polynomial time within factor $O(\log n)$, and solvable in $O(3^{\text{tw}} \cdot n)$ time. Further, being more general, all negative results carry to it as well.

1.1 Our Contribution

Our contribution is fivefold, concerning five different parameterizations.

I. Treewidth. First, in Section 3, we consider the treewidth tw of the given graph as the parameter. We prove that WEIGHTED DOMINATING SET admits a 2-approximation $O(\sqrt{6}^{\text{tw}} \cdot \text{tw}^{O(1)} \cdot n)$ -time algorithm. Our proof is based on “decoupling” the task of domination of the entire input graph G into two separate tasks: we compute a partition (V_1, V_2) of G based on a proposition of [17], and then consider the domination of each V_i separately. We remark that the way that we use the aforementioned proposition is very different than the way it is originally used in [17]. Here, we remind that under the SETH, DOMINATING SET cannot be solved in $(3 - \epsilon)^{\text{tw}} \cdot n^{O(1)}$ time for any fixed $\epsilon > 0$.

II. Modulator to Constant Treewidth. Second, in Section 4, we consider the parameter tw_d , the minimum-size of a vertex modulator of the given graph to treewidth d , for any fixed $d \in \mathbb{N}$. We note that, for any graph G , $\text{tw} \leq \text{tw}_d + d$. We prove that WEIGHTED DOMINATING SET admits a 2-approximation $O(2^{\text{tw}_d} \cdot n)$ -time algorithm. As before, our proof is based on “decoupling” the task of domination of the entire input graph G into two separate tasks: now, these are the task of the domination of the modulator, and the task of the domination of the rest of G . Unlike before, the resolution of these two tasks is different. Concerning the tightness of our result, we refer the reader to Conjecture 30, where we conjecture that the same time complexity cannot be attained by an exact algorithm.

III. Vertex Cover Number. Third, in Section 5, we consider the vertex cover number vc of the given graph as the parameter. We prove that WEIGHTED DOMINATING SET admits an $O(2^{\text{vc}} \cdot n)$ -time algorithm. Our proof is partially based on the idea of our algorithm for the parameter tw_d , combined with the observation that some vertices in the independent

set (being the complement of the given vertex cover) are “forced” to be picked, after having chosen which vertices to pick from the vertex cover. From the perspective of impossibility results, we observe that under any of the SETH and the Set Cover Conjecture, DOMINATING SET cannot be solved in $(2 - \epsilon)^{vc} \cdot n^{O(1)}$ time for any fixed $\epsilon > 0$, thus our time complexity is tight.

IV. Feedback Edge Set Number. Fourth, in Section 6, we consider the parameter fes (feedback edge set number), the minimum-size of a set of edges whose removal transforms the input graph into a forest. Notice that this parameter is a relaxation of tw_d (for any fixed $d \in \mathbb{N}$), which, in turn, is a relaxation of tw . We present two theorems. The first theorem is that DOMINATING SET admits an $O(3^{\frac{\text{fes}}{2}} \cdot n)$ -time algorithm. To this end, we prove the following combinatorial lemma, which is of independent interest: For any graph G , $\text{tw}_2(G) \leq \frac{\text{fes}(G)}{2}$. (Moreover, we prove that there exists an algorithm that, given a graph G , outputs a subset $M \subseteq V(G)$ such that $|M| \leq \frac{\text{fes}(G)}{2}$ and $\text{tw}(G - M) \leq 2$ in $O(\text{fes}(G) + n)$ time.) The second theorem is that an instance G of the DOMINATING SET problem with $\text{fes}(G) = k$ can be compressed in linear time into a “relaxed” instance of the problem on a graph \hat{G} with $O(k)$ edges, requiring the minimum domination of a subset of the vertices in \hat{G} .

V. Solution Size. Fifth, we consider the solution size k as the parameter. We prove that, for any fixed $0 \leq \alpha < 1$, DOMINATING SET admits a $((1 - \alpha) \ln n + O(1))$ -approximation $n^{\alpha k + O(1)}$ -time algorithm. The proof of this theorem is the simplest one in our article, based on the combination of an exhaustive search (to uncover part of the solution) and a known approximation algorithm. This approach somewhat resembles that of [5]. Here, we remind that (under plausible complexity-theoretic assumptions) it is unlikely for an exact algorithm to run in $n^{\alpha k + O(1)}$ time, or for a polynomial-time algorithm to have approximation factor $((1 - \alpha) \ln n + O(1))$.

Due to space constraints, the compression result in Contribution IV and Contribution V are deferred to the full version of this paper.

1.2 Other Related Works

Here, we briefly survey a few works not already mentioned that are directly relevant or related to ours. First, we note that DOMINATING SET can be solved in linear time on series-parallel graphs [14]. In particular, the class of series-parallel graphs is a (strict) subclass of the class of graphs of treewidth 2.

The restriction of DOMINATING SET to planar graphs is known to be both solvable in $2^{O(\sqrt{k})} n$ time (thus, it is in FPT) and admit an EPTAS [12]. Similar results exist also for more general graph classes, such as H -minor free graphs [11] and graphs of bounded expansion [1]. Moreover, DOMINATING SET is also in FPT (specifically, it is solvable in $2^{O(k)} n^{O(1)}$ time, and admits a polynomial kernel) on claw-free graphs, but it remains $W[2]$ -complete on $K_{1,t}$ -free graphs for any $t \geq 4$ [6, 13]. Additionally, DOMINATING SET is $W[1]$ -hard on unit disk graphs [18].

With respect to exact exponential-time algorithms, the currently best known running time upper bound is of $O(1.4969^n)$, based on the branch-and-reduce method [22].

2 Preliminaries

Given a function $f : U \rightarrow \mathbb{R}$ and a subset $U' \subseteq U$, let $f(U') = \sum_{u \in U'} f(u)$. When f is interpreted as a weight function, we will refer to $f(U')$ as the weight of U' .

Standard Graph Notation. Throughout the article, we deal with simple, finite, undirected graphs. Given a graph G , let $V(G)$ and $E(G)$ denote its vertex and edge sets, respectively. When no confusion arises, we denote $|V(G)| = n$ and $|E(G)| = m$. Given a vertex $v \in V(G)$, let $N_G(v) = \{u \in V(G) : \{u, v\} \in E(G)\}$ and $N_G[v] = N_G(v) \cup \{v\}$. Given a subset $U \subseteq V(G)$, let $N_G(U) = \bigcup_{u \in U} N_G(u) \setminus U$ and $N_G[U] = \bigcup_{u \in U} N_G[u]$. When G is clear from context, we drop it from the subscripts. Given a subset $U \subseteq V(G)$, let $G[U]$ denote the subgraph of G induced by U , and let $G - U$ denote the graph $G[V(G) \setminus U]$. Given a subset $F \subseteq E(G)$, let $G - F$ denote the graph on vertex set $V(G)$ and edge set $E(G) \setminus F$. Given subsets $A, B \subseteq V(G)$, we say that A *dominates* B if for every $b \in B$, $N[b] \cap A \neq \emptyset$. A *dominating set* of G is a subset $U \subseteq V(G)$ that dominates $V(G)$. A *vertex cover* of G is a subset $U \subseteq V(G)$ such that $G - U$ is edgeless (i.e., $E(G - U) = \emptyset$). Let $\text{vc}(G)$ denote the minimum size of a vertex cover of G . A *feedback edge set* of G is a subset $F \subseteq E(G)$ such that $G - F$ is a forest. Let $\text{fes}(G)$ denote the minimum size of a feedback edge set of G . We note that $\text{vc}(G)$ and $\text{fes}(G)$ are incomparable. When G is immaterial or clear from context, we denote $\text{vc} = \text{vc}(G)$ and $\text{fes} = \text{fes}(G)$. A *cactus* is a connected graph in which any two simple cycles have at most one vertex in common. We will slightly abuse this term: given graph G such that each connected component of G is a cactus, we will call G a cactus as well.

Problem Definitions. The DOMINATING SET problem is defined as follows: The input consists of an n -vertex graph G , and the objective is to output a minimum-size dominating set in G . The WEIGHTED DOMINATING SET is defined similarly: Here, the input also consists of a weight function $w : V(G) \rightarrow \mathbb{N}$, and the objective is to output a minimum-weight dominating set in G . When parameterized by the solution size k , we consider the decision version of DOMINATING SET, and suppose that the input also consists of a non-negative integer k . Then, the objective is to determine whether G has a dominating set of size at most k . Parameterization by the solution size k can also be defined for WEIGHTED DOMINATING SET. However, we find it to be somewhat less natural (particularly when considered from the perspective of parameterized approximation), and therefore we do not consider it in this paper. Still, we mention that, here, the objective would be to find a minimum-weight dominating set in G among all those of size at most k , if one exists.

The SET COVER problem is defined as follows: The input consists of a universe U and a family $\mathcal{F} \subseteq 2^U$ of subsets of U , and the objective is to output a minimum-size subfamily $\mathcal{S} \subseteq \mathcal{F}$ such that $U = \bigcup \mathcal{S}$. (Here, without loss of generality, we suppose that $U = \bigcup \mathcal{F}$, so there necessarily exist a solution.) The WEIGHTED SET COVER problem is defined similarly: Here, the input also consists of a weight function $w : \mathcal{F} \rightarrow \mathbb{N}$, and the objective is to output a minimum-weight subfamily $\mathcal{S} \subseteq \mathcal{F}$ such that $U = \bigcup \mathcal{S}$. We require a slight generalization of this problem, called GENERALIZED WEIGHTED SET COVER, defined as follows: The input (U, \mathcal{F}, w) is the same, and the objective is to output, for every subset $A \subseteq U$, a minimum-weight subfamily $\mathcal{S}_A \subseteq \mathcal{F}$ such that $U_A = \bigcup \mathcal{S}_A$.

Width Measures. The treewidth of a graph is a standard measure of its ‘‘closeness’’ to a tree, defined as follows.

► **Definition 1.** A tree decomposition of a graph G is a pair $\mathcal{T} = (T, \beta)$, where T is a rooted tree and β is a function from $V(T)$ to $2^{V(G)}$, that satisfies the following conditions.

- For every edge $\{u, v\} \in E(G)$, there exists $x \in V(T)$ such that $\{u, v\} \subseteq \beta(x)$.
- For every vertex $v \in V(G)$, $T[\{x \in V(T) : v \in \beta(x)\}]$ is a tree on at least one vertex.

The width of (T, β) is $\max_{x \in V(T)} |\beta(x)| - 1$. The treewidth of G , denoted by $\text{tw}(G)$, is the minimum width over all tree decompositions of G . For every $x \in V(T)$, $\beta(x)$ is called a bag, and $\gamma(x)$ denotes the union of the bags of x and the descendants of x in T .

When G is immaterial or clear from context, we denote $\text{tw} = \text{tw}(G)$. Following the standard custom in parameterized algorithmics, when we consider a problem parameterized by tw , we suppose that we are given a tree decomposition \mathcal{T} of width tw . Also, following the standard custom, we do not rely on a supposition that the width of \mathcal{T} is tw in the sense that, if the width of \mathcal{T} is larger, then our algorithmic result holds where tw is replaced by this width.

Given a graph G , a *path decomposition* of G is a tree decomposition (T, β) where T is a path, and the **PATHWIDTH** of G is the minimum width over all path decompositions of G .

For $d \in \mathbb{N} \cup \{0\}$ and a graph G , let $\text{tw}_d(G)$ denote the minimum size of a vertex set whose deletion from G results in a graph of treewidth at most d . Observe that, for any graph G , $\text{tw}_0(G) = \text{vc}(G)$. When G is immaterial or clear from context, we denote $\text{tw}_d = \text{tw}_d(G)$. Following the standard custom in parameterized algorithmics, when we consider a problem parameterized by tw_d , we suppose that we are given a vertex set M of size tw_d whose deletion from the input graph results in a graph of treewidth at most d . Also, following the standard custom, we do not rely on a supposition that $|M| = \text{tw}_d$ in the sense that, if $|M|$ is larger, then our algorithmic result holds where tw_d is replaced by $|M|$.

For the design of algorithms based dynamic programming, it is convenient to work with *nice* tree decompositions, defined as follows.

► **Definition 2.** A tree decomposition (T, β) of a graph G is *nice* if for the root $r = \text{root}(T)$ of T , $\beta(r) = \emptyset$, and each node $x \in V(T)$ is of one of the following types.

- **Leaf:** x is a leaf in T and $\beta(x) = \emptyset$.
- **Forget:** x has one child, y , and there is a vertex $v \in \beta(y)$ such that $\beta(x) = \beta(y) \setminus \{v\}$.
- **Introduce:** x has one child, y , and there is a vertex $v \in \beta(x)$ such that $\beta(x) \setminus \{v\} = \beta(y)$.
- **Join:** x has two children, y and z , and $\beta(x) = \beta(y) = \beta(z)$.

► **Proposition 3 ([2]).** Given a tree decomposition (T, β) of a graph G , a nice tree decomposition of G of the same width as (T, β) can be constructed in linear-time (specifically, $O(w^{O(1)} \cdot n)$ where w is the width of (T, β)).

Due to Proposition 3, when we deal with nice tree decompositions of width tw , we suppose that $|V(T)| \leq O(\text{tw}^{O(1)} \cdot n)$.

Parameterized Complexity. Let Π be an NP-hard problem. In the framework of Parameterized Complexity, each instance of Π is associated with a *parameter* k . Here, the goal is to confine the combinatorial explosion in the running time of an algorithm for Π to depend only on k . Formally, we say that Π is *fixed-parameter tractable (FPT)* if any instance (I, k) of Π is solvable in $f(k) \cdot |I|^{O(1)}$ time, where f is an arbitrary function of k . A weaker request is that for every fixed k , the problem Π would be solvable in polynomial time. Formally, we say that Π is *slice-wise polynomial (XP)* if any instance (I, k) of Π is solvable in $f(k) \cdot |I|^{g(k)}$ time, where f and g are arbitrary functions of k . Parameterized Complexity also provides methods to show that a problem is unlikely to be FPT. Here, the concept of W-hardness replaces the one of NP-hardness. For more information, we refer the reader to the book [4].

Essentially tight conditional lower bounds for the running times of parameterized algorithms often rely on the *Exponential-Time Hypothesis (ETH)*, the *Strong ETH (SETH)* and the *Set Cover Conjecture*. To formalize the statements of ETH and SETH, recall that given a formula φ in conjunctive normal form (CNF) with n variables and m clauses, the task of CNF-SAT is to decide whether there is a truth assignment to the variables that satisfies φ . In the p -CNF-SAT problem, each clause is restricted to have at most p literals. First, ETH asserts that 3-CNF-SAT cannot be solved in $O(2^{o(n)})$ time. Second, SETH asserts that for every fixed $\epsilon < 1$, there exists a (large) integer $p = p(\epsilon)$ such that p -CNF-SAT cannot be solved in $O((2 - \epsilon)^n)$ time. Moreover, the Set Cover Conjecture states that for every fixed $\epsilon < 1$ SET COVER cannot be solved in $O((2 - \epsilon)^n)$ time where n is the size of the universe.

Let P and Q be two parameterized problems. A *compression* (or *compression algorithm*) for P is a polynomial-time procedure that, given an instance (x, k) of P , outputs an equivalent instance (x', k') of Q where $|x'|, k' \leq f(k)$ for some computable function f . Then, we say that P admits a *compression of size $f(k)$* . When $P = Q$, compression is called *kernelization*.

When a problem is parameterized by the solution size k , the concept of parameterized approximation must be clarified (given that we then deal with a decision problem). Here, the objective becomes the following where the sought approximation ratio is some α : If there exists a solution of size at most k , then we seek an α -approximate solution; else, we can output any solution. Of course, we do not know (as part of the input) which case is true.

3 Parameter: Treewidth

We will use a translation of (WEIGHTED) DOMINATING SET into two instances of an easier problem, to attain a 2-approximation algorithm for (WEIGHTED) DOMINATING SET.

► **Theorem 4.** *The WEIGHTED DOMINATING SET problem parameterized by tw admits a 2-approximation $O(\sqrt{6}^{\text{tw}} \cdot \text{tw}^{O(1)} \cdot n)$ -time algorithm.*

Towards the proof of this theorem, we first define the easier problem that we aim to solve.

► **Definition 5 (Half-Width Domination).** *In the WEIGHTED HALF-WIDTH DOMINATION problem, the input consists of a graph G , a vertex-weight function $w : V(G) \rightarrow \mathbb{N}$, a nice tree decomposition $\mathcal{T} = (T, \beta)$ of G of width tw , and a subset $D \subseteq V(G)$ such that for every $x \in V(T)$, $|\beta(x) \cap D| \leq \frac{\text{tw}}{2} + O(1)$. The objective is to compute a subset $S \subseteq V(G)$ of minimum weight that dominates D .*

We would have liked to call the algorithm in the following proposition in order to directly solve the WEIGHTED HALF-WIDTH DOMINATION problem.

► **Proposition 6 ([4]).** *The WEIGHTED DOMINATING SET problem admits an $O(3^{\text{tw}} \cdot n)$ -time algorithm.*

Unfortunately, we cannot use it in a black-box manner – we need to modify the dynamic programming table used in the proof. Intuitively, we let vertices outside D correspond to two states (chosen, not chosen) instead of three (chosen, not chosen and dominated, not chosen and not dominated). This is done in the following lemma.

► **Lemma 7.** *The WEIGHTED HALF-WIDTH DOMINATION problem admits an $O(\sqrt{6}^{\text{tw}} \cdot \text{tw}^{O(1)} \cdot n)$ -time algorithm.*

Proof. We first describe the algorithm. Let $(G, w, D, \mathcal{T} = (T, \beta))$ be the give input. We use dynamic programming, and start with the formal definition of the table, denoted by \mathfrak{M} . For every $x \in V(T)$, partition (X, Y) of $\beta(x) \setminus D$ and partition $(\widehat{X}, \widehat{Y}_1, \widehat{Y}_2)$ of $\beta(x) \cap D$, we have a table entry $\mathfrak{M}[x, (X, Y), (\widehat{X}, \widehat{Y}_1, \widehat{Y}_2)]$. The order of the computation is done by postorder on T (where the order of computation of entries with the same first argument is arbitrary). Then, the basis corresponds to the case where x is a leaf. In this case, we initialize $\mathfrak{M}[x, (\emptyset, \emptyset), (\emptyset, \emptyset, \emptyset)] = 0$. Now, suppose that x is not a leaf. Then, we use the following recursive formulas:

1. In case x is of type Forget, let y be its child and $v \in \beta(y) \setminus \beta(x)$. Then, we compute $\mathfrak{M}[x, (X, Y), (\widehat{X}, \widehat{Y}_1, \widehat{Y}_2)]$:
 - If $v \in D$, then we take $\min\{\mathfrak{M}[y, (X, Y), (\widehat{X} \cup \{v\}, \widehat{Y}_1, \widehat{Y}_2)], \mathfrak{M}[y, (X, Y), (\widehat{X}, \widehat{Y}_1 \cup \{v\}, \widehat{Y}_2)]\}$.
 - Else, we take $\min\{\mathfrak{M}[y, (X \cup \{v\}, Y), (\widehat{X}, \widehat{Y}_1, \widehat{Y}_2)], \mathfrak{M}[y, (X, Y \cup \{v\}), (\widehat{X}, \widehat{Y}_1, \widehat{Y}_2)]\}$.
2. In case x is of type Introduce, let y be its child and $v \in \beta(x) \setminus \beta(y)$. Then, we compute $\mathfrak{M}[x, (X, Y), (\widehat{X}, \widehat{Y}_1, \widehat{Y}_2)]$:
 - If $v \in X \cup \widehat{X}$, then we take $\mathfrak{M}[y, (X \setminus \{v\}, Y), (\widehat{X} \setminus \{v\}, \widehat{Y}_1 \setminus N_G(v), \widehat{Y}_2 \cup (\widehat{Y}_1 \cap N_G(v)))] + w(v)$.
 - Else, if $v \in Y \cup \widehat{Y}_2$, then we take $\mathfrak{M}[y, (X, Y \setminus \{v\}), (\widehat{X}, \widehat{Y}_1, \widehat{Y}_2 \setminus \{v\})]$.
 - Else, if $v \in N_G(X \cup \widehat{X})$, then we take $\mathfrak{M}[y, (X, Y), (\widehat{X}, \widehat{Y}_1 \setminus \{v\}, \widehat{Y}_2)]$.
 - Else, we take ∞ .
3. In case x is of type Join, let y and z be its children. Then, $\mathfrak{M}[x, (X, Y), (\widehat{X}, \widehat{Y}_1, \widehat{Y}_2)]$ equals:

$$\min_{(Y_1^y, Y_1^z) \text{ partition of } \widehat{Y}_1} \{\mathfrak{M}[y, (X, Y), (\widehat{X}, Y_1^y, \widehat{Y}_2 \cup (\widehat{Y}_1 \setminus Y_1^y))] + \mathfrak{M}[z, (X, Y), (\widehat{X}, Y_1^z, \widehat{Y}_2 \cup (\widehat{Y}_1 \setminus Y_1^z))]\} - w(X \cup \widehat{X}).$$

Eventually, the algorithm returns the weight stored in $M[\text{root}(T), (\emptyset, \emptyset), (\emptyset, \emptyset, \emptyset)]$, where the matching itself can be retrieved by backtracking its computation (specifically, collecting the vertices inserted into $X \cup \widehat{X}$).

Because for every $x \in V(T)$, $|\beta(x) \cap D| \leq \frac{\text{tw}}{2} + O(1)$, and $|V(T)| \leq O(\text{tw}^{O(1)} \cdot n)$, we derive that the size of \mathfrak{M} is $O(2^{\frac{\text{tw}}{2}} \cdot 3^{\frac{\text{tw}}{2}} \cdot \text{tw}^{O(1)} \cdot n) = O(\sqrt{6}^{\text{tw}} \cdot \text{tw}^{O(1)} \cdot n)$. So, clearly, the computation of all entries corresponding to leaves, Forget nodes and Introduce nodes can be done within this time bound. The computation of all Join nodes can also be done within this time bound by the use of *fast subset convolution* in the exact same manner as it is done for the known exact algorithm for WEIGHTED DOMINATING SET parameterized by tw (see Section 11.1 in [4]).

Correctness can be proved by straightforward induction on the order of the computation (following the same lines as for the exact algorithm for WEIGHTED DOMINATING SET parameterized by tw).

▷ **Claim 8.** Every entry $\mathfrak{M}[x, (X, Y), (\widehat{X}, \widehat{Y}_1, \widehat{Y}_2)]$ stores the minimum weight of a dominating set S of $G[\gamma(x)]$ that satisfies:

- $S \cap \beta(x) = X \cup \widehat{X}$.
- S dominated \widehat{Y}_1 .

If such a matching does not exist, then the entry stores ∞ .

This completes the proof. ◀

For the proof of Theorem 4, we also need the following result.

► **Proposition 9** ([17], Corollary). *There exists an $O(n \cdot \text{tw})$ -time algorithm that, given a graph G and a tree decomposition $\mathcal{T} = (T, \beta)$ of G of width tw , outputs a partition (V_1, V_2) of $V(G)$ such that for every $i \in \{1, 2\}$ and $x \in V(T)$, $|\beta(x) \cap V_i| \leq \frac{\text{tw}}{2} + O(1)$.*

We proceed with the following immediate observation.

► **Observation 10.** *Let (G, w) be an instance of WEIGHTED DOMINATING SET, and let $D \subseteq V(G)$. Then,*

1. *Any dominating set $S \subseteq V(G)$ of G dominates both D and $V(G) \setminus D$*
2. *Let $S_1 \subseteq V(G)$ dominate D , and $S_2 \subseteq V(G)$ dominate $V(G) \setminus D$. Then, $S_1 \cup S_2$ is a dominating set of G .*

Now, we are ready to conclude the correctness of Theorem 4.

Proof of Theorem 4. We first describe the algorithm. Let (G, w, \mathcal{T}) be an instance of WEIGHTED DOMINATING SET parameterized by tw . Due to Proposition 3, we can suppose that \mathcal{T} is nice. First, we call the algorithm of Proposition 9 with (G, w, \mathcal{T}) as input, and let (V_1, V_2) denote its outputs. Then, we call the algorithm of Lemma 7 twice, once with (G, w, \mathcal{T}, V_1) as input and once with (G, w, \mathcal{T}, V_2) as input, and let S_1 and S_2 denote their outputs. We return $S = S_1 \cup S_2$.

Clearly, due to Proposition 9 and Lemma 7, the algorithm runs in $O(\sqrt{6}^{\text{tw}} \cdot \text{tw}^{O(1)} \cdot n)$ time. For correctness, first note that due to the second item in Observation 10, the output set S is a dominating set of G . Moreover, due to the first item in Observation 10, the optimums of (G, w, \mathcal{T}, V_1) and (G, w, \mathcal{T}, V_2) as instances of WEIGHTED HALF-WIDTH DOMINATION are both bounded from above by the optimum of (G, w, \mathcal{T}) as an instance of WEIGHTED DOMINATING SET. Hence, both of $w(S_1), w(S_2)$ are bounded from above by the optimum of (G, w, \mathcal{T}) as an instance of WEIGHTED DOMINATING SET, which implies that $w(S)$ is bounded from above by twice the optimum of (G, w, \mathcal{T}) as an instance of WEIGHTED DOMINATING SET. This completes the proof. ◀

4 Parameter: Size of Vertex Modulator to Constant Treewidth

Similarly to the proof in Section 3, will use a translation of an instance of (WEIGHTED) DOMINATING SET into two instances of two easier problems, to attain a 2-approximation algorithm for (WEIGHTED) DOMINATING SET. Here, however, the translation is somewhat different.

► **Theorem 11.** *For any fixed constant $d \geq 1$, the WEIGHTED DOMINATING SET problem parameterized by tw_d admits a 2-approximation $O(2^{\text{tw}_d} \cdot n)$ -time algorithm.*

Towards the proof of this theorem, we first define the two easier problems that we will aim to solve.

► **Definition 12** (Modulator Domination). *Let $d \in \mathbb{N} \cup \{0\}$. In the WEIGHTED d -MODULATOR DOMINATION problem, the input consists of a graph G , a vertex-weight function $w : V(G) \rightarrow \mathbb{N}$, and a subset $M \subseteq V(G)$ such that the treewidth of $G - M$ is at most d . The objective is to compute a subset $S \subseteq V(G)$ of minimum weight that dominates M .*

► **Definition 13** (Decomposition Domination). *Let $d \in \mathbb{N} \cup \{0\}$. In the WEIGHTED d -DECOMPOSITION DOMINATION problem, the input consists of a graph G , a vertex-weight function $w : V(G) \rightarrow \mathbb{N}$, and a subset $M \subseteq V(G)$ such that the treewidth of $G - M$ is at most d . The objective is to compute a subset $S \subseteq V(G)$ of minimum weight that dominates $V(G) \setminus M$.*

To reuse the result for MODULATOR DOMINATION in Section 5, we require a slight generalization of the problem, defined as follows.

► **Definition 14** (Generalized Modulator Domination). *Let $d \in \mathbb{N} \cup \{0\}$. In the GENERALIZED WEIGHTED d -MODULATOR DOMINATION problem, the input consists of a graph G , a vertex-weight function $w : V(G) \rightarrow \mathbb{N}$, and a subset $M \subseteq V(G)$ such that the treewidth of $G - M$ is at most d . The objective is to compute, for every subset $A \subseteq M$, a subset $S_A \subseteq V(G)$ of minimum weight that dominates A .*

Next, we present our algorithms for GENERALIZED WEIGHTED d -MODULATOR DOMINATION and WEIGHTED d -DECOMPOSITION DOMINATION. For the GENERALIZED WEIGHTED d -MODULATOR DOMINATION problem, we will use the following result.

► **Proposition 15** ([4], Implicit). *The GENERALIZED WEIGHTED SET COVER problem admits an $O(2^n \cdot m)$ -time algorithm, where n is the size of the universe and m is the size of the set-family.*

► **Lemma 16.** *The GENERALIZED WEIGHTED d -MODULATOR DOMINATION problem admits an $O(2^{|M|} \cdot n)$ -time algorithm.*

Proof. We first describe the algorithm. Let (G, w, M) be an instance of the GENERALIZED WEIGHTED d -MODULATOR DOMINATION problem. Then, we construct an instance (U, \mathcal{F}, w') of GENERALIZED WEIGHTED SET COVER as follows:

- $U = M$.
- $\mathcal{F} = \{N[v] \cap M : v \in V(G)\}$.
- For every $F \in \mathcal{F}$, let v_F be a vertex of minimum weight among the vertices $v \in V(G)$ that satisfy $F = N[v] \cap M$, and define $w'(F) = w(v_F)$.

We call the algorithm of Proposition 15 with (U, \mathcal{F}, w') as input, and, for every $A \subseteq U$, let \mathcal{S}_A be its output. Then, for every $A \subseteq M$, we return $S_A = \{v_F : F \in \mathcal{S}_A\}$.

Clearly, due to Proposition 15, the algorithm runs in $O(2^{|M|} \cdot n)$ time. For correctness, consider some $A \subseteq M$. Observe that, on the one hand, if $B \subseteq V(G)$ dominates A , then $\mathcal{B} = \{N[v] \cap A : v \in B\} \subseteq \mathcal{F}$ covers A and $w(B) \geq w'(\mathcal{B})$. On the other hand, if $\mathcal{B} \subseteq \mathcal{F}$ covers A , then $B = \{v_F : F \in \mathcal{B}\} \subseteq V(G)$ dominates A , and $w'(\mathcal{B}) = w(B)$. This completes the proof. ◀

For the WEIGHTED d -DECOMPOSITION DOMINATION problem, we will use Proposition 6 and the following result.

► **Proposition 17** ([2]). *There exists an algorithm that, given a graph G , outputs a tree decomposition of G of width $t = \text{tw}(G)$ in $t^{O(t^3)} \cdot n$ time.*

► **Lemma 18.** *The WEIGHTED d -DECOMPOSITION DOMINATION problem admits an $O(2^{|M|} \cdot n)$ -time algorithm.*

Proof. We first describe the algorithm. Let (G, w, M) be an instance of the WEIGHTED d -DECOMPOSITION DOMINATION problem. Then, for every subset $L \subseteq M$, we construct an instance $I_L = (G_L, w_L, \mathcal{T}_L)$ of WEIGHTED DOMINATING SET parameterized by tw as follows:

- Let $V(G_L) = (V(G) \setminus M) \cup \{x\}$ for $x \notin V(G)$, and $E(G_L) = E(G - M) \cup \{\{x, v\} : v \in N_G(L) \setminus M\}$. That is, we construct G_L from G by removing the vertices in M and the edges incident to them, and adding a new vertex x adjacent to all of the vertices in $N_G(L) \setminus M$.
- For every $v \in V(G_L)$, define $w_L(v) = w(v)$ if $v \in V(G) \setminus M$, and $w_L(v) = w(L)$ otherwise (for $v = x$).
- Use the algorithm of Proposition 17 with G_L as input, and let \mathcal{T}_L be its output.

31:10 Sidestepping Barriers for Dominating Set in Parameterized Complexity

Let $\mathcal{I} = \{I_L : L \subseteq M\}$. For every $I_L \in \mathcal{I}$, we call the algorithm of Proposition 6 with I_L as input, let S'_L be its output, and define S_L as S'_L if $x \notin S'_L$ and $S'_L \cup L$ otherwise. Let $\mathcal{S} = \{S_L : L \subseteq M\}$. Then, we return the set S of minimum-weight with respect to w among the sets in \mathcal{S} .

For the time complexity analysis, observe that $|\mathcal{I}| = 2^{|M|}$. Moreover, observe that for every $L \subseteq M$, $\text{tw}(G_L) \leq \text{tw}(G - M) + 1 \leq d + 1$; hence, each call to the algorithm of Proposition 17 runs in $(d + 1)^{O((d+1)^{d+1})} \cdot |V(G_L)| \leq O(n)$ time, and each call to the algorithm of Proposition 6 runs in $O(3^{d+1} \cdot |V(G_L)|) \leq O(n)$ time. Thus, the total running time of our algorithm is $O(2^{|M|} \cdot n)$.

Now, we turn to consider the correctness of the algorithm. To this end, consider some subset $L \subseteq M$. On the one hand, consider some subset $A \subseteq V(G)$ that satisfies $A \cap M = L$ and A dominates $V(G) \setminus M$. Let $A' = (A \setminus M) \cup \{x\}$. Then, $A \setminus M$ dominates $V(G_L) \setminus N_G(L)$ and x dominates $N_G(L)$, hence A' dominates $V(G_L)$, and our definition of w_L directly implies that $w(A) = w_L(A')$. On the other hand, consider some subset $A' \subseteq V(G_L)$ that dominates $V(G_L)$. Then, define A as A' if $x \notin A'$ and $A' \cup L$ otherwise. So, it is easy to see that A dominates $V(G) \setminus M$ and $w_L(A') = w(A)$.

We conclude that, on the one hand, if $A \subseteq V(G)$ dominates M , then, for $L = A \cap M$, a minimum-weight dominating set of G_L with respect to w_L is of weight $w(A)$. So, the output dominating set cannot have weight larger than $w(A)$. On the other hand, for every $L \subseteq M$, the minimum weight of a dominating set of G_L with respect to w_L is bounded from below by the minimum weight of a dominating set of G with respect to w . So, obviously, the output dominating set cannot have weight larger than the minimum one. This completes the proof. \blacktriangleleft

Now, we are ready to conclude the correctness of Theorem 11.

Proof of Theorem 11. We first describe the algorithm. Let (G, w, M) be an instance of WEIGHTED DOMINATING SET parameterized by tw_d . Then, we call the algorithms of Lemmas 16 and 18 with (G, w, M) as input, and let S_1 and S_2 denote their outputs. We return $S = S_1 \cup S_2$.

Clearly, due to Lemmas 16 and 18, and since $|M| = \text{tw}_d$, the algorithm runs in $O(2^{\text{tw}_d} \cdot n)$ time. For correctness, first note that due to the second item in Observation 10, the output set S is a dominating set of G . Moreover, due to the first item in Observation 10, the optimum of (G, w, M) as an instance of WEIGHTED d -MODULATOR DOMINATION (or WEIGHTED d -DECOMPOSITION DOMINATION) is bounded from above by the optimum of (G, w, M) as an instance of WEIGHTED DOMINATING SET. Hence, both of $w(S_1), w(S_2)$ are bounded from above by the optimum of (G, w, M) as an instance of WEIGHTED DOMINATING SET, which implies that $w(S)$ is bounded from above by twice the optimum of (G, w, M) as an instance of WEIGHTED DOMINATING SET. This completes the proof. \blacktriangleleft

5 Parameter: Vertex Cover Number

In this section, we prove that in the case of $\text{vc} = \text{tw}_0$, we can attain an exact algorithm with the same running time as in Theorem 11.

► **Theorem 19.** *The WEIGHTED DOMINATING SET problem parameterized by vc admits a $O(2^{\text{vc}} \cdot n)$ -time algorithm.*

Proof. We suppose that the input also consists of a subset $M \subseteq V(G)$ that is a vertex cover of G of size vc , since such a subset can be easily computed in $O(2^{\text{vc}} \cdot n)$ time [8, 4]. Now, we describe the algorithm. Let (G, w, M) be an instance of WEIGHTED DOMINATING SET parametrized by vc . We perform the following steps:

1. Call the algorithm of Lemma 16 with (G, w, M) as input of WEIGHTED 0-MODULATOR DOMINATION. Let $\{\tilde{S}_A : A \subseteq M\}$ be its output.
2. For every $A \subseteq M$:
 - a. Let $\hat{S}_A = A \cup (V(G) \setminus (N_G(A) \cup M))$. That is, \hat{S}_A is the union of A and the set of vertices in the independent set $V(G) \setminus M$ that are not dominated by the vertices in A .
 - b. Let $S_A = \hat{S}_A \cup \tilde{S}_{M \setminus N_G[\hat{S}_A]}$. Notice that $M \setminus N_G[\hat{S}_A]$ is the set of vertices in M that are not dominated by the vertices in \hat{S}_A .
3. Return the set S of minimum weight among the sets in $\{S_A : A \subseteq M\}$.

Clearly, due to Lemma 16, the algorithm runs in $O(2^{vc} \cdot n)$ time. Moreover, it is clear that the output set S is a dominating set of G . So, it remains to show that S is of minimum weight among all dominating sets of G . To this end, let S^* be a dominating set of G of minimum weight. Consider the iteration of the algorithm that corresponds to $A^* = S^* \cap U$. Notice that, since S^* dominates $V(G) \setminus M$ which is an independent set, it must hold that $V(G) \setminus (N_G(A^*) \cup M) \subseteq S^*$. So, $\hat{S}_{A^*} \subseteq S^*$. Further, since S^* dominates $M \setminus N_G[\hat{S}_{A^*}]$, we have that $S^* \setminus \hat{S}_{A^*}$ dominates $M \setminus N_G[\hat{S}_{A^*}]$. By the correctness of the algorithm of Lemma 16, this implies that $w(\tilde{S}_{M \setminus N_G[\hat{S}_{A^*}]}) \leq w(S^* \setminus \hat{S}_{A^*})$. Thus, we conclude that $w(S) \leq w(S_{A^*}) \leq w(S^*)$. ◀

Additionally, we observe that the time complexity in Theorem 11 is tight. Due to lack of space, the proof is deferred to the full version of this paper.

► **Observation 20.** *Under any of the SETH and the Set Cover Conjecture, the DOMINATING SET problem parameterized by vc cannot be solved in $O((2 - \epsilon)^{vc} \cdot n)$ time for any fixed $\epsilon > 0$.*

6 Parameter: Feedback Edge Set Number: FPT Algorithm

In this section, we first prove a combinatorial result (stated in Lemma 22). In particular, this result implies a parameterized algorithm where the basis of the exponent is smaller than 3 (stated in Theorem 29). For our combinatorial result, we will use the following proposition.

► **Proposition 21** ([3]). *The treewidth of a cactus is at most 2.*

► **Lemma 22.** *For any graph G , $\text{tw}_2(G) \leq \frac{\text{fes}(G)}{2}$. Moreover, there exists an algorithm that, given a graph G , outputs a subset $M \subseteq V(G)$ such that $|M| \leq \frac{\text{fes}(G)}{2}$ and $\text{tw}(G - M) \leq 2$ in $O(\text{fes}(G) + n)$ time.*

The idea behind the algorithm presented in the proof is quite simple (though, perhaps, if we did not demand it to run in $O(\text{fes}(G) + n)$ time, it could have been further simplified). Specifically, we scan a depth-first search (DFS) tree T of G from top to bottom. For each vertex that we remove (and insert into M), we aim to argue that at least two edges in $F = E(G) \setminus E(T)$ have become “irrelevant” – that is, not part of any cycle. To identify which vertices to remove, we maintain a variable e , which stores an edge from F whose “top” is above (or equal to) and whose “bottom” is below (or equal to) the vertex currently under consideration, and, most importantly, which is still “relevant”. When no such edge exists, it stores nil. In particular, we notice two situations where we can (and it suffices) to remove a vertex: first, when it is the top of two edges from F , and second, when it is the top of an edge from F and e is some other edge from F . We now proceed to present the formal description of the algorithm and its proof.

31:12 Sidestepping Barriers for Dominating Set in Parameterized Complexity

Proof of Lemma 22. To describe the algorithm, let G be a graph. Without loss of generality, we suppose that G is connected, else we can consider each of its connected components separately. We compute a DFS tree T of G . Let $F = E(G) \setminus E(T)$, and note that $|F| = \text{fes}(G)$. Given an edge $e \in F$, we refer to the *top* and *bottom* of e as the endpoint of e that is closer to the root of T and the other endpoint of e , respectively. (Since T is a depth-first search tree, the terms top and bottom are uniquely defined.)

Initialize $M = \emptyset$ and $e = \text{nil}$. For every $v \in V(T)$ where T is traversed in preorder, we perform the following computation:

1. If v is the bottom of e (in this case, $e \neq \text{nil}$), update $e = \text{nil}$.
2. If v is not the top of any edge in F , we proceed to the next iteration.
3. If either v is the top of at least two edges in F or $e \neq \text{nil}$, then:
 - a. Insert v into M .
 - b. Update $e = \text{nil}$.
 - c. Proceed to the next iteration.
4. Update e to be the edge in F whose top is v , and prioritize the preorder traversal to first visit the vertices on the subpath of T from v to the bottom of e . (In case a previous prioritization exists, override it.)

At the end, we return the set M .

Clearly, the algorithm runs in $O(n + m) = O(\text{fes}(G) + n)$ time.

We now turn to consider the correctness of the algorithm. Towards that, we define the following terminology. Given an edge $e \in F$, let the *span* of e be the subpath of T from the top to bottom of e , and let the *truncated span* of e be the subpath that results from the removal of the bottom of e from the span of e . We say that two distinct edges $e, e' \in F$ have a *conflict* if the top of one of them belongs to truncated span of the other. Given an edge $e \in F$ and a subset $M \subseteq V(G)$, we say that e is *active* in M if $G - M$ contains a cycle that traverses e . Observe that all of the edges in F are active in \emptyset .

Towards the proof of our main inductive claim, we present the following claim.

▷ **Claim 23.** Let C be a cycle in G , $e \in E(C) \cap F$, and suppose that C is not the cycle formed by e and its span. Then, C contains an edge $e' \in F \setminus \{e\}$ that has a conflict with e , and whose top is either the top of e or an ancestor of it.

Proof. Let t and b be the top and bottom of e , respectively. Targeting a contradiction, we assume that C does not contain an edge $e' \in F \setminus \{e\}$ that has a conflict with e , and whose top is either t or an ancestor of t . Let P denote the subpath of C between t and b that does not contain e . Due to our assumption, this path cannot contain an edge between t or an ancestor of t and a descendant of t , with the exception of the edge between t and its children in T , because such an edge must belong to F and have a conflict with e . Due to this, and because T is a depth-first tree, P cannot contain an edge between a vertex that is not a descendant of t and a descendant of t , with the exception of the edge between t and its child that belongs to the span of e , which we denote by c . So far, we conclude that P does not contain any ancestor of t and that it contains the edge $\{t, c\}$. However, again, because T is a depth-first tree, P also cannot contain an edge between a vertex that is a descendant of a vertex, say, x , that belongs to the span of e and a vertex that is neither x nor an ancestor of x . In turn, this implies that P is equal to the span of e , which is a contradiction to the supposition of the claim that C is not the cycle formed by e and its span. ◁

Now, we are ready to present our main inductive argument.

▷ **Claim 24.** Consider an iteration of the preorder traversal. Let M' be the set M at the end of this iteration. Let e' denote the value of e at the end of this iteration. Let v be the vertex traversed in this iteration. Then:

1. The set M' does not contain any descendant of v .
2. There do not exist two edges in F that are active in M' , have a conflict and the top of each one of them is either v or an ancestor of v in T .
3. If $e' = \text{nil}$, then there does not exist an edge in F that is active in M' and such that v belongs to the truncated span of that edge.
4. If $e' \neq \text{nil}$, then: (i) v belongs to the truncated span of e' ; (ii) there does not exist an edge in F other than e' that is active in M' and such that v belongs to the truncated span of that edge; (iii) M' does not contain any vertex from the span of e' . (In particular due to item 1 and (iii), e' is active, and this is witnessed by the cycle formed by e' and its span.)

Proof. We use induction on the preorder traversal. Consider the first iteration, where v is the root of T and, hence, the only edges in F such that v belongs to their span are those that have v as their top. Then, all of the items in the claim directly follow from the pseudocode.

Now, consider an iteration that is not the first, and suppose that the claim is correct up to this iteration. By the inductive hypothesis (item 1) and the pseudocode, it should be clear that item 1 of the claim holds. Let M'' and e'' denote the values of M and e at the beginning of the iteration. Let u be the parent of v in T . By the inductive hypothesis (item 2), there do not exist two edges in F that are active in M'' , have a conflict and the top of each one of them is an ancestor of u in T . Yet, to prove item 2 of the claim, we still need to argue that there do not exist two edges in F that are active in M' , have a conflict, the top of one of them is v , and the top of the other is either v or an ancestor of v . Note that if there exist two such edges, then v belongs to the truncated span of both of these edges. We consider the two following cases.

- First, suppose that $e'' = \text{nil}$. Then, by the inductive hypothesis (item 3), there does not exist an edge in F that is active in M'' and such that u belongs to the truncated span of that edge. So, the only edges that are active in M'' and such that v belongs to their span are those that have v as their top. If v is not the top of any edge in F , then $e' = \text{nil}$, $M' = M''$, and items 2 and 3 of the claim follow. If v is the top of at least two edges in F , then $e' = \text{nil}$, $M' = M'' \cup \{v\}$ (so, these edges are non-active in M'), and items 2 and 3 of the claim follow. If v is the top of exactly one edge in F , then this edge is e' (and $M' = M''$), and, hence, items 2 and 4 of the claim follow.
- Second, suppose that $e'' \neq \text{nil}$. Then, by the inductive hypothesis (item 4), u belongs to the truncated span of e'' , there does not exist an edge in F other than e'' that is active in M'' and such that u belongs to the truncated span of that edge, and M'' does not contain any vertex from the span of e'' . We further consider the three following sub-cases:
 1. First, suppose that v is the bottom of e'' . This implies that the only edges that are active in M'' and such that v belongs to their span are those that have v as their top. Then, e is updated to be nil in the first step of the iteration, and the proof proceeds as in the first case.
 2. Second, suppose that v is neither the bottom of e'' nor the top of any edge in F . This implies that v belongs to the truncated span of e'' , and that there does not exist an edge in F other than e'' that is active in M'' and such that v belongs to the span of that edge. As $e' = e''$ and $M' = M''$, items 2 and 4 of the claim follow.
 3. Third, suppose that v is not the bottom of e'' , and that v is the top of at least one edge in F . Then, $e' = \text{nil}$ and $M' = M'' \cup \{v\}$. Hence, there does not exist an edge in F that is active in M' and has v as its top. Hence, item 2 of the claim follows, and to complete the proof of item 4 of the claim, it suffices to show that e'' is non-active in M' .

31:14 Sidestepping Barriers for Dominating Set in Parameterized Complexity

Targeting a contradiction, suppose that e'' is active in M' , and let t and b denote its top and bottom, respectively. Then, there exists a cycle C in $G - M'$ that contains e'' . In particular, there exists a path P in $G - M'$ between t and b that does not contain e'' . Due to Claim 23, if P is not equal to the span of e'' , then C contains an edge $\hat{e} \in F \setminus \{e''\}$ that has a conflict with e'' and whose top is either t or an ancestor of t , and because this edge belongs to C (which exists in $G - M'$), it must be active in M' ; however, this is a contradiction to item 2 of the claim. Thus, P is equal to the span of e'' , which is a contradiction, since v belongs to this span as well as to M' . So, e'' is non-active in M' .

This completes the proof. \triangleleft

We proceed to prove the following claim, which will imply the desired bound the size of M .

\triangleright **Claim 25.** Consider an iteration of the preorder traversal. Let M' be the set M at the end of this iteration. Suppose that in this iteration, the vertex v was inserted into M' . Then, there exist two distinct edges in F that are active in $M' \setminus \{v\}$ but are non-active in M' .

Proof. Let e'' denote the value of e at the start of this iteration. Then, one of the two following cases holds.

Case I. Suppose that v is the top of at least two edges in F , say, e_1^v and e_2^v . Then, due to item 1 of Claim 24, both of these edges are active in $M' \setminus \{v\}$ (witnesses by the cycles formed by these edges and their spans). However, both of these edges clearly become non-active in M' .

Case II. Suppose that $e'' \neq \text{nil}$ and v is the top of exactly one edge in F , denoted by e^v . Note that $e'' \neq e^v$, since the value of e is updated when its top is traversed (and v is only being traversed in the current iteration, after e already holds e''). As in Case I, e^v is active in $M' \setminus \{v\}$ but becomes non-active in M' . By item 4 of Claim 24 with respect to the previous iteration, e'' is active in $M' \setminus \{v\}$, and by the same item with respect to the current iteration, e'' is non-active in M' .

In either case, we conclude that the claim holds. \triangleleft

In particular, from Claim 25 we conclude that $|M| \leq \frac{|F|}{2} = \frac{\text{fes}(G)}{2}$. (For every vertex inserted into M , at least two edges in F that are active at that moment become non-active, and they never become active again later).

In order to bound the treewidth of $G - M$, we turn to prove several additional claims.

\triangleright **Claim 26.** Let $X \subseteq V(G)$. Then, $\{e \in F : e \text{ is active in } X\}$ is a feedback edge set of $G - X$.

Proof. The claim directly follows from the definition of active edges, and because F is a feedback edge set of G . \triangleleft

\triangleright **Claim 27.** Let $X \subseteq V(G)$. Let C, C' be two distinct cycles in $G - X$ that have at least two vertices in common. Then, there exist two distinct edges $e, e' \in F$ that are active in X and have a conflict.

Proof. By Claim 23 and since C, C' belong to $G - X$, we can assume that C and C' are the cycles that consist of some edges $e, e' \in F$ and their spans, respectively, else the proof is complete. Since C and C' have at least two vertices in common, the intersection of the spans of e and e' must be of size at least 2. However, this implies that the top of one of them must belong to the truncated span of the other, and hence they have a conflict. \triangleleft

\triangleright Claim 28. There do not exist two distinct edges $e, e' \in F$ that are active in M and have a conflict.

Proof. The claim directly follows from item 2 of Claim 24 by considering the iterations in which the leaves of T were traversed. \triangleleft

From Claims 27 and 28, we derive that $G - M$ is a cactus graph. So, by Proposition 21, we conclude that its treewidth is at most 2. This completes the proof. \blacktriangleleft

\blacktriangleright **Theorem 29.** *The WEIGHTED DOMINATING SET problem parameterized by fes admits an $O(3^{\frac{\text{fes}}{2}} \cdot n)$ -time algorithm.*

Proof. To describe the algorithm, let (G, w) be an instance of WEIGHTED DOMINATING SET. Then, we call the algorithm of Lemma 22, and let M be its output. So, $|M| \leq \frac{\text{fes}(G)}{2}$ and $\text{tw}(G - M) \leq 2$. Afterwards, we call the algorithm of Proposition 17 with $G - M$ as input, and let \mathcal{T}' be its output. So, \mathcal{T}' is a tree decomposition of width at most 2 of $G - M$. We insert M into each of the bags of \mathcal{T}' to attain a tree decomposition \mathcal{T} of G of width at most $|M| + 2 \leq \frac{\text{fes}(G)}{2} + 2$. Lastly, we call the algorithm of Proposition 6 with (G, w, \mathcal{T}) as input, and return its result.

Clearly, correctness is immediate. As for the time complexity, observe that the calls to the algorithms of Lemma 22 and Propositions 17 and 6 run in $O(\text{fes}(G) + n)$, $2^{O(2^3)} \cdot n = O(n)$ and $O(3^{\frac{\text{fes}(G)}{2} + 2} \cdot n) \leq O(3^{\frac{\text{fes}(G)}{2}} \cdot n)$ times, respectively. Thus, the total running time of our algorithm is $O(3^{\frac{\text{fes}(G)}{2}} \cdot n)$. \blacktriangleleft

7 Conclusion and Future Directions

We presented algorithmic results that sidestep time complexity barriers for DOMINATING SET. For this purpose, we incorporated approximation for the parameters solution size and treewidth, larger parameterization for the parameters vertex cover and feedback edge set compared to treewidth, or both for the parameter vertex modulator to constant treewidth compared to treewidth.

Extension of Our Approaches. While we have focused on DOMINATING SET, we believe that some of our approaches might be applicable to other problems as well. For example, consider the GRAPH COLORING problem, where, given a graph G and an integer $q \geq 3$, the objective is to determine whether G admits a proper coloring in q colors. Under the SETH, GRAPH COLORING cannot be solved in $(q - \epsilon)^{\text{tw}} \cdot n^{O(1)}$ time for any fixed $\epsilon > 0$ [16]. Then, we follow a simplification of the approach we presented in Section 4. Briefly, the idea is to consider two problems: one problem concerns the graph induced by the modulator, and the other problem concerns the rest of the graph. So, suppose we are given a subset $M \subseteq V(G)$ such that the treewidth of $G - M$ is at most d (where d is a fixed constant). On the one hand, we solve GRAPH COLORING on $G[M]$ in $2^{|M|} \cdot |M|^{O(1)}$ time using the algorithm in [15], and on the other hand, we solve GRAPH COLORING on $G - M$ in $n^{O(1)}$ time based on straightforward dynamic programming. We consider the color sets used by the two solutions

to be disjoint, thereby obtaining a 2-approximate solution in $2^{\text{tw}_d} \cdot n^{O(1)}$ time. Essentially the same approach works for INDEPENDENT SET as well, where, given a graph G and a non-negative integer k , the objective is to determine whether G admits an independent set of size at least k . Under the SETH, INDEPENDENT SET cannot be solved in $(2 - \epsilon)^{\text{tw}} \cdot n^{O(1)}$ time for any fixed $\epsilon > 0$ [16]. On the one hand, we solve INDEPENDENT SET on $G[M]$ in $1.19997^{|M|} \cdot |M|^{O(1)}$ time using the algorithm in [24], and on the other hand, we solve INDEPENDENT SET on $G - M$ in $n^{O(1)}$ time based on straightforward dynamic programming. We output the largest among the two solutions, thereby obtaining a 2-approximate solution in $1.19997^{\text{tw}_d} \cdot n^{O(1)}$ time.

Directions for Future Research. Firstly, we find the questions of improvements of the performance of our algorithms (in terms of running times and approximation ratios) interesting. In particular, does DOMINATING SET admit a 2-approximation $O(2^{\text{tw}} \cdot n)$ -time algorithm, or a $(1 + \epsilon)$ -approximation $O(2^{\text{tw}} \cdot n)$ -time algorithm for any fixed $\epsilon > 0$? Additionally, we have the following questions regarding DOMINATING SET:

1. Prove or refute the following conjecture:

► **Conjecture 30.** *Under the SETH, there exists a fixed constant $d \in \mathbb{N}$ such that (WEIGHTED) DOMINATING SET cannot be solved in $(3 - \epsilon)^{\text{tw}_d} \cdot n^{f(d)}$ time for any fixed constant $\epsilon > 0$ and function f of d , where tw_d is the minimum size of a vertex set whose deletion from G results in a graph of treewidth at most d .*

2. Study DOMINATING SET parameterized by the solution size plus the distance (e.g., number of vertex or edge deletions or contractions) to graph classes where it belongs to FPT, particularly planar graphs and claw-free graphs.
3. Conduct a similar study for problems beyond DOMINATING SET. Here, possibly and as argued above, the ideas presented in this article can be re-used.

References

- 1 Saeed Akhoondian Amiri, Patrice Ossona de Mendez, Roman Rabinovich, and Sebastian Siebertz. Distributed domination on graph classes of bounded expansion. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, SPAA '18, pages 143–151, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3210377.3210383.
- 2 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.
- 3 Hans L. Bodlaender. A partial k -arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998.
- 4 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshantov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.
- 5 Marek Cygan, Lukasz Kowalik, and Mateusz Wykurz. Exponential-time approximation of weighted set cover. *Inf. Process. Lett.*, 109(16):957–961, 2009.
- 6 Marek Cygan, Geevarghese Philip, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. Dominating set is fixed parameter tractable in claw-free graphs. *Theoretical Computer Science*, 412(50):6982–7000, November 2011. doi:10.1016/j.tcs.2011.09.010.
- 7 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.

- 8 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer Publishing Company, Incorporated, 2013.
- 9 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1):57–67, 2004. doi:10.1016/j.tcs.2004.05.009.
- 10 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, July 1998. doi:10.1145/285055.285059.
- 11 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Linear kernels for (connected) dominating set on h -minor-free graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '12*, pages 82–93, USA, 2012. Society for Industrial and Applied Mathematics.
- 12 Fedor V. Fomin and Dimitrios M. Thilikos. Dominating sets in planar graphs: Branch-width and exponential speed-up. *SIAM Journal on Computing*, 36(2):281–309, 2006. doi:10.1137/S0097539702419649.
- 13 Danny Hermelin, Matthias Mnich, Erik Jan van Leeuwen, and Gerhard J. Woeginger. Domination when the stars are out. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, pages 462–473, 2011. doi:10.1007/978-3-642-22006-7_39.
- 14 Tohru Kikuno, Noriyoshi Yoshida, and Yoshiaki Kakuda. A linear algorithm for the domination number of a series-parallel graph. *Discret. Appl. Math.*, 5:299–311, 1983.
- 15 Mikko Koivisto. An $o^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion–exclusion. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 583–590, 2006.
- 16 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2), April 2018. doi:10.1145/3170442.
- 17 Daniel Lokshtanov, Pranabendu Misra, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Fpt-approximation for FPT problems. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 199–218, 2021. doi:10.1137/1.9781611976465.14.
- 18 Dániel Marx. Parameterized Complexity of Independence and Domination on Geometric Graphs. In Hans L. Bodlaender and Michael A. Langston, editors, *Parameterized and Exact Computation*, pages 154–165, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- 19 Mihai Pătraşcu and Ryan Williams. On the possibility of faster sat algorithms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '10*, pages 1065–1075, USA, 2010. Society for Industrial and Applied Mathematics.
- 20 Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. *J. ACM*, 66(5), August 2019. doi:10.1145/3325116.
- 21 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009*, pages 566–577, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- 22 Johan M.M. van Rooij and Hans L. Bodlaender. Exact algorithms for dominating set. *Discrete Applied Mathematics*, 159(17):2147–2164, 2011. doi:10.1016/j.dam.2011.07.001.
- 23 Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001. URL: <http://www.springer.com/computer/theoretical+computer+science/book/978-3-540-65367-7>.
- 24 Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Inf. Comput.*, 255:126–146, 2017.

Approximate Turing Kernelization and Lower Bounds for Domination Problems

Stefan Kratsch  

Algorithm Engineering, Humboldt-Universität zu Berlin, Germany

Pascal Kunz  

Algorithm Engineering, Humboldt-Universität zu Berlin, Germany

Abstract

An α -approximate polynomial Turing kernelization is a polynomial-time algorithm that computes an (αc) -approximate solution for a parameterized optimization problem when given access to an oracle that can compute c -approximate solutions to instances with size bounded by a polynomial in the parameter. Hols et al. [ESA 2020] showed that a wide array of graph problems admit a $(1 + \varepsilon)$ -approximate polynomial Turing kernelization when parameterized by the treewidth of the graph and left open whether DOMINATING SET also admits such a kernelization.

We show that DOMINATING SET and several related problems parameterized by treewidth do not admit constant-factor approximate polynomial Turing kernelizations, even with respect to the much larger parameter vertex cover number, under certain reasonable complexity assumptions. On the positive side, we show that all of them do have a $(1 + \varepsilon)$ -approximate polynomial Turing kernelization for every $\varepsilon > 0$ for the joint parameterization by treewidth and maximum degree, a parameter which generalizes cutwidth, for example.

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

Keywords and phrases Approximate Turing kernelization, approximation lower bounds, exponential-time hypothesis, dominating set, capacitated dominating, connected dominating set, independent dominating set, treewidth, vertex cover number

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.32

Funding *Pascal Kunz*: Supported by the DFG Research Training Group 2434 “Facets of Complexity”.

1 Introduction

The gold standard in kernelization is a polynomial (exact) kernelization, i.e. a compression of input instances to a parameterized problem to a size that is polynomial in the parameter such that an exact solution for the original instance can be recovered from the compressed instance. Several weaker notions of kernelization have been developed for problems that do not admit polynomial kernelizations. Turing kernelization [1, 11] does away with the restriction that the solution must be recovered from a single compressed instance and instead allow several small instances to be created and the solution to be extracted from solutions to all of these instances. Lossy kernelizations [21], in turn, do away with the requirement that the solution that can be recovered from the compressed instance be an optimum solution, allowing the solution to the original instance to be worse than optimal by a constant factor. Hols et al. [12] introduced lossy Turing kernelizations, which allow both multiple compressed instances and approximate solutions, and showed that several graph problems parameterized by treewidth admit $(1 + \varepsilon)$ -approximate Turing kernelizations for every $\varepsilon > 0$. They left as an open question whether or not the problem DOMINATING SET parameterized by treewidth also admits a constant-factor approximate Turing kernelization.



© Stefan Kratsch and Pascal Kunz;

licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 32; pp. 32:1–32:17

Leibniz International Proceedings in Informatics



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

Our contribution. We answer this question in the negative and show that a $\mathcal{O}(2^{\log^c \text{vc}})$ -approximate polynomial Turing kernelization for DOMINATING SET[vc]¹, where vc refers to the vertex cover number, would contradict the Exponential Time Hypothesis. We prove analogous lower bounds for CAPACITATED DOMINATING SET[vc], CONNECTED DOMINATING SET[vc] for HITTING SET[|U|], where U is the universe, and for NODE STEINER TREE[|V \setminus T|] where $V \setminus T$ is the set of non-terminal vertices. Of course, the lower bounds for the vertex cover number also imply lower bounds for the smaller parameter treewidth. Using a second approach for obtaining lower bounds for approximate Turing kernelizations, essentially a gap-introducing polynomial parameter transformation (PPT), we show that INDEPENDENT DOMINATING SET[vc] does not have an α -approximate polynomial Turing kernelization for any constant α , unless every problem in the complexity class MK[2] has a polynomial (exact) Turing kernelization, which would contradict a conjecture by Hermelin et al. [11]. We then show that for the joint parameterization by treewidth and the maximum degree, each of the aforementioned domination problems does have a $(1+\varepsilon)$ -approximate polynomial Turing kernelization for every $\varepsilon > 0$. This generalizes, for instance, parameterization for cutwidth or bandwidth.

Related work. For an introduction to kernelization, including brief overviews on lossy and Turing kernelizations, we refer to the standard textbook [7]. Binkele-Raible et al. [1] introduced the first Turing kernelization for a problem that does not admit a polynomial kernelization. Since then numerous Turing kernelizations have been published for such problems. Hermelin et al. [11] introduced a framework, which we will make use of in Section 3.2, for ruling out (exact) polynomial Turing kernelizations. Fellows et al. [6] were the first to combine the fields of kernelization and approximation. A later study by Lokshtanov et al. [21] introduced the framework of lossy kernelization that has become more established. Finally, Hols et al. [12] gave the first approximate Turing kernelizations.

Approximation algorithms and lower bounds for domination problems have received considerable attention. They are closely related to the problems HITTING SET and SET COVER. A classical result by Chvátal [3] implies a polynomial-time $\mathcal{O}(\log n)$ -factor approximation for DOMINATING SET. There are also $\mathcal{O}(\log n)$ -factor approximations for CONNECTED DOMINATING SET [8] and CAPACITATED DOMINATING SET [23], but INDEPENDENT DOMINATING SET does not have a $\mathcal{O}(n^{1-\varepsilon})$ -approximation for any $\varepsilon > 0$, unless $\text{P} = \text{NP}$ [9]. Chlebík and Chlebíková [2] showed that DOMINATING SET, CONNECTED DOMINATING SET, and CAPACITATED DOMINATING SET do not have constant-factor approximations even on graphs with maximum degree bounded by a constant Δ and that INDEPENDENT DOMINATING SET does not have better than a Δ -factor approximation.

2 Preliminaries

Graphs. We use standard graph terminology and all graphs are undirected, simple, and finite. For a graph $G = (V, E)$ and $X \subseteq V$, we use $N[X] := X \cup \{v \in V \mid \exists u \in X: \{u, v\} \in E\}$ to denote the *closed neighborhood* of X and, if $v \in V$, then we let $N[v] := N[\{v\}]$. We will use Δ and vc to refer to the maximum degree and vertex cover number of a graph, respectively.

Let $G = (V, E)$ be a graph $X \subseteq V$ a vertex set. The set $X \subseteq V$ is a *dominating set* if $N[X] = V$. It is an *independent set* if there is no edge $\{u, v\} \in E$ with $u, v \in X$. It is an *independent dominating set* if it is both an independent and a dominating set. It is a *connected*

¹ We use FOO[X] to refer to the problem FOO parameterized by X .

dominating set if it is a dominating set and the graph $G[X]$ is connected. A *capacitated graph* $G = (V, E, \text{cap})$ consists of a graph (V, E) and a *capacity function* $\text{cap}: V \rightarrow \mathbb{N}$. A *capacitated dominating set* in a capacitated graph $G = (V, E, \text{cap})$ is a pair (X, f) where $X \subseteq V$ and $f: V \setminus X \rightarrow X$ such that (i) v and $f(v)$ are adjacent for all $v \in V \setminus X$ and (ii) $|f^{-1}(v)| \leq \text{cap}(v)$ for all $v \in X$. The size of (X, f) is $|X|$.

Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $\mathcal{T} = (T = (W, F), \{X_t\}_{t \in W})$ where T is a tree, $X_t \subseteq V$ for all $t \in W$, $\bigcup_{t \in W} X_t = V$, for each $e \in E$ there is a $t \in W$ such that $e \subseteq X_t$, and for each $v \in V$ the node set $\{t \in W \mid v \in X_t\}$ induces a connected subgraph of T . The *width* of \mathcal{T} is $\max_{t \in W} |X_t| - 1$. The *treewidth* $\text{tw}(G)$ of G is the minimum width of any tree decomposition of G . A *rooted tree decomposition* consists of a tree decomposition $\mathcal{T} = (T = (W, F), \{X_t\}_{t \in W})$ along with a designated root $r \in W$. Given this rooted decomposition and a node $t \in W$, we will use $V_t \subseteq V$ to denote the set of vertices v such that $v \in X_{t'}$ and t' is a descendant (possibly t itself) of t in the rooted tree (T, r) . The rooted tree decomposition is *nice* if $X_r = \emptyset$ and $X_t = \emptyset$ for every leaf of T and every other node t of T is of one of three types: (i) a *forget node*, in which case t has a single child t' and there is a vertex $v \in V$ such that $X_{t'} = X_t \cup \{v\}$, (ii) an *introduce node*, in which case t has a single child t' and there is a vertex $v \in V$ such that $X_t = X_{t'} \cup \{v\}$, or (iii) a *join node*, in which case t has exactly two children t_1 and t_2 and $X_t = X_{t_1} = X_{t_2}$.

Turing kernelization. A *parameterized decision problem* is a set $L \subseteq \Sigma^* \times \mathbb{N}$. A *Turing kernelization* of size $f: \mathbb{N} \rightarrow \mathbb{N}$ for a parameterized decision problem L is a polynomial-time algorithm that receives as input an instance $(x, k) \in \Sigma^* \times \mathbb{N}$ and access to an oracle that, for any instance $(x', k') \in \Sigma^* \times \mathbb{N}$ with $|x'| + k' \leq f(k)$, outputs whether $(x', k') \in L$ in a single step, and decides whether $(x, k) \in L$. It is a *polynomial Turing kernel* if f is polynomially bounded.

A *polynomial parameter transformation* (PPT) from one parameterized decision problem L to a second such problem L' is a polynomial-time computable function $f: \Sigma^* \times \mathbb{N} \rightarrow \Sigma^* \times \mathbb{N}$ such that $(x, k) \in L$ if and only if $(x', k') \in L'$ and there is a polynomially bounded function p such that $k' \leq p(k)$ for all $(x, k), (x', k') \in \Sigma^* \times \mathbb{N}$ with $f(x, k) = (x', k')$.

A *parameterized minimization problem* is defined by a computable function $\mathcal{P}: \Sigma^* \times \mathbb{N} \times \Sigma^* \rightarrow \mathbb{R} \cup \{\infty, -\infty\}$. The optimum value for an instance $(I, k) \in \Sigma^* \times \mathbb{N}$ is $\text{OPT}_{\mathcal{P}}(I, k) := \min_{x \in \Sigma^*} \mathcal{P}(I, k, x)$. We will say that a solution $x \in \Sigma^*$ is α -*approximate* if $\mathcal{P}(I, k, x) \leq \alpha \cdot \text{OPT}_{\mathcal{P}}(I, k)$. In order to simplify notation, we will allow ourselves to write $\mathcal{P}(I, x)$ instead of $\mathcal{P}(I, k, x)$ and $\text{OPT}_{\mathcal{P}}(I)$ instead of $\text{OPT}_{\mathcal{P}}(I, k)$ if those values do not depend on k . The problems (CAPACITATED/CONNECTED/INDEPENDENT) DOMINATING SET are defined by $\mathcal{P}(G, X) := |X|$ if X is a (capacitated/connected/independent) dominating set in G and $\mathcal{P}(G, X) := \infty$, otherwise. The problem NODE STEINER TREE is defined by $\text{NST}((G, T), X) := |X|$ if $G[X \cup T]$ is connected and $\text{NST}((G, T), X) := \infty$, otherwise. HITTING SET is a problem whose input (U, \mathcal{S}) consists of a set U and a family $\mathcal{S} \subseteq 2^U$ of nonempty sets, a solution X is a subset of U , and $\text{HS}(X) := \infty$ if there is an $S \in \mathcal{S}$ such that $X \cap S = \emptyset$ and $\text{HS}(X) := |X|$, otherwise.

Let $\alpha \in \mathbb{R}$ with $\alpha \geq 1$ and let \mathcal{P} be a parameterized minimization problem. An α -*approximate Turing kernelization* of size $f: \mathbb{N} \rightarrow \mathbb{N}$ for \mathcal{P} is a polynomial-time algorithm that given an instance (I, k) computes a $(c\alpha)$ -approximate solution when given access to an oracle for \mathcal{P} which outputs a c -approximate solution to any instance (I', k') with $|I'| + k' \leq f(k)$ in a single step. It is an α -*approximate polynomial Turing kernelization* if f is polynomially bounded. Note that the algorithm is not given access to c , the approximation factor of the oracle, and is not allowed to depend on c . In practice, it can also be helpful

for the approximate Turing kernelization algorithm to receive a witness for the parameter value k as input. Similarly to Hols et al. [12], we will assume that our approximate Turing kernelizations for the parameterization treewidth plus maximum degree are given as input a graph G and a nice tree decomposition of width $\text{tw}(G)$. Alternatively, one could also use the polynomial-time algorithm due to Feige et al. [5] to compute a tree decomposition of width $\mathcal{O}(\sqrt{\log \text{tw}(G)} \cdot \text{tw}(G))$ and then use this tree decomposition. We will also assume that the given tree decomposition is nice, which is not really a restriction, because there is a polynomial-time algorithm that converts any tree decomposition into a nice tree decomposition without changing the width [18].

3 Lower bounds

3.1 Exponential-time hypothesis

In the following, we will show that several problems do not have an approximate Turing kernelization assuming the exponential-time hypothesis (ETH). The proof builds on a proof due to Lokshtanov et al. [20, Theorem 12] showing that HITTING SET parameterized by the size of the universe does not admit a lossy (Karp) kernelization unless the ETH fails.

Let 3-CNF-SAT denote the satisfiability problem for Boolean formulas in conjunctive normal form with at most three literals in each clause. The *exponential-time hypothesis* (ETH) [13] states that there is a fixed $c > 0$ such that 3-CNF-SAT is not solvable in time $2^{cn} \cdot (n + m)^{\mathcal{O}(1)}$, where n and m are the numbers of variables and clauses, respectively.

Let \mathcal{P} and \mathcal{P}' be parameterized minimization problems and $f: \Sigma^* \times \mathbb{N} \rightarrow \mathbb{R}_+$ a real-valued function that takes instances of \mathcal{P} as input. An *f -approximation-preserving polynomial parameter transformation* (f -APPT) from \mathcal{P} to \mathcal{P}' consists of two algorithms:

- a polynomial-time algorithm \mathcal{A} (the *reduction algorithm*) that receives as input an instance (I, k) for \mathcal{P} and outputs an instance (I', k') for \mathcal{P}' with $k' \leq p(k)$ for some polynomially bounded function p and
- a polynomial-time algorithm \mathcal{B} (the *lifting algorithm*) that receives as input the instances (I, k) , (I', k') , where the latter is the output of \mathcal{A} when given the former, as well as a solution x for (I', k') and outputs a solution y for (I, k) with

$$\mathcal{P}(I, k, y) \leq \frac{f(I, k) \cdot \text{OPT}_{\mathcal{P}}(I, k) \cdot \mathcal{P}'(I', k, x)}{\text{OPT}_{\mathcal{P}'}(I', k)}.$$

We will use the following lemma, which is a weaker version of a result by Nelson [22].

► **Lemma 1** ([22]). *If there are a constant $c < 1$ and a polynomial-time algorithm that computes an $\mathcal{O}(2^{\log^c |U|})$ -factor approximation for HITTING SET, then the ETH fails.*

► **Lemma 2.** *Let \mathcal{P} be a parameterized minimization problem that satisfies the following two conditions:*

- (a) *There is an f -APPT from HITTING SET[$|U|$] to \mathcal{P} with $f(U, \mathcal{S}) \in \mathcal{O}(2^{\log^{c_1} |U|})$ for some constant $c_1 < 1$.*
- (b) *There is a constant $c_2 < 1$ and a polynomial-time algorithm that computes a $\mathcal{O}(2^{\log^{c_2} |I|})$ -factor approximation for \mathcal{P} where I is an instance for \mathcal{P} .*

Then, there is no $\mathcal{O}(2^{\log^{c_3} k})$ -approximate polynomial Turing kernelization for \mathcal{P} for any $c_3 < 1$, unless the ETH fails.

Proof. Suppose that \mathcal{P} satisfies conditions (a) and (b) and admits a $\mathcal{O}(2^{\log^{c_3} k})$ -approximate polynomial Turing kernelization of size $\mathcal{O}(k^d)$. Furthermore, assume that $p(n) \leq \mathcal{O}(n^{d'})$ where p is the polynomial parameter bound for the reduction algorithm of the f -APPT. Choose any constant c with $\max\{c_1, c_2, c_3\} < c < 1$ and observe that for any constant α and $i \in \{1, 2, 3\}$ we have that $2^{\alpha \log^{c_i} n} \leq \mathcal{O}(2^{\log^c n})$. We will give a $\mathcal{O}(2^{\log^c |U|})$ -approximation algorithm for HITTING SET. By Lemma 1, this proves the claim.

The algorithm proceeds as follows. Given an instance $I = (U, \mathcal{S})$ of HITTING SET as input, it first applies the reduction algorithm of the APPT to obtain an instance (I', k) of \mathcal{P} . Then, it runs the given approximate Turing kernelization on (I', k) . Whenever this Turing kernelization queries the oracle, this query is answered by running the approximation algorithm given by condition (b). Once the Turing kernelization outputs a solution X , the algorithm calls the lifting algorithm of the APPT on X , $(I, |U|)$, and (I', k) . The algorithm outputs the solution Y given by the lifting algorithm.

It remains to show that $|Y| \leq \mathcal{O}(2^{\log^{c_3} |U|} \cdot \text{OPT}_{\text{HS}}(I))$. First, observe that the $\mathcal{O}(2^{\log^{c_2} |I|})$ -factor approximation algorithm is only run on instances (J, ℓ) with $|J| \leq \mathcal{O}(k^d)$, so it always outputs a solution Z with $\mathcal{P}(J, \ell, Z) \leq \mathcal{O}(2^{\log^{c_2} k^d} \cdot \text{OPT}_{\mathcal{P}}(J, \ell))$. Hence, in the algorithm described above the Turing kernelization is given a $\mathcal{O}(2^{d \log^{c_2} k})$ -approximate oracle, so it follows that $\mathcal{P}(I', k, X) \leq \mathcal{O}(2^{\log^{c_2} k} \cdot 2^{d \log^{c_3} k} \cdot \text{OPT}_{\mathcal{P}}(I', k))$. Since $k \leq \mathcal{O}(|U|^{d'})$, it follows that $\mathcal{P}(I', k, X) \leq \mathcal{O}(2^{\log^{c_3} |U|^{d'}} \cdot 2^{d \log^{c_2} |U|^{d'}} \cdot \text{OPT}_{\mathcal{P}}(I', k))$. Therefore:

$$\begin{aligned} |Y| &\leq \frac{f(I, k) \cdot \text{OPT}_{\text{HS}}(I) \cdot \mathcal{P}(I', k, X)}{\text{OPT}_{\mathcal{P}}(I', k)} \leq \mathcal{O}\left(2^{\log^{c_3} |U|^{d'}} \cdot 2^{d \log^{c_2} |U|^{d'}} \cdot f(I, k) \cdot \text{OPT}_{\text{HS}}(I)\right) \\ &\leq \mathcal{O}\left(2^{\log^{c_3} |U|^{d'}} \cdot 2^{d \log^{c_2} |U|^{d'}} \cdot f(I) \cdot \text{OPT}_{\text{HS}}(I)\right) \\ &\leq \mathcal{O}\left(2^{\log^{c_3} |U|^{d'}} \cdot 2^{d \log^{c_2} |U|^{d'}} \cdot 2^{\log^{c_1} |U|} \cdot \text{OPT}_{\text{HS}}(I)\right) \\ &\leq \mathcal{O}(2^{\log^c |U|} \cdot \text{OPT}_{\text{HS}}(I)). \end{aligned} \quad \blacktriangleleft$$

With Lemma 2, we can prove approximate Turing kernelization lower bounds for several parameterized minimization problems.

► **Theorem 3.** *Unless the ETH fails, there are no $\mathcal{O}(2^{\log^c k})$ -approximate polynomial Turing kernels, for any $c < 1$ and where k denotes the respective parameter, for the following parameterized minimization problems:*

- (i) HITTING SET $[|U|]$,
- (ii) DOMINATING SET $[vc]$,
- (iii) CAPACITATED DOMINATING SET $[vc]$,
- (iv) CONNECTED DOMINATING SET $[vc]$, and
- (v) NODE STEINER TREE $[V \setminus T]$.

Proof. For each problem, we will prove conditions (a) and (b) from Lemma 2.

- (i) (a) Immediate.
- (b) Chvátal [3] gives a $\mathcal{O}(\log |S|)$ -factor approximation algorithm.
- (ii) (a) The following folklore reduction is a 1-APPT. Let (U, \mathcal{S}) be an instance of hitting set. The algorithm \mathcal{A} creates a graph G as follows. For every $x \in U$ and for every $S \in \mathcal{S}$, G contains vertices v_x and w_S , respectively, and G also contains an additional vertex u . The vertices $\{v_x \mid x \in U\} \cup \{u\}$ form a clique and there is an edge between v_x and w_S if and only if $x \in S$. Observe that the vertices $\{v_x \mid x \in U\}$ form a vertex cover in G , so clearly $\text{vc}(G) \leq |U|$, and that $\text{OPT}_{\text{HS}}(U, \mathcal{S}) = \text{OPT}_{\text{DS}}(G)$. Let X be a dominating set in G . Let X' be obtained from X by removing z and replacing any w_S by an arbitrary v_x with $x \in S$ (such an element u must

exist, as we assume that all $S \in \mathcal{S}$ are non-empty). The algorithm \mathcal{B} outputs $Y := \{x \in U \mid v_x \in X'\}$. This set is a hitting set, because for any $S \in \mathcal{S}$ one of the following cases applies: (i) $w_S \notin X$, meaning that X contains a neighbor v_x of w_S . Then, also $x \in X'$ and, hence $x \in Y$ and $x \in S$. (ii) $w_S \in X$, meaning that w_S is replaced by v_x with $x \in S$ when creating X' . Then $x \in Y$ and $x \in S$. Finally, $|Y| = |X| = \frac{\text{OPT}_{\text{HS}}(U, \mathcal{S}) \cdot |X|}{\text{OPT}_{\text{DS}}(G)}$, since $\text{OPT}_{\text{HS}}(U, \mathcal{S}) = \text{OPT}_{\text{DS}}(G)$.

- (b) The $\mathcal{O}(\log n)$ -factor approximation for HITTING SET [3] can also be used in a straightforward manner to approximate DOMINATING SET.
- (iii) (a) Any instance of DOMINATING SET can be transformed into an equivalent instance of CAPACITATED DOMINATING SET by setting $\text{cap}(v) := \deg(v)$ for all vertices v . The claim then follows by the same argument as for DOMINATING SET.
 - (b) Wolsey [23] gives a $\mathcal{O}(\log|\mathcal{S}|)$ factor approximation for CAPACITATED HITTING SET which can be adapted to approximate CAPACITATED DOMINATING SET.
- (iv) (a) The APPT given in (ii) for DOMINATING SET also works for CONNECTED DOMINATING SET, because, in the graphs produced by \mathcal{A} , $\text{OPT}_{\text{CON}}(G) = \text{OPT}_{\text{DS}}(G)$ and the solution output by \mathcal{B} is always a clique and, therefore, connected.
 - (b) Guha and Khuller [8] give a $\mathcal{O}(\log \Delta) \leq \mathcal{O}(\log n)$ -factor approximation for CONNECTED DOMINATING SET.
- (v) (a) The following reduction is essentially the same as the one given by Dom et al. [4] Let (U, \mathcal{S}) be an instance of HITTING SET. The algorithm \mathcal{A} creates the graph G as in the reduction for DOMINATING SET in (ii) and sets $T := \{w_s \mid s \in \mathcal{S}\} \cup \{u\}$. Clearly, $|V \setminus T| = |U|$ and it easy to show that $\text{OPT}_{\text{HS}}(U, \mathcal{S}) = \text{OPT}_{\text{NST}}(G, T)$. By a similar argument as in (ii), the algorithm \mathcal{B} can output $\{x \in U \mid v_x \in X\}$ where X is a given solution for the NODE STEINER TREE instance (G, T) .
 - (b) Klein and Ravi [17] give a $\mathcal{O}(\log n)$ -factor approximation for this problem. ◀

If \mathcal{C} is a hereditary class of graphs, then we may define the RESTRICTED \mathcal{C} -DELETION problem as follows: We are given a graph $G = (V, E)$ and $X \subseteq V$ such that $G - X \in \mathcal{C}$ and are asked to find a minimum $Y \subseteq X$ such that $G - Y \in \mathcal{C}$. For RESTRICTED PERFECT DELETION[$|X|$], RESTRICTED WEAKLY CHORDAL DELETION[$|X|$], and RESTRICTED WHEEL-FREE DELETION[$|X|$], reductions given by Heggernes et al. [10] and Lokshantov [19] can be shown to be 1-APPTs. However, it is open whether they have a $\mathcal{O}(2^{\log^c |I|})$ -factor approximation with $c < 1$, so we cannot rule out an approximate polynomial Turing kernelization. However, we can observe that, under ETH, they cannot have both an approximation algorithm with the aforementioned guarantee *and* an approximate polynomial Turing kernelization.

More generally, we can deduce from the proof of Lemma 2 the following about any parameterized minimization problem \mathcal{P} that only satisfies the first condition in Lemma 2: If we define an *approximate polynomial Turing compression* of a problem \mathcal{P} into a problem \mathcal{P}' to be essentially an approximate polynomial Turing kernelization for \mathcal{P} , except that it is given access to an approximate oracle for \mathcal{P}' rather than \mathcal{P} , then we can rule out (under ETH) an approximate polynomial Turing compression of any problem \mathcal{P} that satisfies the first condition into any problem \mathcal{P}' that satisfies the second condition in the same lemma.

3.2 MK[2]-hardness

The approach described in Section 3.1 is unlikely to work for the problem INDEPENDENT DOMINATING SET. That approach requires a $\mathcal{O}(2^{\log^c n})$ -factor approximation algorithm with $c < 1$ to answer the queries of the Turing kernelization. However, there is no $\mathcal{O}(n^{1-\varepsilon})$ -factor approximation for this problem for any $\varepsilon > 0$ unless $\text{P} = \text{NP}$ [9].

In the following, we will prove that there is no constant-factor approximate polynomial Turing kernelization for INDEPENDENT DOMINATING SET[vc], assuming a conjecture by Hermelin et al. [11] stating that parameterized decision problems that are hard for the complexity class MK[2] do not admit polynomial (exact) Turing kernelizations.

Let CNF-SAT denote the satisfiability problem for Boolean formulas in conjunctive normal form. The class MK[2] may be defined as the set of all parameterized problems that can be reduced with a PPT to CNF-SAT[n] where n denotes the number of variables.²

We will prove that an α -approximate polynomial Turing kernelization for INDEPENDENT DOMINATING SET[vc] implies the existence of a polynomial Turing kernelization for CNF-SAT[n]. For this, we will need the following lemma allowing us to translate queries between oracles for INDEPENDENT DOMINATING SET and CNF-SAT using a standard self-reduction:

► **Lemma 4.** *There is a polynomial-time algorithm that, given as input a graph G and access to an oracle that decides in a single step instances of CNF-SAT whose size is polynomially bounded in the size of G , outputs a minimum independent dominating set of G .*

Proof. The decision version of INDEPENDENT DOMINATING SET, in which one is given a graph H and an integer k and is asked to decide whether H contains an independent dominating set of size at most k , is in NP and CNF-SAT is NP-hard, so there is a polynomial-time many-one reduction from INDEPENDENT DOMINATING SET to CNF-SAT. For any graph H and integer k let $R(H, k)$ denote the instance of CNF-SAT obtained by applying this reduction to (H, k) .

Let n be the number of vertices in G . We first determine the size of a minimum independent dominating set in G by querying the oracle for CNF-SAT on the instance $R(G, k)$ for each $k \in [n]$. Observe that the size of $R(G, k)$ is polynomially bounded in the size of G . Hence, we may input this instance to the oracle. Let k_0 be the smallest value of k for which this query returns yes.

We must then construct an independent dominating set of size k_0 in G . We initially set $\ell := k_0$, $H := G$, and $S := \emptyset$ and perform the following operation as long as $\ell > 0$. For each vertex u in H , we query the oracle on the instance $R(H - N[u], \ell - 1)$. If this query returns yes, then we add u to S and set $H := H - N[u]$ and $\ell := \ell - 1$. Once $\ell = 0$, we output S .

We claim that this procedure returns an independent dominating set of size k_0 if k_0 is the size of a minimum independent dominating set in G . Let X be a minimum independent dominating set in G and $u \in X$. Then, $X \setminus \{u\}$ is a minimum dominating set of size $k_0 - 1$ in $G - N[u]$ and the claim follows inductively. ◀

► **Theorem 5.** *If, for any $\alpha \geq 1$, there is an α -approximate polynomial Turing kernelization for INDEPENDENT DOMINATING SET[vc], then there is a polynomial Turing kernelization for CNF-SAT[n].*

Proof. Assume that there is an α -approximate Turing kernelization for INDEPENDENT DOMINATING SET[vc] whose size is bounded by the polynomial p . We will give a polynomial Turing kernelization for CNF-SAT[n].

Let the input be a formula F in conjunctive normal form over the variables x_1, \dots, x_n consisting of the clauses C_1, \dots, C_m . First, we compute a graph G on which we then run the approximate Turing kernelization for INDEPENDENT DOMINATING SET. The construction of the graph G in the following is due to Irving [14].

² This is not directly the definition given by Hermelin et al. [11], but an equivalent characterization.

Let $s := \lceil \alpha \cdot n \rceil + 1$. The graph $G = (V, E)$ contains vertices v_1, \dots, v_n and $\bar{v}_1, \dots, \bar{v}_n$, representing the literals that may occur in F . Additionally, for each $j \in [m]$, there are s vertices w_j^1, \dots, w_j^s representing the clause C_j . For each $i \in [n]$, there is an edge between v_i and \bar{v}_i . There is also an edge between v_i and w_j^ℓ for all $\ell \in [s]$ if $x_i \in C_j$ and an edge between \bar{v}_i and w_j^ℓ for all $\ell \in [s]$ if $\neg x_i \in C_j$. The intuition behind this construction is as follows: In G any independent dominating set may contain at most one of v_i and \bar{v}_i for each $i \in [n]$, so any such set represents a partial truth assignment of the variables x_1, \dots, x_n . If F is satisfiable, then G contains an independent dominating set of size n . Conversely, if F is not satisfiable, then any independent dominating set must contain w_j^1, \dots, w_j^s for some $j \in [m]$, so it must have size at least $s > \alpha \cdot n$, thus creating a gap of size greater than α between yes and no instances. Moreover, $\{v_i, \bar{v}_i \mid i \in [n]\}$ is a vertex cover in G , so the vertex cover number of G is polynomially bounded in n .

The Turing kernelization for CNF-SAT proceeds in the following manner. Given the formula F , it first computes the graph G and runs the α -approximate Turing kernelization for INDEPENDENT DOMINATING SET on G . Whenever the α -approximate Turing kernelization queries the oracle on an instance G' , this query is answered using the algorithm given by Lemma 4. Observe that the size of G' is polynomially bounded in the vertex cover number of G , which in turn is polynomially bounded in n , so the oracle queries made by this algorithm are possible. Let X be the independent dominating set for G output by the approximate Turing kernelization. Since this Turing kernelization is given access to a 1-approximate oracle, $|X| \leq \alpha \cdot \text{OPT}_{\text{IND}}(G)$. We claim that F is satisfiable if and only if $|X| \leq \alpha \cdot n$. With this claim, the Turing kernelization can return yes if and only if this condition is met.

If F is satisfiable and $\varphi: \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$ is a satisfying truth assignment, then $Y := \{v_i \mid \varphi(x_i) = 1\} \cup \{\bar{v}_i \mid \varphi(x_i) = 0\}$ is an independent dominating set in G . Hence, $|X| \leq \alpha \cdot \text{OPT}_{\text{IND}}(G) \leq \alpha \cdot |Y| = \alpha \cdot n$. Conversely, suppose that F is not satisfiable. For each $i \in [n]$, the set X may contain at most one of the vertices v_i and \bar{v}_i . Consider the partial truth assignment with $\varphi(x_i) := 1$ if $v_i \in X$ and $\varphi(x_i) := 0$ if $\bar{v}_i \in X$. Because F is unsatisfiable there is a clause C_j that is not satisfied by φ . Hence, X must contain the vertices w_j^1, \dots, w_j^s . Therefore, $|X| \geq s > \alpha \cdot n$. \blacktriangleleft

4 Turing kernelizations for parameter $\text{tw} + \Delta$

In this section, we will prove that the domination problems for which we proved lower bounds when parameterized by the vertex cover number do have $(1 + \varepsilon)$ -approximate polynomial Turing kernels when parameterized by treewidth plus maximum degree. The following lemma is a generalization of [12, Lemma 11] and can be proved in the same way.

► **Lemma 6.** *Let G be a graph with n vertices, \mathcal{T} be a nice tree decomposition of G , and $s \leq n$. Then, there is a node t of \mathcal{T} such that $s \leq |V_t| \leq 2s$. Moreover, such a node t can be found in polynomial time.*

4.1 Dominating Set

We start with DOMINATING SET.

► **Lemma 7.** *Let $G = (V, E)$ be a graph.*

- (i) *If Δ is the maximum degree of G , then $\text{OPT}_{\text{DS}}(G) \geq \frac{|V|}{\Delta+1}$.*
- (ii) *If $A, B, C \subseteq V$ with $A \cup C = V$, $A \cap C = B$, and there are no edges between $A \setminus B$ and $C \setminus B$, then $\text{OPT}_{\text{DS}}(G) \geq \text{OPT}_{\text{DS}}(G[A]) + \text{OPT}_{\text{DS}}(G[C]) - 2|B|$.*

■ **Algorithm 1** A $(1 + \varepsilon)$ -approximate polynomial Turing kernelization for DOMINATING SET parameterized by $\text{tw} + \Delta$.

```

input : A graph  $G = (V, E)$ , nice tree decomposition  $\mathcal{T}$  of width  $\text{tw}$ ,  $\varepsilon > 0$ 
1  $s \leftarrow 2 \cdot \frac{1+\varepsilon}{\varepsilon} \cdot (\text{tw} + 1) \cdot (\Delta + 1)$ 
2 if  $|V| \leq s$  then
3   | Apply the  $c$ -approximate oracle to  $G$  and output the result.
4 else
5   | Use Lemma 6 to find a node  $t$  in  $\mathcal{T}$  such that  $s \leq |V_t| \leq 2s$ .
6   | Apply the  $c$ -approximate oracle to  $G[V_t]$  and let  $S_t$  be the solution output by the
   | oracle.
7   | Let  $\mathcal{T}'$  be the tree obtained by deleting the subtree rooted at  $t$  except for the
   | node  $t$  from  $\mathcal{T}$ .
8   | Apply this algorithm to  $(G - (V_t \setminus X_t), \mathcal{T}', \varepsilon)$  and let  $S'$  be the returned solution.
9   | Return  $S_t \cup S'$ .
10 end

```

Proof.

- (i) Every vertex can only dominate its at most Δ neighbors and itself.
- (ii) Let X be a dominating set in G of size $\text{OPT}_{\text{DS}}(G)$. Then $Y := (X \cap A) \cup B$ and $Z := (X \cap C) \cup B$ are dominating sets in $G[A]$ and $G[C]$, respectively. Hence,

$$\begin{aligned}
 \text{OPT}_{\text{DS}}(G) &= |X| = |X \cap A| + |X \cap C| - |X \cap B| \\
 &\geq |Y| - |B \setminus X| + |Z| - |B| \\
 &\geq \text{OPT}_{\text{DS}}(G[A]) + \text{OPT}_{\text{DS}}(G[C]) - 2|B|. \quad \blacktriangleleft
 \end{aligned}$$

► **Theorem 8.** For every $\varepsilon > 0$, there is a $(1 + \varepsilon)$ -approximate Turing kernelization for DOMINATING SET with $\mathcal{O}(\frac{1+\varepsilon}{\varepsilon} \cdot \text{tw} \cdot \Delta)$ vertices.

Proof. Consider Algorithm 1. This algorithm always returns a dominating set of G . If the algorithm terminates in line 3, then this is true because the oracle always outputs a dominating set. If it terminates in line 9, then let $v \in V$ be an arbitrary vertex. If $v \in V_t$, then v is dominated by a vertex in S_t , because S_t is a dominating set in $G[V_t]$. If $v \in V \setminus V_t$, then v is dominated by a vertex in S' .

The algorithm runs in polynomial time.

Finally, we must show that the solution output by the algorithm contains at most $c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{DS}}(G)$ vertices. We prove the claim by induction on the number of recursive calls. If there is no recursive call, the algorithm terminates in line 3 and the solution contains at most $c \cdot \text{OPT}_{\text{DS}}(G)$ vertices. Otherwise, by induction:

$$\begin{aligned}
 |S_t \cup S'| &\leq |S_t| + |S'| \leq c \cdot \text{OPT}_{\text{DS}}(G[V_t]) + |S'| \\
 &= c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{DS}}(G[V_t]) - c \cdot \varepsilon \cdot \text{OPT}_{\text{DS}}(G[V_t]) + |S'| \\
 &\stackrel{\dagger}{\leq} c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{DS}}(G[V_t]) - \frac{c \cdot \varepsilon \cdot |V_t|}{\Delta + 1} + |S'| \\
 &\leq c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{DS}}(G[V_t]) - 2 \cdot c \cdot (1 + \varepsilon) \cdot (\text{tw} + 1) + |S'| \\
 &\leq c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{DS}}(G[V_t]) - 2 \cdot c \cdot (1 + \varepsilon) \cdot |X_t| + |S'|
 \end{aligned}$$

$$\begin{aligned}
&\leq c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{DS}}(G[V_t]) - 2 \cdot c \cdot (1 + \varepsilon) \cdot |X_t| + c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{DS}}(G - V_t) \\
&= c \cdot (1 + \varepsilon) \cdot (\text{OPT}_{\text{DS}}(G[V_t]) - 2|X_t| + \text{OPT}_{\text{DS}}(G - V_t)) \\
&\stackrel{\dagger}{\leq} c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{DS}}(G).
\end{aligned}$$

Here, the inequality marked \dagger follows from Lemma 7(i) and the one marked \ddagger follows from Lemma 7(ii) with $A = (V \setminus V_t) \cup X_t$, $B = X_t$, and $C = V_t \setminus X_t$. \blacktriangleleft

4.2 Capacitated Dominating Set

Next, we consider CAPACITATED DOMINATING SET.

► **Lemma 9.** *Let $G = (V, E, \text{cap})$ be a capacitated graph with maximum degree Δ and $A, B, C \subseteq V$ such that $A \cup C = V$, $A \cap C = B$, and there are no edges from $A \setminus B$ to $C \setminus B$.*

- (i) $\text{OPT}_{\text{CAP}}(G) \geq \text{OPT}_{\text{CAP}}(G[A]) + \text{OPT}_{\text{CAP}}(G[C]) - 2|B|$.
- (ii) *Given capacitated dominating sets (X, f) and (Y, g) in $G[A]$ and $G[C]$, respectively, one can in construct in polynomial time a capacitated dominating set for G of size at most $|X| + |Y| + (\Delta + 1) \cdot |B|$.*

Proof.

- (i) Let (X, f) with $X \subseteq V$ and $f: V \setminus X \rightarrow X$ be a capacitated dominating set of size $\text{OPT}_{\text{CAP}}(G)$. Then, (Y, g) with $Y := (X \cap A) \cup B$ and $g(v) := f(v)$ for all $v \in A \setminus Y$ is a capacitated dominating set in $G[A]$ and (Z, h) with $Z := (X \cap C) \cup B$ and $h(v) := f(v)$ for all $v \in C \setminus Z$ is a capacitated dominating set in $G[C]$. Hence,

$$\begin{aligned}
\text{OPT}_{\text{CAP}}(G) &= |X| = |X \cap A| + |X \cap C| - |X \cap B| \\
&= |Y| - |B \setminus X| + |Z| - |B| - |X \cap B| \\
&\geq |Y| + |Z| - 2|B| \\
&\geq \text{OPT}_{\text{CAP}}(G[A]) + \text{OPT}_{\text{CAP}}(G[C]) - 2|B|.
\end{aligned}$$

- (ii) We construct the capacitated dominating set (Z, h) for G as follows. Let $Z := X \cup Y \cup N[B]$. Observe that $|N[B]| \leq (\Delta + 1)|B|$. Define h by setting $h(v) := f(v)$ for all $v \in A \setminus Z$ and $h(v) := g(v)$ for all $v \in C \setminus Z$. One can easily verify that this is a capacitated dominating set. \blacktriangleleft

► **Theorem 10.** *For every $\varepsilon > 0$, there is a $(1 + \varepsilon)$ -approximate Turing kernelization for CAPACITATED DOMINATING SET with $\mathcal{O}(\frac{1+\varepsilon}{\varepsilon} \cdot \text{tw} \cdot \Delta^2)$ vertices.*

Proof. Consider Algorithm 2. This algorithm always returns a capacitated dominating set of G . If the algorithm terminates in line 3, then this is true because the oracle always outputs a capacitated dominating set. If it terminates in line 10, then (S_t, f_t) and (S', f') are capacitated dominating sets for $G[V_t]$ and $G - (V_t \setminus X_t)$, respectively. It follows by Lemma 9(ii), that (S, f) is a capacitated dominating set for G .

The algorithm runs in polynomial time.

Finally, we must show that the solution output by the algorithm contains at most $c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CAP}}(G)$ vertices. We prove the claim by induction on the number of recursive calls. If there is no recursive call, the algorithm terminates in line 3 and the solution contains at most $c \cdot \text{OPT}_{\text{CAP}}(G)$ vertices. Otherwise, by induction:

■ **Algorithm 2** A $(1 + \varepsilon)$ -approximate polynomial Turing kernelization for CAPACITATED DOMINATING SET parameterized by $\text{tw} + \Delta$.

```

input : A graph  $G = (V, E)$ , nice tree decomposition  $\mathcal{T}$  of width  $\text{tw}$ ,  $\varepsilon > 0$ 
1  $s \leftarrow 3 \cdot \frac{1+\varepsilon}{\varepsilon} \cdot (\text{tw} + 1) \cdot (\Delta + 1)^2$ 
2 if  $|V| \leq s$  then
3   | Apply the  $c$ -approximate oracle to  $G$  and output the result.
4 else
5   | Use Lemma 6 to find a node  $t$  in  $\mathcal{T}$  such that  $s \leq |V_t| \leq 2s$ .
6   | Apply the  $c$ -approximate oracle to  $G[V_t]$  and let  $(S_t, f_t)$  be the solution output by
   | the oracle.
7   | Let  $\mathcal{T}'$  be the tree obtained by deleting the subtree rooted at  $t$  except for the
   | node  $t$  from  $\mathcal{T}$ .
8   | Apply this algorithm to  $(G - (V_t \setminus X_t), \mathcal{T}', \varepsilon)$  and let  $(S', f')$  be the returned
   | solution.
9   | Apply Lemma 9(ii) with  $(X, f) = (S', f')$ ,  $(Y, g) = (S_t, f_t)$ ,  $A = (V \setminus V_t) \cup X_t$ ,
   |  $B = X_t$ , and  $C = V_t$ . Let  $(S, f)$  be the resulting solution for  $G$ .
10  | Return  $(S, f)$ .
11 end

```

$$\begin{aligned}
|S| &\leq |S'_t| + |S'| + (\Delta + 1) \cdot |X_t| \leq c \cdot \text{OPT}_{\text{CAP}}(G[V_t]) + |S'| + (\Delta + 1) \cdot |X_t| \\
&= c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CAP}}(G[V_t]) - c \cdot \varepsilon \cdot \text{OPT}_{\text{CAP}}(G[V_t]) + |S'| + (\Delta + 1) \cdot |X_t| \\
&\stackrel{\dagger}{\leq} c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CAP}}(G[V_t]) - \frac{c \cdot \varepsilon \cdot |V_t|}{\Delta + 1} + |S'| + (\Delta + 1) \cdot |X_t| \\
&\leq c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CAP}}(G[V_t]) - 3 \cdot c \cdot (1 + \varepsilon) \cdot (\text{tw} + 1) \cdot (\Delta + 1) + |S'| + (\Delta + 1) \cdot |X_t| \\
&\stackrel{\ddagger}{\leq} c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CAP}}(G[V_t]) - 3 \cdot c \cdot (1 + \varepsilon) \cdot |X_t| \cdot (\Delta + 1) + |S'| \\
&\quad + c \cdot (1 + \varepsilon) \cdot (\Delta + 1) \cdot |X_t| \\
&\leq c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CAP}}(G[V_t]) - 2 \cdot c \cdot (1 + \varepsilon) \cdot |X_t| \cdot (\Delta + 1) + |S'| \\
&\leq c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CAP}}(G[V_t]) - 2 \cdot c \cdot (1 + \varepsilon) \cdot |X_t| + c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CAP}}(G - V_t) \\
&= c \cdot (1 + \varepsilon) \cdot (\text{OPT}_{\text{CAP}}(G[V_t]) - 2|X_t| + \text{OPT}_{\text{CAP}}(G - V_t)) \\
&\stackrel{\blacklozenge}{\leq} c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CAP}}(G)
\end{aligned}$$

The inequality marked \dagger follows from Lemma 7(i) and the fact that $\text{OPT}_{\text{CAP}}(G) \geq \text{OPT}_{\text{DS}}(G)$.
 \ddagger follows from the fact that $c \cdot (1 + \varepsilon) \geq 1$ and \blacklozenge from Lemma 9(i). ◀

4.3 Independent Dominating Set

The next problem we consider is INDEPENDENT DOMINATING SET.

► **Lemma 11.** *Let $G = (V, E)$ be a graph with maximum degree Δ and $A, B, C \subseteq V$ such that $A \cup C = V$, $A \cap C = B$, and there are no edges from $A \setminus B$ to $C \setminus B$.*

- (i) *If X is an independent set in G , then there is an independent dominating set X' that contains X and $|X' \setminus X|$ is at most the number of vertices not dominated by X , and such a set X' can be computed in polynomial time.*
- (ii) $\text{OPT}_{\text{IND}}(G) \geq \text{OPT}_{\text{IND}}(G[A]) + \text{OPT}_{\text{IND}}(G[C]) - 2|B|$.
- (iii) *Given independent dominating sets X and Y in $G[A]$ and $G[C]$, respectively, one can in construct in polynomial time an independent dominating set for G of size at most $|X| + |Y| + (\Delta + 1) \cdot |B|$.*

■ **Algorithm 3** A $(1 + \varepsilon)$ -approximate polynomial Turing kernelization for INDEPENDENT DOMINATING SET parameterized by $\text{tw} + \Delta$.

```

input : A graph  $G = (V, E)$ , nice tree decomposition  $\mathcal{T}$  of width  $\text{tw}$ ,  $\varepsilon > 0$ 
1  $s \leftarrow |V| \leq 3 \cdot \frac{1+\varepsilon}{\varepsilon} \cdot (\text{tw} + 1) \cdot (\Delta + 1)^2$ 
2 if  $|V| \leq s$  then
3   | Apply the  $c$ -approximate oracle to  $G$  and output the result.
4 else
5   | Use Lemma 6 to find a node  $t$  in  $\mathcal{T}$  such that  $s \leq |V_t| \leq 2s$ .
6   | Apply the  $c$ -approximate oracle to  $G[V_t]$  and let  $S_t$  be the solution output by the
   | oracle.
7   | Let  $\mathcal{T}'$  be the tree obtained by deleting the subtree rooted at  $t$  except for the
   | node  $t$  from  $\mathcal{T}$ .
8   | Apply this algorithm to  $(G - (V_t \setminus X_t), \mathcal{T}', \varepsilon)$  and let  $S'$  be the returned solution.
9   | Apply Lemma 11(iii) with  $X = S'$ ,  $Y = S_t$ ,  $A = (V \setminus V_t) \cup X_t$ ,  $B = X_t$ , and
   |  $C = V_t$ . Let  $S$  be the resulting solution for  $G$ .
10  | Return  $S$ .
11 end

```

Proof.

- (i) If X is a dominating set, then $X' := X$. Otherwise, there is a vertex $v \in V \setminus N[X]$. We add v to X and continue. Observe that when v is added to X , the latter remains an independent set.
- (ii) Let X be an independent dominating set of size $\text{OPT}_{\text{IND}}(G)$ in G . Let $Y := X \cap A$. Since $Y \subseteq X$, it follows that Y is an independent set. Moreover, Y dominates all vertices in $(A \setminus B) \cup (X \cap B)$, leaving at most $B \setminus X$ vertices undominated. We apply (i) to Y and obtain Y' , an independent dominating set in $G[A]$ of size at most $|X \cap A| + |B| - |X \cap B|$. We apply the same argument to $G[C]$ to obtain an independent dominating set Z of size at most $|X \cap C| + |B| - |X \cap B|$. It follows that:

$$\begin{aligned}
\text{OPT}_{\text{IND}}(G) &= |X| = |X \cap A| + |X \cap C| - |X \cap B| \\
&= |Y| - |B \setminus X| + |Z| - |B| - |X \cap B| \\
&\geq |Y| + |Z| - 2|B| \\
&\geq \text{OPT}_{\text{IND}}(G[A]) + \text{OPT}_{\text{IND}}(G[C]) - 2|B|.
\end{aligned}$$

- (iii) $Z := (X \cup Y) \setminus B$ is an independent set in G . Since $X \cup Y$ is a dominating set and at most $(\Delta + 1) \cdot |B|$ vertices can be dominated by vertices in B , it follows that Z leaves at most that many vertices in G undominated. Applying (i) to Z yields an independent dominating set of size at most $|X| + |Y| + (\Delta + 1) \cdot |B|$. ◀

► **Theorem 12.** For every $\varepsilon > 0$, there is a $(1 + \varepsilon)$ -approximate Turing kernelization for INDEPENDENT DOMINATING SET with $\mathcal{O}(\frac{1+\varepsilon}{\varepsilon} \cdot \text{tw} \cdot \Delta^2)$ vertices.

Proof. Consider Algorithm 3. This algorithm always returns an independent dominating set of G . If the algorithm terminates in line 3, then this is true because the oracle always outputs an independent dominating set. If it terminates in line 10, then S_t and S' are independent dominating sets for $G[V_t]$ and $G - (V_t \setminus X_t)$, respectively. It follows by Lemma 11(iii), that S is an independent dominating set for G .

The algorithm runs in polynomial time.

Finally, we must show that the solution output by the algorithm contains at most $c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{IND}}(G)$ vertices. We prove the claim by induction on the number of recursive calls. If there is no recursive call, the algorithm terminates in line 3 and the solution contains at most $c \cdot \text{OPT}_{\text{IND}}(G)$ vertices. Otherwise, by induction:

$$\begin{aligned}
|S| &\leq |S'_t| + |S'| + (\Delta + 1) \cdot |X_t| \leq c \cdot \text{OPT}_{\text{IND}}(G[V_t]) + |S'| + (\Delta + 1) \cdot |X_t| \\
&= c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{IND}}(G[V_t]) - c \cdot \varepsilon \cdot \text{OPT}_{\text{IND}}(G[V_t]) + |S'| + (\Delta + 1) \cdot |X_t| \\
&\stackrel{\dagger}{\leq} c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{IND}}(G[V_t]) - \frac{c \cdot \varepsilon \cdot |V_t|}{\Delta + 1} + |S'| + (\Delta + 1) \cdot |X_t| \\
&\leq c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{IND}}(G[V_t]) - 3 \cdot c \cdot (1 + \varepsilon) \cdot (\text{tw} + 1) \cdot (\Delta + 1) + |S'| + (\Delta + 1) \cdot |X_t| \\
&\stackrel{\ddagger}{\leq} c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{IND}}(G[V_t]) - 3 \cdot c \cdot (1 + \varepsilon) \cdot |X_t| \cdot (\Delta + 1) \\
&\quad + |S'| + c \cdot (1 + \varepsilon) \cdot (\Delta + 1) \cdot |X_t| \\
&\leq c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{IND}}(G[V_t]) - 2 \cdot c \cdot (1 + \varepsilon) \cdot |X_t| \cdot (\Delta + 1) + |S'| \\
&\leq c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{IND}}(G[V_t]) - 2 \cdot c \cdot (1 + \varepsilon) \cdot |X_t| + c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{IND}}(G - V_t) \\
&= c \cdot (1 + \varepsilon) \cdot (\text{OPT}_{\text{IND}}(G[V_t]) - 2|X_t| + \text{OPT}_{\text{IND}}(G - V_t)) \\
&\stackrel{\blacklozenge}{\leq} c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{IND}}(G)
\end{aligned}$$

The inequality marked with \dagger follows from Lemma 7(i) and the fact that $\text{OPT}_{\text{IND}}(G) \geq \text{OPT}_{\text{DS}}(G)$. \ddagger follows from the fact that $c \cdot (1 + \varepsilon) \geq 1$ and \blacklozenge from Lemma 11(ii). \blacktriangleleft

4.4 Connected Dominating Set

Finally, we consider the problem CONNECTED DOMINATING SET. If $S \subseteq V$ is a vertex set in a graph $G = (V, E)$, then let $R(G, S)$ denote the graph obtained by deleting S , introducing a new vertex z , and connecting z to any vertex in $V \setminus S$ that has a neighbor in S .

► **Lemma 13.** *Let $G = (V, E)$ be a connected graph and $A, B, C \subseteq V$ such that $A \cup C = V$, $A \cap C = B$, there are no edges from $A \setminus B$ to $C \setminus B$, and $A \setminus B$ and $C \setminus B$ are both non-empty.*

- (i) $\text{OPT}_{\text{CON}}(G) \geq \text{OPT}_{\text{CON}}(R(G[A], B)) + \text{OPT}_{\text{CON}}(R(G[C], B)) - 2$.
- (ii) *Given connected dominating sets X and Y in $R(G[A], B)$ and $R(G[C], B)$, respectively, one can in construct in polynomial time a connected dominating set for G of size at most $|X| + |Y| + 3|B|$.*

Proof.

- (i) Let X be a connected dominating set in G of size $\text{OPT}_{\text{CON}}(G)$. We claim that $Y := (X \cap (A \setminus B)) \cup \{z\}$ and $Z := (X \cap (C \setminus B)) \cup \{z\}$ are connected dominating sets in $R(G[A], B)$ and $R(G[C], B)$, respectively. We only prove this for Y and $R(G[A], B)$, as the case of Z and $R(G[C], B)$ is analogous.

First, we show that Y is a dominating set. Let v be a vertex in $R(G[A], B)$. If $v \in \{z, z'\}$, then v is dominated by z in Y . Otherwise, $v \in A \setminus B$ and there is a vertex $w \in X$ that dominates v in G . If $w \in B$, then z is adjacent to v in $R(G[A], B)$ and v is dominated by z in that graph. If $w \in A \setminus B$, then $w \in Y$ and v is dominated by w in $R(G[A], B)$.

We must also show that the subgraph of $R(G[A], B)$ induced by Y is connected. Let $v, v' \in Y$. We must show that the subgraph of $R(G[A], B)$ induced by Y contains a path from v to v' . First we assume that $v, v' \neq z, z'$, implying that $v, v' \in X$. Hence, there is a path P from v to v' in $G[X]$. If $P \subseteq A \setminus B$, then $P \subseteq Y$ and we are done. Otherwise, P must pass through B . Let w be the first vertex in B on P and w' the

final one. Obtain P' by replacing the subpath of P between and including w and w' with z . Then, P' is a path from v to v' in the subgraph of $R(G[A], B)$ induced by Y . Finally, suppose that $v' = z$. If $v = z$, there is nothing to show, so we assume that $v \neq z$ and, therefore, $v \in X \cap (A \setminus B)$. Because $C \setminus B$ is non-empty, X must contain a vertex $w \in B$. Because $G[X]$ is connected, $G[X]$ must also contain a path P from v to w . Let w' be the first vertex in B on the path P (possibly, $w' = w$). We obtain P' , a path from v to z in the subgraph of $R(G[A], B)$ induced by Y , by taking the subpath of P from v to w' and replacing w' with z . This proves that Y is a connected dominating set in $R(G[A], B)$. Then,

$$\begin{aligned} \text{OPT}_{\text{CON}}(G) &= |X| \geq |X \cap (A \setminus B)| + |X \cap (C \setminus B)| \\ &\geq |Y| - 1 + |Z| - 1 \\ &\geq \text{OPT}_{\text{CON}}(R(G[A], B)) + \text{OPT}_{\text{CON}}(R(G[C], B)) - 2. \end{aligned}$$

- (ii) Let $Z' := X \cup Y \cup B$. Every connected component of $G[Z']$ contains a vertex in B , so this graph has at most $|B|$ connected components. We obtain a connected dominating set Z in G as follows. We start with $Z := Z'$. Choose two connected components C_1, C_2 in $G[Z]$. Because G is connected, it contains a path P starting in $v_1 \in C_1$ and ending in $v_2 \in C_2$. This path must contain a vertex that is not adjacent to any vertex in C_1 , because if every vertex in $P \setminus C_1$ were adjacent to a vertex in C_1 , then v_2 is adjacent to a vertex in C_1 , implying that C_1 and C_2 are not distinct connected components in $G[Z]$. Let w be the first vertex on P that is not adjacent to a vertex in C_1 . Because Z is a dominating set in G , there must be a vertex $x \in Z \setminus C_1$ such that $w \in N[x]$ (note that, possibly $w = x$). Adding w and x merges C_1 with the connected component of $G[Z]$ containing x . This process must be repeated at most $|B|$ times to obtain a connected dominating set. In each iteration at most two vertices are added to Z . Since Z initially contains $|X| + |Z| + |B|$ vertices, we obtain a connected dominating set containing at most $|X| + |Z| + 3|B|$ vertices. \blacktriangleleft

► **Theorem 14.** *For every $\varepsilon > 0$, there is a $(1 + \varepsilon)$ -approximate Turing kernelization for CONNECTED DOMINATING SET with $\mathcal{O}(\frac{1+\varepsilon}{\varepsilon} \cdot \text{tw} \cdot \Delta)$ vertices.*

Proof. Consider Algorithm 4. This algorithm always returns a connected dominating set of G . If the algorithm terminates in line 3, then this is true because the oracle always outputs a connected dominating set. If it terminates in line 10, then S_t and S' are connected dominating sets for $R(G[V_t], X_t)$ and $R(G - (V_t \setminus X_t))$, respectively. It follows by Lemma 13(ii), that S is a connected dominating set for G .

The algorithm runs in polynomial time.

Finally, we must show that the solution output by the algorithm contains at most $c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CON}}(G)$ vertices. We prove the claim by induction on the number of recursive calls. If there is no recursive call, the algorithm terminates in line 3 and the solution contains at most $c \cdot \text{OPT}_{\text{CAP}}(G)$ vertices. Otherwise, by induction:

$$\begin{aligned} |S| &\leq |S'_t| + |S'| + 3|X_t| \leq c \cdot \text{OPT}_{\text{CON}}(R(G[V_t], X_t)) + |S'| + 3|X_t| \\ &= c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CON}}(R(G[V_t], X_t)) - c \cdot \varepsilon \cdot \text{OPT}_{\text{CON}}(R(G[V_t], X_t)) + |S'| + 3|X_t| \\ &\stackrel{\dagger}{\leq} c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CON}}(R(G[V_t], X_t)) - \frac{c \cdot \varepsilon \cdot (|V_t| - |X_t| + 1)}{\Delta + 1} + |S'| + 3|X_t| \\ &= c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CON}}(R(G[V_t], X_t)) - \frac{c \cdot \varepsilon \cdot |V_t|}{\Delta + 1} + |S'| + \frac{c \cdot \varepsilon \cdot (|X_t| + 1)}{\Delta + 1} + 3|X_t| \end{aligned}$$

■ **Algorithm 4** A $(1 + \varepsilon)$ -approximate polynomial Turing kernelization for CONNECTED DOMINATING SET parameterized by $\text{tw} + \Delta$.

```

input : A graph  $G = (V, E)$ , nice tree decomposition  $\mathcal{T}$  of width  $\text{tw}$ ,  $\varepsilon > 0$ 
1  $s \leftarrow 4 \cdot \frac{1+\varepsilon}{\varepsilon}(\Delta + 1)(\text{tw} + 1) + \frac{(2\Delta+2)(1+\varepsilon)}{\varepsilon}$ 
2 if  $|V| \leq s$  then
3   | Apply the  $c$ -approximate oracle to  $G$  and output the result.
4 else
5   | Use Lemma 6 to find a node  $t$  in  $\mathcal{T}$  such that  $s \leq |V_t| \leq 2s$ .
6   | Apply the  $c$ -approximate oracle to  $R(G[V_t], X_t)$  and let  $S_t$  be the solution output
   | by the oracle..
7   | Let  $\mathcal{T}'$  be the tree obtained by deleting the subtree rooted at  $t$  except for the
   | node  $t$  from  $\mathcal{T}$ .
8   | Apply this algorithm to  $(R(G - (V_t \setminus X_t), X_t), \mathcal{T}', \varepsilon)$  and let  $S'$  be the returned
   | solution.
9   | Apply Lemma 13(ii) with  $X = S'$ ,  $Y = S_t$ ,  $A = (V \setminus V_t) \cup X_t$ ,  $B = X_t$ , and
   |  $C = V_t$ . Let  $S$  be the resulting solution for  $G$ .
10  | Return  $S$ .
11 end

```

$$\begin{aligned}
&\leq c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CON}}(G[V_t]) - 4 \cdot c \cdot (1 + \varepsilon) \cdot (\text{tw} + 2) - 2c(1 + \varepsilon) + |S'| \\
&\quad + \frac{c \cdot \varepsilon \cdot (|X_t| + 1)}{\Delta + 1} + 3|X_t| \\
&\stackrel{\ddagger}{\leq} c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CON}}(G[V_t]) - 4 \cdot c \cdot (1 + \varepsilon) \cdot (|X_t| + 1) - 2c(1 + \varepsilon) + |S'| \\
&\quad + c \cdot (1 + \varepsilon) \cdot (4|X_t| + 1) \\
&\leq c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CON}}(G[V_t]) - 2 \cdot c(1 + \varepsilon) + |S'| \\
&\leq c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CON}}(G[V_t]) - 2c(1 + \varepsilon) + c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CON}}(G - V_t) \\
&= c \cdot (1 + \varepsilon) \cdot (\text{OPT}_{\text{CON}}(G[V_t]) - 2 + \text{OPT}_{\text{CON}}(G - V_t)) \\
&\stackrel{\clubsuit}{\leq} c \cdot (1 + \varepsilon) \cdot \text{OPT}_{\text{CON}}(G)
\end{aligned}$$

The inequality marked with \ddagger follows from Lemma 7(ii) and the fact that $\text{OPT}_{\text{CON}}(G) \geq \text{OPT}_{\text{DS}}(G)$. \clubsuit follows from the fact that $c \cdot (1 + \varepsilon) \geq 1$ and \spadesuit from Lemma 13(i). ◀

5 Conclusion

We conclude by pointing out two open problems concerning approximate Turing kernelization:

- Does CONNECTED FEEDBACK VERTEX SET parameterized by treewidth admit an approximate polynomial Turing kernelization? The approach employed by Hols et al. [12] for CONNECTED VERTEX COVER and here for CONNECTED DOMINATING SET cannot be used for CONNECTED FEEDBACK VERTEX SET, because the ratio between the size of a minimum connected feedback vertex and the size of a minimum feedback vertex set is unbounded.
- The biggest open question in Turing kernelization is whether or not there are polynomial Turing kernelizations for the problems LONGEST PATH and LONGEST CYCLE parameterized by the solution size [11]. There has been some progress on this problem by considering the restriction to certain graph classes [15, 16]. Developing an *approximate* Turing kernelization may be another way of achieving progress in this regard.

References

- 1 Daniel Binkele-Raible, Henning Fernau, Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. *ACM Transactions on Algorithms*, 8(4), 2012. doi:10.1145/2344422.2344428.
- 2 M. Chlebík and J. Chlebíková. Approximation hardness of dominating set problems in bounded degree graphs. *Information and Computation*, 206(11):1264–1275, 2008. doi:10.1016/j.ic.2008.07.003.
- 3 Vasek Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979. doi:10.1287/moor.4.3.233.
- 4 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and IDs. *ACM Transactions on Algorithms*, 11(2):1–20, 2014. doi:10.1145/2650261.
- 5 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629–657, 2008. doi:10.1137/05064299X.
- 6 Michael R. Fellows, Ariel Kulik, Frances Rosamond, and Hadas Shachnai. Parameterized approximation via fidelity preserving transformations. *Journal of Computer and System Sciences*, 93:30–40, 2018. doi:10.1016/j.jcss.2017.11.001.
- 7 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.
- 8 Sudipto Guha and Samir Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20:374–387, 1998. doi:10.1007/PL00009201.
- 9 Magnús M. Halldórsson. Approximating the minimum maximal independence number. *Information Processing Letters*, 46(4):169–172, 1993. doi:10.1016/0020-0190(93)90022-2.
- 10 Pinar Heggenes, Pim van 't Hof, Bart M.P. Jansen, Stefan Kratsch, and Yngve Villanger. Parameterized complexity of vertex deletion into perfect graph classes. *Theoretical Computer Science*, 511:172–180, 2013. doi:10.1016/j.tcs.2012.03.013.
- 11 Danny Hermelin, Stefan Kratsch, Karolina Sołtys, Magnus Wahlström, and Xi Wu. A completeness theory for polynomial (Turing) kernelization. *Algorithmica*, 71(3):702–730, 2015. doi:10.1007/s00453-014-9910-8.
- 12 Eva-Maria C. Hols, Stefan Kratsch, and Astrid Pieterse. Approximate Turing kernelization for problems parameterized by treewidth. In *Proceedings of the 28th Annual European Symposium on Algorithms (ESA)*, pages 60:1–60:23, 2020. doi:10.4230/LIPIcs.ESA.2020.60.
- 13 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k -SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
- 14 Robert W. Irving. On approximating the minimum independent dominating set. *Information Processing Letters*, 37(4):197–200, 1991. doi:10.1016/0020-0190(91)90188-N.
- 15 Bart M. P. Jansen. Turing kernelization for finding long paths and cycles in restricted graph classes. *Journal of Computer and System Sciences*, 85:18–37, 2017. doi:10.1016/j.jcss.2016.10.008.
- 16 Bart M. P. Jansen, Marcin Pilipczuk, and Marcin Wrochna. Turing kernelization for finding long paths in graphs excluding a topological minor. In *Proceedings of the 12th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 23:1–23:13, 2018. doi:10.4230/LIPIcs.IPEC.2017.23.
- 17 Philip Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted Steiner trees. *Journal of Algorithms*, 19(1):104–115, 1995. doi:10.1006/jagm.1995.1029.
- 18 Ton Kloks. *Treewidth: Computations and Approximations*. Springer, 1994. doi:10.1007/BFb0045375.
- 19 Daniel Lokshtanov. Wheel-free deletion is $W[2]$ -hard. In *Proceedings of the 3rd International Symposium on Parameterized and Exact Computation (IPEC)*, pages 141–147, 2008. doi:10.1007/978-3-540-79723-4_14.

- 20 Daniel Lokshantov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. *CoRR*, abs/1604.04111, 2016. Full version of [21]. doi:10.48550/arXiv.1604.04111.
- 21 Daniel Lokshantov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC 2017)*, pages 224–237, 2017. doi:10.1145/3055399.3055456.
- 22 Jelani Nelson. A note on set cover inapproximability independent of universe size. *Electronic Colloquium on Computational Complexity*, TR07-105, 2007. URL: <https://eccc.weizmann.ac.il/eccc-reports/2007/TR07-105/index.html>.
- 23 Laurence A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982. doi:10.1007/BF02579435.

A Parameterized Approximation Scheme for the Geometric Knapsack Problem with Wide Items

Mathieu Mari

Institute of Informatics, University of Warsaw, Poland
IDEAS-NCBR, Warsaw, Poland

Timothé Picavet 

ENS de Lyon, France
Aalto University, Finland

Michał Pilipczuk  

Institute of Informatics, University of Warsaw, Poland

Abstract

We study a natural geometric variant of the classic KNAPSACK problem called 2D-KNAPSACK: we are given a set of axis-parallel rectangles and a rectangular bounding box, and the goal is to pack as many of these rectangles inside the box without overlap. Naturally, this problem is NP-complete. Recently, Grandoni et al. [ESA'19] showed that it is also $W[1]$ -hard when parameterized by the size k of the sought packing, and they presented a parameterized approximation scheme (PAS) for the variant where we are allowed to rotate the rectangles by 90° before packing them into the box. Obtaining a PAS for the original 2D-KNAPSACK problem, without rotation, appears to be a challenging open question.

In this work, we make progress towards this goal by showing a PAS under the following assumptions:

- both the box and all the input rectangles have integral, polynomially bounded sidelengths;
- every input rectangle is wide – its width is greater than its height; and
- the aspect ratio of the box is bounded by a constant.

Our approximation scheme relies on a mix of various parameterized and approximation techniques, including color coding, rounding, and searching for a structured near-optimum packing using dynamic programming.

2012 ACM Subject Classification Theory of computation \rightarrow Fixed parameter tractability; Theory of computation \rightarrow Packing and covering problems

Keywords and phrases Parameterized complexity, Approximation scheme, Geometric knapsack, Color coding, Dynamic programming, Computational geometry

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.33

Funding *Mathieu Mari*: Partially supported by the ERC CoG grant TUGBOAT no 772346.

Timothé Picavet: This work was supported in part by the Research Council of Finland, Grant 333837.

Michał Pilipczuk: This work is a part of project BOBR that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 948057).



1 Introduction

We study a natural geometric variant of the classic KNAPSACK problem, called 2D KNAPSACK and defined as follows. On input, we are given a rectangular box B and a set \mathcal{R} of items, each being a rectangle. The task is to place as many items from \mathcal{R} as possible in B so that the placed items do not overlap. Note that this problem generalizes classic KNAPSACK: given



© Mathieu Mari, Timothé Picavet, and Michał Pilipczuk;
licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 33; pp. 33:1–33:20

Leibniz International Proceedings in Informatics



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

an instance of KNAPSACK with items of sizes a_1, \dots, a_n and a knapsack of size K , we can create an instance of 2D KNAPSACK with B being the $K \times 1$ rectangle and \mathcal{R} consisting of $a_i \times 1$ rectangles, for all $i \in \{1, \dots, n\}$.

As in the case of KNAPSACK, there are two natural variants of the problem depending on how the input is encoded. In the *binary variant*, both B and the rectangles of \mathcal{R} have integral sidelengths encoded in binary, hence these sidelengths can be exponential in the total input size. In the *unary variant* the difference is that the sidelengths are encoded in unary, or equivalently, one assumes that all the sidelengths are bounded polynomially in the total input size. In this work we focus on the unary variant.

Again as in the case KNAPSACK, adopting the unary variant helps tremendously for the design of algorithms for 2D KNAPSACK, for instance due to allowing to perform dynamic programming over the dimensions of the box. While the problem remains NP-hard even in the unary variant [10], Adamaszek and Wiese [1] gave a QPTAS for the problem in this setting. The best approximation factor known to be achievable in polynomial time in the unary variant is $4/3 + \varepsilon$ due to Galvez et al. [6]; earlier, a $(2 + \varepsilon)$ -approximation was given in [8] and a $(558/325 + \varepsilon)$ -approximation was given in [5]. It is believed that the problem should admit a PTAS, but this question remains widely open to this day.

We remark that the abovementioned works also study the weighted variant of the problem. In this work we only consider the unweighted version, hence an interested reader is invited to the relevant discussion in the references.

Recently, Grandoni et al. [7] proposed to approach the question about the existence of a PTAS for 2D KNAPSACK by adding parameterization by the solution size to the picture. That is, they presented a *parameterized approximation scheme (PAS)* with running time of the form $k^{\mathcal{O}(k/\varepsilon)} \cdot n^{\mathcal{O}(1/\varepsilon^3)}$ that either finds a packing of size at least $(1 - \varepsilon)k$ or correctly concludes that there is no packing of size k . However, this result applies *only* to the variant of the problem where each input rectangle can be rotated by 90° before packing it into the box, and the question about the existence of a PAS for 2D KNAPSACK without rotation was explicitly left open by Grandoni et al. This is in contrast with the other mentioned works on 2D KNAPSACK which all apply both to the variant with rotation and without rotation (for the variant with rotation, Galvez et al. [6] reported a better approximation ratio of $5/4 + \varepsilon$).

We note that the PAS of Grandoni et al. actually works in the binary variant of the problem. Also, reliance on approximation is probably necessary: as proved in [7], the exact version of the problem is W[1]-hard when parameterized by k .

Our contribution. In this work we approach – though not completely solve – the open problem left by Grandoni et al. [7] by giving a parameterized approximation scheme with running time of the form $f(k, \varepsilon, \delta) \cdot n^{g(\varepsilon)}$ for 2D KNAPSACK under the following assumptions:

- we consider the unary variant of the problem, thus the dimensions of the box are bounded polynomially in n ;
- we assume that every item is *wide*: its width is not smaller than its height; and
- we assume that the *aspect ratio* (ratio between the dimensions, always greater or equal to 1) of the box is at most δ .

See Theorem 1 for a formal statement of our result and an explicit formula for the running time. Note that in the context of the variant with rotation, the second assumption can be always achieved by rotating every input rectangle so that it is wide, while the third assumption for $\delta = 1$ can be obtained by scaling both the box and all rectangles on input.

Let us elaborate on our approach and how it is different from the approach of Grandoni et al. [7]. The approach of [7] can be summarized as follows.

- Consider a hypothetical packing \mathcal{S} of size k .
- *Freeing a strip*: Remove a small fraction of \mathcal{S} and shift the items slightly in order to free up a horizontal strip of height N/k^b at the bottom of the box, where N is the sidelength of the box and $b = \mathcal{O}(1/\varepsilon)$ is an integer. This strip can now accommodate all *thin* items: those of height at most N/k^{b+1} .
- *Resource augmentation*: After the previous step, we may assume that all items are *large* – both dimensions are at least N/k^{b+1} – and there is still a considerable strip free at the bottom of the box. Now, one can round the heights of items up to the nearest multiplicity of, say, N/k^{b+2} and argue that even the rounded items can be packed, due to the free strip at the bottom of the box. After rounding the rectangles have at most $k^{\mathcal{O}(1)}$ different heights, so keeping k narrowest rectangles of each possible height gives us a polynomial in k number of candidate rectangles that can be reasonably used in the packing. This easily leads to a PAS.

The possibility of rotating rectangles is crucially used in the second step, freeing a strip. Without this assumption, thin rectangles come in two different flavors: there are *wide* rectangles, of very small height and possibly large width, and symmetric *tall* rectangles. The strategy of freeing a strip presented in [7] can be applied also in the setting without rotation, but then it results in either freeing a horizontal strip at the bottom of the box, or a vertical strip at the left side of the box; there is no control over which strip will be freed. Consequently, only one type of thin rectangles can be disposed of as a result of freeing the strip, and there is no control over which one it is.

In the setting of Grandoni et al., our assumptions on the problem essentially mean that we allow the existence of wide rectangles, but not of the tall ones. The application of the approach of Grandoni et al. could result in freeing a vertical strip, in which the wide rectangles cannot fit. Consequently, we do not see how to fix the approach presented in [7] to solve our case where only wide rectangles are present and no rotation is allowed. We therefore abandon this approach and propose a completely new one.

Instead, we prove a different result about the existence of a well-structured near-optimum solution. Our structural lemma (Lemma 9) says that at the cost of sacrificing a small fraction of rectangles, the considered packing can be divided into regions B_1, B_2, \dots, B_m so that:

- Every region B_i is delimited by the left side of the box, the right side of the box, and two x -monotone axis-parallel polylines connecting the left and the right side. Moreover, each of the polylines defining the division B_1, B_2, \dots, B_m consists of $\mathcal{O}(1/\varepsilon)$ segments.
- Every region B_i is either *light* – contains only $\mathcal{O}(1/\varepsilon^2)$ rectangles from the packing – or *roundable* – rectangles within B_i could be packed inside B_i even after rounding them to the nearest multiple of (roughly) N_1/k^2 , where N_1 is the width of the box.

Having such a structural lemma, a near-optimum solution can be constructed using a bottom-up dynamic programming that guesses the regions B_i one by one. For each region B_i we consider two cases: either B_i is light and a solution within it can be guessed (essentially) by brute-force, or B_i is roundable and using the same trick as in [7], one can restrict attention to $k^{\mathcal{O}(1)}$ many different candidates for rectangles that will be packed into B_i .

There is a technical caveat in the plan presented above. Namely, in dynamic programming we need to make sure that we do not reuse the same rectangle from \mathcal{R} in two or more different regions B_i . We resolve this issue using color-coding. Namely, by applying color-coding upfront we may assume that all the rectangles in \mathcal{R} are colored with k colors, and we look for a packing consisting of rectangles of pairwise different colors. Then our dynamic programming keeps track of the subset of colors that have already been used, which adds only another dimension of size 2^k to the dynamic programming table.

2 Preliminaries

Basic terminology. For a positive integer N , we write $[N] = \{1, 2, \dots, N\}$. For a pair of reals (x, y) , we call x the vertical coordinate and y the horizontal coordinate.

A *rectangle* is a pair of positive integers $R = (w, h) \in \mathbb{Z}_+^2$, and a *placed rectangle* is a set of the form $Q = [x, x + w] \times [y, y + h] \subseteq \mathbb{R}^2$, where $R = (w, h)$ is a rectangle and $(x, y) \in \mathbb{Z}^2$ is the bottom-left corner of Q ; we will also say that such Q is a *placement* of R . In the notation, we will sometimes treat placed rectangles as their non-placed counterparts; the meaning of this will be always clear from the context.

Both for placed and non-placed rectangles, w and h are called the *width* (length on the horizontal dimension) and the *height* (length on the vertical dimension), respectively, and may be denoted by $w(P)$ and $h(P)$, where P is the (placed) rectangle in question. The *interior* of a placed rectangle $Q = [x, x + w] \times [y, y + h]$ is the set $I(Q) = (x, x + w) \times (y, y + h)$. Two placed rectangles *overlap* if their interiors intersect.

A *zone* is simply a subset of \mathbb{R}^2 . For a zone Z and a set of placed rectangles \mathcal{R} , by $\mathcal{R}[Z] = \{R \in \mathcal{R} \mid R \subseteq Z\}$ we denote the set of all rectangles from \mathcal{R} that are entirely contained in Z . For a zone Z and a set of non-placed rectangles \mathcal{R} , a *packing* of \mathcal{R} in Z is a set $\mathcal{R}' = \{R' : R \in \mathcal{R}\}$ consisting of pairwise non-overlapping placed rectangles contained in Z , where R' is a placement of R for each $R \in \mathcal{R}$.

The problem and the main result. In the (parameterized variant of) 2D KNAPSACK problem, we are given a rectangular zone $B = [0, N_1] \times [0, N_2] \subseteq \mathbb{R}^2$ called the *box*, where N_1, N_2 are positive integers, a set \mathcal{R} of rectangles called *items*, and an integer k . The question is whether there exists a packing of some k items from \mathcal{R} in the box B .

In the context of an instance (B, \mathcal{R}, k) of 2D KNAPSACK, the *size* of the box B is $\|B\| = N_1 + N_2$, and the *aspect ratio* of B is $\delta(B) = \max\left(\frac{N_1}{N_2}, \frac{N_2}{N_1}\right) \geq 1$. Further, an item $R \in \mathcal{R}$ is *wide* if $w(R) \geq h(R)$. Note that in the variant of the problem where rotations by 90° are allowed, one may always rotate the items so that they are wide. When the instance (B, \mathcal{R}, k) is clear from the context, by a *packing* we mean a packing of a subset of \mathcal{R} in B .

With these definitions in place, we can state our main result.

► **Theorem 1.** *There exists an algorithm that given an accuracy parameter $\varepsilon > 0$ and an instance (B, \mathcal{R}, k) of 2D KNAPSACK, where \mathcal{R} consists only of wide items, either returns a packing of size at least $(1 - \varepsilon)k$ or correctly concludes that there is no packing of size k . The running time of the algorithm is $\delta(B)^{\mathcal{O}(k)} \cdot (k + 1/\varepsilon)^{\mathcal{O}(k+1/\varepsilon^2)} \cdot (|\mathcal{R}| \|B\|)^{\mathcal{O}(1/\varepsilon^2)}$.*

Polylines and containers. In our algorithm for 2D KNAPSACK we will decompose the box into zones delimited by borders of low complexity, allowing those borders to be efficiently guessed. Formally, each border will be a polyline defined as follows.

► **Definition 2 (Axis-parallel polyline).** *An axis-parallel polyline \mathcal{P} is a union of horizontal or vertical segments S_1, S_2, \dots, S_m such that for $1 \leq i \leq m - 1$, the end of segment S_i is the beginning of segment S_{i+1} . Then m is called the complexity of \mathcal{P} .*

For brevity, axis-parallel polylines will be just called polylines. We will only work with *monotone* polylines, meaning that all horizontal coordinates of points on S_j will not be smaller than the horizontal coordinates of the points on S_i , whenever $i < j$. A polyline P *crosses* a placed rectangle R if P intersects the interior of R .

Next we introduce *containers*. We will use them to capture the idea of decomposing the box into zones.

► **Definition 3** (Container). A container \mathcal{C} is a union of horizontal or vertical segments S_1, S_2, \dots, S_m such that:

- for $1 \leq i \leq m - 1$, the end of segment S_i is the beginning of segment S_{i+1} , and
- the end of segment S_m is the beginning of segment S_1 .

Furthermore, we require that \mathcal{C} is weakly-simple (as introduced in [9, 3, 2]) in the following sense: if S^1 is the unit circle and $\gamma: S^1 \rightarrow \mathbb{R}^2$ is a parameterization of \mathcal{C} , then for every $\varepsilon > 0$ there exists an injective continuous $\gamma_\varepsilon: S^1 \rightarrow \mathbb{R}^2$ such that $\max_{v \in S^1} \|\gamma(v) - \gamma_\varepsilon(v)\| \leq \varepsilon$ for some norm $\|\cdot\|$.

The inside of the container, denoted $I(\mathcal{C})$, is the bounded open region delimited by the segments. Moreover, the complexity of the container is defined as m .

Note that the inside of a container is not necessarily connected. For clarification, see Figure 1.

3 Exact algorithm

In this section, we give an exact algorithm for the problem, which will be later used as a subroutine in the proof of Theorem 1. The point here is that we allow the box to be delimited by an arbitrary container, and we measure the running time in the complexity of the container. Formally, we will prove the following statement.

► **Lemma 4.** *Given a set of rectangles \mathcal{R} and a container \mathcal{C} of complexity m , one can determine whether there is a packing of the rectangles of \mathcal{R} inside \mathcal{C} in time $(m + |\mathcal{R}|)^{\mathcal{O}(|\mathcal{R}|)}$.*

The main idea of our algorithm is to push the packing bottom-left, as explained in the next definition.

► **Definition 5.** *A packing \mathcal{R} inside a container \mathcal{C} is said to be pushed bottom-left if for every rectangle $R \in \mathcal{R}$, its left (resp. bottom) side intersects either a vertical (resp. horizontal) segment of the container, or a right (resp. top) side of another rectangle $R' \in \mathcal{R}$.*

It is not hard to see that if a packing is pushed bottom-left, then there must be a rectangle in the packing whose left and bottom sides rest on the perimeter of the container. This is formally proved in the following statement. (Proofs of statements marked with (♠) can be found in Appendix B.)

► **Proposition 6** (♠). *Suppose \mathcal{R} is a non-empty packing of rectangles inside a container \mathcal{C} that is pushed bottom-left. Then there exists a rectangle $R \in \mathcal{R}$ such that its left side intersects a vertical segment of the container and its bottom side intersects an horizontal segment of the container.*

With Proposition 6 established, we can conclude our goal using a simple branching strategy.

Proof of Lemma 4. We prove a stronger statement where we allow \mathcal{C} to be the union of several disjoint containers, and we let m be the sum of their complexities.

Suppose there is a packing \mathcal{S} of the rectangles of \mathcal{R} into \mathcal{C} . We can assume without loss of generality that \mathcal{S} is pushed bottom-left within every container of \mathcal{C} . Now by Proposition 6, there exists a rectangle R such that its left (resp. bottom) side intersects a vertical (resp. horizontal) segment of a container in \mathcal{C} .

So here is a recursive procedure to solve the problem. First, guess (by trying all possibilities) the rectangle R satisfying the condition above; there are n different possibilities for R , where $n = |\mathcal{R}|$. Second, guess which pair of segments of the containers intersect the left and the bottom side of R ; there are at most m^2 possibilities. Place rectangle R according

to the latter guess and verify that it is indeed fully contained in \mathcal{C} . Then, “carve out” the rectangle, i.e., define a new union of containers \mathcal{C}' so that the $I(\mathcal{C}') = I(\mathcal{C}) \setminus R$. It is easy to see that the total complexity of the new union of containers \mathcal{C}' is at most $m + 6$ and it can be computed in time polynomial in m . Then, recurse on \mathcal{R}' and \mathcal{C}' where $\mathcal{R}' = \mathcal{R} \setminus \{R\}$.

It is clear that the algorithm is correct. To analyze its running time, note that the recursion tree has depth bounded by n and branching bounded by $n(m + 6n)^2$, hence it consists of $(m + n)^{\mathcal{O}(n)}$ nodes. The internal computation at each node take time polynomial in n and m , so the total running time of $(m + n)^{\mathcal{O}(n)}$ follows. ◀

4 Giving structure to the packing

In this section we prove structural results that can be summarized as follows: at the cost of sacrificing a small fraction of the packing, one can apply resource augmentation – round the packing – so that it gains a certain structure. Once this structure is achieved, we will argue later that structured packings can be efficiently computed using dynamic programming.

Throughout this section we fix an instance (B, \mathcal{R}, k) of 2D KNAPSACK, where $B = [0, N_1] \times [0, N_2]$ and \mathcal{R} consists only of wide rectangles: $w(R) \geq h(R)$ for all $R \in \mathcal{R}$.

To perform resource augmentation, we need the following notion of rounding a rectangle. Informally, a rounded rectangle is the original rectangle with its width rounded up to the nearest multiple of $\ell' = \ell^2/N_1$; here is a formal definition.

► **Definition 7** (Rounded rectangles). *Let $R = (w, h)$ be a rectangle and $\ell > 0$ be a positive real. Then the ℓ -rounded rectangle $\text{round}_\ell(R)$ is the rectangle $(\ell' \lceil w/\ell' \rceil, h)$ where $\ell' = \ell^2/N_1$. For a set \mathcal{R} of rectangles, we define similarly $\text{round}_\ell(\mathcal{R}) = \{\text{round}_\ell(R) : R \in \mathcal{R}\}$.*

As mentioned, the key idea behind our algorithm is to look for a specifically structured packing. This structure is quantified formally in the following definition. Broadly speaking, we look for a packing that is partitioned into regions of low complexity and such that the rectangles in each region behave well.

► **Definition 8** (Structured packing). *Fix any $\varepsilon, \ell > 0$. Consider a set of pairwise non-intersecting monotone polylines P_1, P_2, \dots, P_m contained in the box B , where each P_i starts at the left side of B and finishes at the right side of B , and the polylines P_1, \dots, P_m are naturally numbered from bottom to top. We define the partition of the box B into regions B_0, B_1, \dots, B_m so that each region B_i is delimited by the polylines P_i and P_{i+1} and the left and the right side of B (here we define for convenience P_0 to be the bottom side of B and P_{m+1} to be the top of B).*

We say that a packing of rectangles \mathcal{Q} in B is an (ε, ℓ) -structured packing if every rectangle in \mathcal{Q} has width at least 2ℓ and there exist polylines P_1, P_2, \dots, P_m as above, each of complexity at most $4/\varepsilon + 1$, such that no rectangle of \mathcal{Q} is crossed by any polyline P_i , $i \in [m]$, and for each $i, 0 \leq i \leq m$ at least one of the following conditions holds:

- $|\mathcal{Q}[B_i]| \leq 2/\varepsilon^2$, or
- $\text{round}_\ell(\mathcal{Q}[B_i])$ can be packed into B_i .

The rest of the section is dedicated to proving the following structural lemma (recall that the instance (B, \mathcal{R}, k) is fixed in the context):

► **Lemma 9** (structural lemma). *Suppose $\ell > 0$ is a positive real such that there is a packing of size k consisting of rectangles of width at least 2ℓ each. Then for every $\varepsilon > 0$, there exists also an (ε, ℓ) -structured packing of size at least $(1 - 3\varepsilon)k$.*

This section is divided into 3 parts. In the first subsection we study the assumed packing of size k and define an associated *conflict graph*, which turns out to be planar. In the second subsection, we show that if there exists a packing of rectangles in a specific zone on the box, then at the cost of removing a few rectangles, there exists a packing of the rounded rectangles into a slightly bigger rounded version of the zone. In the last section, we define the specific polylines that will divide the zones and finish the proof of the Lemma 9.

By assumption, there exists a packing \mathcal{S} in B consisting of k rectangles from \mathcal{R} , each of width at least 2ℓ . Fix \mathcal{S} for the remainder of this section.

4.1 Conflict graph

For the definition of the conflict graph, we need the following notion of horizontal visibility.

► **Definition 10.** *Two different placed rectangles $R, R' \in \mathcal{S}$ see each other if there is an horizontal segment s intersecting the interior of the right side of R and the interior of the left side of R' (or vice versa) such that s does not intersect any other rectangle of \mathcal{S} . Notice that s may consist of a single point, if R and R' are intersecting. For convenience, we extend this definition to the case where R is the left side of B or R' is the right side of B . For instance with the left side of B we associate the placed rectangle $R_{\text{left}} = [-1, 0] \times [0, N_2]$ and say that R and the left side of B see each other if R_{left} and R see each other; similarly for the right side of B . The left side of B and the right side of B do not see each other.*

Note two rectangles intersecting only at their common corner do *not* see each other.

► **Definition 11 (Conflict graph).** *For a packing \mathcal{S} , we define the conflict graph of \mathcal{S} to be the graph G defined as follows: the vertex set contains all the rectangles of \mathcal{S} , and in addition there are two special vertices s and t identified with the left side and the right side of B , respectively. Two vertices of G are adjacent if and only if they see each other.*

It is easy to see that the conflict graph is planar; see Figure 2. We formalize this intuition in the following lemma.

► **Lemma 12 (♠).** *For any packing \mathcal{Q} , the conflict graph of \mathcal{Q} is planar.*

4.2 Packing rounded rectangles

Next, we analyze a packing within some zone $Z \subseteq \mathbb{R}^2$, with the goal of understanding when and how the rectangles of this packing can be rounded to obtain a rounded packing of substantial size. We fix some positive real $\ell > 0$ for the rest of this subsection.

First, we need some definitions about expanding zones.

► **Definition 13.** *Let $Z \subseteq \mathbb{R}^2$. We define:*

- *the negatively shifted zone $\overleftarrow{Z}(\ell) = \left(\bigcup_{(x,y) \in Z} [x - \ell, x] \times \{y\} \right) \cap [0, N_1 - \ell] \times [0, N_2]$,*
- *the positively shifted zone $\overrightarrow{Z}(\ell) = \left(\bigcup_{(x,y) \in Z} [x, x + \ell] \times \{y\} \right) \cap [0, N_1] \times [0, N_2]$,*
- *and the rounded zone $\overleftrightarrow{Z}(\ell) = \left(\bigcup_{(x,y) \in Z} [x - \ell, x + \ell] \times \{y\} \right) \cap [0, N_1] \times [0, N_2]$.*

Note that if $Z' = \overleftarrow{Z}(\ell)$ then $\overleftrightarrow{Z'}(\ell) = \overrightarrow{Z}(\ell)$, and that the first two definitions are not symmetric.

Our main goal in this subsection is to prove the following lemma. It intuitively says that at the cost of removing an st -separator in the conflict graph, one can find a packing of the rounded rectangles into a slightly extended zone. Here, an st -separator is a set of vertices (rectangles) that hits every s - t path.

► **Lemma 14.** *Let \mathcal{Q} be a packing in a zone $Z \subseteq B$ such that every rectangle of \mathcal{Q} has width at least 2ℓ . Further, let C be an st -separator in the conflict graph of \mathcal{Q} . Then $\text{round}_\ell(\mathcal{Q} \setminus C)$ can be packed inside the zone $\overleftarrow{Z}(\ell)$.*

The first step towards the proof of Lemma 14 is to repack \mathcal{Q} into the negatively shifted zone Z at the cost of deleting a few rectangles.

► **Proposition 15 (♠).** *Let \mathcal{Q} be a packing in a zone $Z \subseteq B$ such that every rectangle of \mathcal{Q} has width at least 2ℓ . Further, let C be an st -separator in the conflict graph of \mathcal{Q} . Then $\mathcal{Q} \setminus C$ can be packed inside the zone $\overleftarrow{Z}(\ell)$.*

For an illustration of the proof, see Figure 3.

Now that we have emptied a strip to the right of the zone, we can do some resource augmentation in order to replace the original rectangles by their rounded versions, while still being able to pack them inside the rounded zone.

► **Proposition 16 (♠).** *Let \mathcal{Q} be a packing in a zone $Z \subseteq [0, N_1 - \ell] \times [0, N_2]$ such that every rectangle of \mathcal{Q} has width at least 2ℓ . Then $\text{round}_\ell(\mathcal{Q})$ can be packed inside the zone $\overrightarrow{Z}(\ell)$.*

For clarification, see Figure 4. We may now combine Proposition 15 and Proposition 16 to achieve our goal.

Proof of Lemma 14. Apply Proposition 15 and Proposition 16 to get that $\text{round}_\ell(\mathcal{Q} \setminus C)$ can be packed in $\overrightarrow{Z'}(\ell)$, where $Z' = \overleftarrow{Z}(\ell)$. As $\overrightarrow{Z'}(\ell) = \overleftarrow{Z}(\ell)$, the proof is finished. ◀

4.3 Proof of the Structural Lemma

Finally, in this subsection we define the polylines that we are interested in and prove some results about zones and polylines to finish the proof of Lemma 9. The main idea is to construct some well-chosen polylines by looking at the rectangles on short s - t paths. These polylines are then used to delimit zones in which we can find a separator of bounded size, and apply the ideas of the previous subsections.

Recall that we are working with a packing \mathcal{S} of size k consisting of rectangles of width at least 2ℓ each. Let G be the conflict graph of \mathcal{S} . For every $R \in \mathcal{S}$, by v_R we denote the vertex of G corresponding to R . First, we need to understand how s - t paths in G can be mapped to polylines.

► **Definition 17 (Bottom polyline of a path).** *Consider an s - t path $P = (s, v_{R_1}, v_{R_2}, \dots, v_{R_m}, t)$ in G , and suppose that for each $i \in \{0, 1, \dots, m\}$, that R_i and R_{i+1} see each other is witnessed by the segment $s_i = [x(R_i) + w(R_i), x(R_{i+1})] \times \{y_i\}$ (where $R_0 = s$ and $R_{m+1} = t$). Then define the bottom polyline of P as the polyline \mathcal{P} formed by the union of the following segments:*

- $[x(R_i), x(R_i) + w(R_i)] \times \{y(R_i)\}$ for each $i \in [m]$,
- $\{x(R_i)\} \times [\min\{y(R_i), y_{i-1}\}, \max\{y(R_i), y_{i-1}\}]$ for each $i \in [m]$,
- $\{x(R_i) + w(R_i)\} \times [\min\{y(R_i), y_i\}, \max\{y(R_i), y_i\}]$ for each $i \in [m]$, and
- s_i for each $i \in \{0, 1, \dots, m\}$.

Less formally, \mathcal{P} is the union of the segments s_i joining the rectangles of the path, the bottom sides of the rectangles, and parts of the left/right sides of the rectangles to join the segments to the bottom sides.

Similarly, we define the notion of the top polyline of an s - t path in G . When defining at the same time the top and the bottom polyline of the same path, we always use the same segments s_i to define how rectangles R_i and R_{i+1} should be linked. Finally, we will also need the middle polyline.

► **Definition 18** (Middle polyline of a path). Consider an s - t path $P = (s, v_{R_1}, v_{R_2}, \dots, v_{R_m}, t)$ in G , and suppose that for each $i \in \{0, 1, \dots, m\}$, that R_i and R_{i+1} see each other is witnessed by the segment $s_i = [x(R_i) + w(R_i), x(R_{i+1})] \times \{y_i\}$ (where $R_0 = s$ and $R_{m+1} = t$). Then define the middle polyline of P as the polyline \mathcal{P} formed by the union of the following segments:

- $\{x(R_i) + w(R_i)/2\} \times [\min\{y(R_i) + h(R_i)/2, y_i\}, \max\{y(R_i) + h(R_i)/2, y_i\}]$ for each $i \in [m]$,
- $\{x(R_i) + w(R_i)/2\} \times [\min\{y(R_i) + h(R_i)/2, y_{i+1}\}, \max\{y(R_i) + h(R_i)/2, y_{i+1}\}]$ for each $i \in [m]$, and
- $[\max(x(R_i) + w(R_i)/2, 0), \min(x(R_{i+1}) + w(R_{i+1})/2, N_2)] \times \{y_i\}$ for each $i \in \{0, 1, \dots, m\}$.

Less formally, \mathcal{P} is the union of:

- a vertical segment from the center of each R_i to the vertical position of s_i ,
- a vertical segment from the center of each R_i to the vertical position of s_{i+1} ,
- all segments s_i extended so that they reach the horizontal coordinates of the centers of the corresponding rectangles.

For a visual representation, see Figure 5. The following is clear.

► **Proposition 19.** The top, bottom and middle polylines of a path P have complexity at most $4|P| + 1$, where $|P|$ denotes the number of vertices on P .

Moreover, the middle polyline is defined so that we have space to the left and the right when performing resource augmentation. This will be made clear in the following definitions and lemmas; see Figure 6.

► **Proposition 20** (♠). Suppose P is an s - t path in the conflict graph G of the packing \mathcal{S} . Let \mathcal{P} be the middle polyline of P and let \mathcal{Q} be the packing obtained from \mathcal{S} by removing all the rectangles participating in P . Then \mathcal{P} does not cross $\overleftarrow{\mathcal{Q}}(\ell)$. The same goes for $\overrightarrow{\mathcal{Q}}(\ell)$, and therefore also for $\overleftrightarrow{\mathcal{Q}}(\ell)$.

Next, we need the following graph-theoretic observation.

► **Proposition 21.** Let G be a graph containing vertices s and t . Suppose every s - t path in G contains at least $1/\varepsilon$ internal vertices. Then G contains an st -separator of size at most $\varepsilon(|V(G)| - 2)$.

Proof. As every s - t path in G contains at least $1/\varepsilon$ internal vertices, one cannot find more than $\varepsilon(|V(G)| - 2)$ internally disjoint s - t paths in G . By Menger's theorem, there is an st -separator of size at most $\varepsilon(|V(G)| - 2)$. ◀

We can now wrap up the section by proving the Structural Lemma.

Proof of Lemma 9. Based on the assumed packing \mathcal{S} , we construct another packing \mathcal{S}' and then we prove that it is structured and has size at least $(1 - 3\varepsilon)k$. Let G be the conflict graph of \mathcal{S} . Let \mathcal{F} be an inclusion-wise maximal family \mathcal{F} of internally disjoint s - t paths in G , each with at most $1/\varepsilon$ internal vertices. As the paths from \mathcal{F} are internally disjoint, we can naturally enumerate them from bottom to top: $\mathcal{F} = \{P_1, P_2, \dots, P_m\}$. For convenience, let $P_0 = P_{m+1} = \emptyset$.

Because the conflict graph is planar by Lemma 12, by the Jordan Curve theorem, for each $i \in \{0, 1, \dots, m\}$ there is a set V_i of vertices of G that lies inside the cycle $P_i \cup P_{i+1}$. By construction of the conflict graph, V_i is exactly the set of rectangles lying in the area Z_i delimited by the box, the top polyline of P_i and the bottom polyline of P_{i+1} . For each V_i we construct a separating polyline \mathcal{P}_i as follows:

33:10 Parameterized Approximation Scheme for Geometric Knapsack with Wide Items

- If $|V_{i-1}| \leq 1/\varepsilon^2$ and $|V_i| \leq 1/\varepsilon^2$, select the bottom polyline of P_i as the separating polyline.
- Otherwise, select the middle polyline of P_i as the separating polyline.

All the polylines created are of complexity at most $4/\varepsilon + 1$ by Proposition 19. They partition the box into regions $B_0, B_1, \dots, B_{m+1} \subseteq B$, from the bottom to the top. Note that $B_i \supseteq Z_i$ for each relevant i .

Notice that in $G[V_i \cup \{s, t\}]$ there is no s - t path of length at most $1/\varepsilon$, because \mathcal{F} is maximal. Let C_i be the separator given by Proposition 21 for the graph $G[V_i \cup \{s, t\}]$. Then we have $|C_i| \leq \varepsilon|V_i|$.

We can now specify which rectangles we want to include in \mathcal{S}' . We define \mathcal{S}' to be the union of sets V'_i for $i \in \{0, 1, \dots, m\}$, where

$$V'_i = \begin{cases} V_i \cup V(P_i) \setminus \{s, t\} & \text{if } |V_{i-1}| \leq 1/\varepsilon^2 \text{ and } |V_i| \leq 1/\varepsilon^2, \\ V_i & \text{if } |V_{i-1}| > 1/\varepsilon^2 \text{ and } |V_i| \leq 1/\varepsilon^2, \\ V_i \setminus C_i & \text{otherwise.} \end{cases}$$

We now argue that for each $i \in \{0, 1, \dots, m\}$, either $|V'_i| \leq 2/\varepsilon^2$ and $V'_i \subseteq \mathcal{S}[B_i]$, or $\text{round}_\ell(V'_i)$ can be packed in B_i .

First, observe that if $|V_i| \leq 1/\varepsilon^2$, then $|V'_i| \leq |V_i \cup V(P_i) \setminus \{s, t\}| \leq 1/\varepsilon^2 + 1/\varepsilon \leq 2/\varepsilon^2$. Further, if $|V_{i-1}| > 1/\varepsilon^2$ then $V'_i = V_i$ and trivially $\mathcal{S}[B_i] \supseteq \mathcal{S}[Z_i] = V_i$, and if $|V_{i-1}| \leq 1/\varepsilon^2$ then \mathcal{P}_i is the bottom polyline of P_i and we have $\mathcal{S}[B_i] \supseteq V_i \cup V(P_i) \setminus \{s, t\} = V'_i$ as well.

Second, consider the case when $|V_i| > 1/\varepsilon^2$. Notice that then \mathcal{P}_i is the middle polyline of P_i and \mathcal{P}_{i+1} is the middle polyline of P_{i+1} , and $V'_i = V_i \setminus C_i$. Because V_i can be packed inside Z_i , we can use Lemma 14 on V_i and Z_i to pack $\text{round}_\ell(V'_i)$ into $\overleftarrow{\langle Z_i \rangle}(\ell)$. By Proposition 20, we know that $\overleftarrow{\langle Z_i \rangle}(\ell) \subseteq B_i$, hence we can pack $\text{round}_\ell(V'_i)$ into B_i .

We conclude that indeed, \mathcal{S}' is an (ε, ℓ) -structured packing, as witnessed by the polylines \mathcal{P}_i for $i \in [m]$. What is left to show is that $|\mathcal{S}'| \geq (1 - 3\varepsilon)k$. Call an index $i \in [m]$ *heavy* if $|V_i| > 1/\varepsilon^2$. Observe that $\mathcal{S}' \supseteq \mathcal{S} \setminus \bigcup_{i: \text{heavy}} (C_i \cup V(P_i) \cup V(P_{i+1}))$, hence it suffices to prove that $\left| \bigcup_{i: \text{heavy}} (C_i \cup V(P_i) \cup V(P_{i+1})) \setminus \{s, t\} \right| \leq 3\varepsilon k$. Fix a heavy index i . First, observe that $|(V(P_i) \cup V(P_{i+1})) \setminus \{s, t\}| \leq 2/\varepsilon \leq 2\varepsilon|V_i|$, as each path P_i has at most $1/\varepsilon$ internal vertices. Second, by construction we have $|C_i| \leq \varepsilon|V_i|$. Summing those inequalities throughout all heavy i yields that

$$\left| \bigcup_{i: \text{heavy}} (C_i \cup V(P_i) \cup V(P_{i+1})) \setminus \{s, t\} \right| \leq \sum_{i: \text{heavy}} 3\varepsilon|V_i| \leq 3\varepsilon k,$$

as required. ◀

5 The algorithm

In this section we finalize the proof of Theorem 1. The section is divided into two parts. The first subsection describes an algorithm working under the assumption that the input set \mathcal{R} only contains rectangles of width at least 2ℓ , for some $\ell > 0$. In the second subsection, we show how to obtain the assumption that \mathcal{R} only contains rectangles of width at least 2ℓ for $\ell = N_1/(\delta(B)k^2)$, at the expense of deleting an ε fraction of the rectangles in the packing. Therefore, we get a full algorithm as a corollary.

Throughout this section, fix an instance (B, \mathcal{R}, k) of 2D KNAPSACK, where $B = [0, N_1] \times [0, N_2]$ and \mathcal{R} consists of wide items.

5.1 The algorithm for rectangles of substantial width

The dynamic programming algorithm will gradually guess a good partition of the box into regions (that we know exists by Lemma 9), and then solve the problem in each region independently. In order to avoid repeating the use of the same rectangles in different regions, we use color-coding.

► **Definition 22** (Good coloring). *Given a set of rectangles \mathcal{R} and a subset $\mathcal{S} \subseteq \mathcal{R}$ of size k , a function $col: \mathcal{R} \rightarrow [k]$ is a good coloring for \mathcal{S} if rectangles of \mathcal{S} have pairwise different colors under col .*

We cannot directly guess a good coloring of the rectangles, as a priori there are too many candidates. We instead use the following classic result of Naor et al. [11], which says that there is only an fpt-sized family of candidates for a good coloring.

► **Proposition 23** (Naor et al. [11]). *For every set \mathcal{R} and positive integer k , there exists a family \mathcal{F} of colorings of \mathcal{R} with color set $[k]$ such that $|\mathcal{F}| \leq e^k k^{\mathcal{O}(\log k)} \log |\mathcal{R}|$ and for every $\mathcal{S} \subseteq \mathcal{R}$ of size k , in \mathcal{F} there is a good coloring for \mathcal{S} . Moreover, \mathcal{F} can be computed in time $e^k k^{\mathcal{O}(\log k)} |\mathcal{R}| \log |\mathcal{R}|$.*

Next, we observe that once the number of different widths in the instance has been bounded, one can restrict attention to a small set of candidate rectangles. For this, notice the following: if we have a colored packing (a packing of colored rectangles) of size k that contains a rectangle R , and in the packing we did not use another rectangle R' of the same color and width as R , but satisfying $h(R') \leq h(R)$, then we can replace R with R' and we will still have a colored packing. This observation leads to defining the following operation.

► **Definition 24** ($\text{reduce}_k(\mathcal{R}, col)$). *Suppose $col: \mathcal{R} \rightarrow [k]$ is a coloring of a set of rectangles \mathcal{R} with color set $[k]$. Then for a positive integer w and color $i \in [k]$, let $\mathcal{R}_{w,i}$ be the set of k smallest-height rectangles among the rectangles of $\{R \in \mathcal{R} \mid w(R) = w, col(R) = i\}$. In case $|\{R \in \mathcal{R} \mid w(R) = w, col(R) = i\}| < k$, we set $\mathcal{R}_{w,i} = \{R \in \mathcal{R} \mid w(R) = w, col(R) = i\}$. We define $\text{reduce}_k(\mathcal{R}) = \bigcup_{w \in w(\mathcal{R}), i \in [k]} \mathcal{R}_{w,i}$.*

Notice that $\text{reduce}_k(\mathcal{R})$ contains at most $k^2 |w(\mathcal{R})|$ elements: for every possible width and every possible color, there are at most k rectangles of this specific width and of smallest height. Also, we have the following very simple observation.

► **Lemma 25** (♠). *Suppose \mathcal{R} is a set of rectangles and $col: \mathcal{R} \rightarrow [k]$ is a coloring function such that $k' \leq k$ rectangles from \mathcal{R} with pairwise different colors can be packed in a zone $Z \subseteq \mathbb{R}^2$. Then one can also pack in Z a set of k' rectangles from $\text{reduce}_k(\mathcal{R}, col)$ with pairwise different colors.*

Next, we use the following definitions to guess the polylines in a bottom to top order. For two monotone polylines $\mathcal{P}, \mathcal{P}'$ that start at the left side of B and finish at the right side of B , we say that \mathcal{P}' is below \mathcal{P} (denoted by $\mathcal{P}' \leq \mathcal{P}$), if for every x, y, y' , $(x, y) \in \mathcal{P}$ and $(x, y') \in \mathcal{P}'$ implies $y' \leq y$. We write $\mathcal{P}' < \mathcal{P}$ if $\mathcal{P}' \leq \mathcal{P}$ and $\mathcal{P}' \neq \mathcal{P}$. Given two polylines $\mathcal{P}' < \mathcal{P}$, we want to be able to solve the problem in the following sub-region:

► **Definition 26** ($\text{container}(\mathcal{P}', \mathcal{P})$). *For polylines $\mathcal{P}' < \mathcal{P}$, $\text{container}(\mathcal{P}', \mathcal{P})$ is the container (c.f. Definition 3) delimited by the box B , \mathcal{P}' at the bottom and \mathcal{P} at the top.*

Notice that if \mathcal{P} has complexity m and \mathcal{P}' has complexity m' then $\text{container}(\mathcal{P}', \mathcal{P})$ has complexity $m + m' + 2$.

Now we give the algorithm in the case when all rectangles in \mathcal{R} have substantial width. This algorithm is encapsulated in the following lemma.

► **Lemma 27.** *There is an algorithm that given $\varepsilon > 0$ and an instance (B, \mathcal{R}, k) of 2D KNAPSACK in which all items are wide and have width at least N_1/α , either returns a packing of size at least $(1 - \varepsilon)k$ or correctly concludes that there is no packing of size k . The running time is $(k + 1/\varepsilon)^{\mathcal{O}(k+1/\varepsilon)} \cdot \alpha^{\mathcal{O}(k)} \cdot (|\mathcal{R}|||B||)^{\mathcal{O}(1/\varepsilon^2)}$.*

Proof. Let $\ell = N_1/(2\alpha)$; thus every rectangle on input has width at least 2ℓ . For clarity of presentation we allow the algorithm to output a packing of size at least $(1 - 3\varepsilon)k$; then the result as stated in the lemma can be obtained by rescaling ε by factor 3.

We first explain the algorithm. Compute \mathcal{F} as given by Proposition 23. Then, guess (by trying all choices) a coloring $\text{col} \in \mathcal{F}$. The idea is now to use dynamic programming to compute a maximum-size structured packing for the colored instance. More precisely, for every monotone polyline \mathcal{P} of complexity at most $4/\varepsilon + 1$ connecting the left and the right side of B , and for every $C \subseteq [k]$, we shall compute the value $\text{dp}[\mathcal{P}, C]$ defined as follows: $\text{dp}[\mathcal{P}, C]$ is a maximum-size packing that contains only rectangles with colors in C , is colored injectively by col , and is placed entirely below \mathcal{P} with the added constraint that it is a subset of some (ε, ℓ) -structured packing.

To compute the value $\text{dp}[\mathcal{P}, C]$ for given \mathcal{P} and C , we iterate over all polylines \mathcal{P}' of complexity at most $4/\varepsilon + 1$ that are below \mathcal{P} . Let $B' = \text{container}(\mathcal{P}', \mathcal{P})$ be the container between \mathcal{P} and \mathcal{P}' . Iterate over all $C' \subseteq C$; this is the set of colors guessed to be used in B' . Let $\mathcal{R}' = \text{reduce}_k(\text{round}_\ell(\mathcal{R}), \text{col})$ be the reduced set of rounded rectangles, where colors are naturally inherited from \mathcal{R} during rounding. Compute the following packings:

- \mathcal{S}_1 is the largest packing in B' consisting of at most $2/\varepsilon^2$ rectangles with pairwise different colors from C' . This packing can be computed in time $|\mathcal{R}'|^{\mathcal{O}(1/\varepsilon^2)} \cdot (1/\varepsilon)^{\mathcal{O}(1/\varepsilon^2)}$ by first guessing the set of rectangles participating in it, and then checking whether the packing can be realized using the algorithm of Lemma 4.
- \mathcal{S}_2 is the largest packing in B' consisting of at most k rectangles from \mathcal{R}' with pairwise different colors from C' . Again, this packing can be computed in time $|\mathcal{R}'|^{\mathcal{O}(k)} \cdot (k+1/\varepsilon)^{\mathcal{O}(k)}$ by first guessing the set of rectangles participating in it, and then checking whether the packing can be realized using the algorithm of Lemma 4.

Iterate over $\mathcal{S} \in \{\mathcal{S}_1, \mathcal{S}_2\}$, and keep as $\text{dp}[\mathcal{P}, C]$ the set $\text{dp}[\mathcal{P}', C \setminus C'] \cup \mathcal{S}$ of maximum size over all the sets iterated on. Finally, as the solution to the overall problem, return $\text{dp}[\mathcal{P}, [k]]$ where \mathcal{P} is the top side of B , provided this packing has size at least $(1 - 3\varepsilon)k$. Otherwise, return that there is no packing of size k .

This concludes the description of the algorithm. We are left with (i) analyzing its running time and (ii) arguing that in case there is a packing of size at least k , the algorithm will output a packing of size at least $(1 - 3\varepsilon)k$.

Let us start with assertion (ii). For this, suppose there exists a packing \mathcal{S} of size k . Since all rectangles of \mathcal{S} have width at least 2ℓ , by Lemma 9 there exists an (ε, ℓ) -structured packing \mathcal{S}' of size at least $(1 - 3\varepsilon)k$. Further, by the properties of \mathcal{F} , there exists $\text{col} \in \mathcal{F}$ such that col is injective on \mathcal{S}' . Now, let $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_m$ be the polylines witnessing the structuredness of \mathcal{S}' , and let $\emptyset = C_0 \subseteq C_1 \subseteq C_2 \subseteq \dots \subseteq C_m \subseteq C_{m+1} = [k]$ be such that C_i is the sets of colors used by the rectangles of \mathcal{S}' lying below \mathcal{P}_i , where \mathcal{P}_0 and \mathcal{P}_{m+1} are the bottom and the top side of B , respectively. A straightforward inductive argument using the structuredness of \mathcal{S}' and Lemma 25 shows now that for $i = 0, 1, \dots, m + 1$, the cell $\text{dp}[\mathcal{P}_i, C_i]$ will contain a packing of size at least as large as the number of rectangles of \mathcal{S}' lying below \mathcal{P}_i . Hence, the algorithm will return a packing of size at least $|\mathcal{S}'| \geq (1 - 3\varepsilon)k$, as promised.

We are left with analyzing the running time. The number of different colorings $\text{col} \in \mathcal{F}$ is $|\mathcal{F}| \leq 2^{\mathcal{O}(k)} \cdot \log |\mathcal{R}|$. Further, observe that the number of different polylines considered by the algorithm is bounded by $||B||^{\mathcal{O}(1/\varepsilon)}$ and there are 2^k different subsets of colors. Hence, the

total number of cells $\text{dp}[\mathcal{P}, C]$ considered by the algorithm is bounded by $2^k \cdot \|B\|^{\mathcal{O}(1/\varepsilon)}$. As argued, the time spent on computing a single value of $\text{dp}[\mathcal{P}, C]$ is bounded by $2^k \cdot \|B\|^{\mathcal{O}(1/\varepsilon)}$ (the number of choices for \mathcal{P}' and C') times

$$|\mathcal{R}|^{\mathcal{O}(1/\varepsilon^2)} \cdot (1/\varepsilon)^{\mathcal{O}(1/\varepsilon^2)} + |\mathcal{R}'|^{\mathcal{O}(k)} \cdot (k + 1/\varepsilon)^{\mathcal{O}(k)}.$$

Observe now that the rectangles of $\text{round}_\ell(\mathcal{R})$ have at most $\mathcal{O}(N_1/\ell')$ different widths, where $\ell' = \ell^2/N_1$. Since $\ell = N_1/2\alpha$, we conclude that the total number of different widths of the rectangles of $\text{round}_\ell(\mathcal{R})$ is bounded by

$$\mathcal{O}(N_1/\ell') = \mathcal{O}(N_1^2/\ell^2) \leq \mathcal{O}(\alpha^2).$$

Therefore,

$$|\mathcal{R}'| = |\text{reduce}_k(\text{round}_\ell(\mathcal{R}, \text{col}))| \leq \mathcal{O}(\alpha^2 k^2).$$

Putting everything together, we infer that the running time of the algorithm is bounded by

$$\begin{aligned} & 2^{\mathcal{O}(k)} \cdot \log |\mathcal{R}| \cdot 2^{\mathcal{O}(k)} \cdot \|B\|^{\mathcal{O}(1/\varepsilon)} \cdot \left(|\mathcal{R}|^{\mathcal{O}(1/\varepsilon^2)} \cdot (1/\varepsilon)^{\mathcal{O}(1/\varepsilon^2)} + (\alpha^2 k^2)^{\mathcal{O}(k)} \cdot (k + 1/\varepsilon)^{\mathcal{O}(k)} \right) \\ & \leq (k + 1/\varepsilon)^{\mathcal{O}(k+1/\varepsilon^2)} \cdot \alpha^{\mathcal{O}(k)} \cdot (|\mathcal{R}||B|)^{\mathcal{O}(1/\varepsilon^2)}, \end{aligned}$$

as promised. ◀

5.2 Full algorithm

We now present the complete algorithm, which essentially boils down to making a reduction to the case when all rectangles on input have width at least 2ℓ , where $\ell = N_1/(\delta(B)k^2)$. In the next lemma, we explain how to perform this reduction at the cost of removing εk rectangles from the packing.

► **Lemma 28.** *Let $\varepsilon > 0$. Suppose there is an algorithm \mathcal{A} that, given a 2D KNAPSACK instance $(B = [0, N_1] \times [0, N_2], \mathcal{R}, p)$ in which all items are wide and have width at least $N_1/(\delta q^2)$ and the aspect ratio of B is δ , returns a packing of size at least $(1 - \varepsilon)p$ or attests that there is no packing of size p in time $f(p, q, \varepsilon, \delta, \|B\|, |\mathcal{R}|)$. Then there is an algorithm \mathcal{B} that, given a 2D KNAPSACK instance $(B = [0, N_1] \times [0, N_2], \mathcal{R}, k)$ in which all items are wide and the aspect ratio of B is δ , returns a packing of size $(1 - 2\varepsilon)k$ or attests that there is no packing of size k in time $f(k, k, \varepsilon, \delta, \|B\|, |\mathcal{R}|) + (1/\varepsilon + |\mathcal{R}|)^{\mathcal{O}(1/\varepsilon)}$.*

Proof. We present the algorithm \mathcal{B} . Without loss of generality, we can assume $k > 1/\varepsilon$, as otherwise the number of rectangles in the sought packing is at most $1/\varepsilon$ and we can solve the problem in time $(1/\varepsilon + |\mathcal{R}|)^{\mathcal{O}(1/\varepsilon)}$ by applying Lemma 4 to every k -tuple of rectangles in \mathcal{R} .

Let \mathcal{W} be the set of rectangles of \mathcal{R} that have width at most $N_1/(\delta k^2)$, and let $w = |\mathcal{W}|$. Note that since all rectangles are wide, the rectangles of \mathcal{W} also have height bounded by $N_1/(\delta k^2)$. If $w \geq k$, then we can immediately construct a packing of size k by stacking any k rectangles of \mathcal{W} vertically: they fit in the vertical dimension, because $k \cdot N_1/(\delta k^2) \leq N_2$. Otherwise, let $k' = k - w$. Run \mathcal{A} on a modified instance where all rectangles of \mathcal{W} are removed, with parameter k' . If there is no packing of size k' for this instance, then clearly there is no packing of size k for the original instance, and this conclusion may be reported by the algorithm. Otherwise, \mathcal{B} returns a packing \mathcal{S}' of size at least $(1 - \varepsilon)k'$ consisting of rectangles from $\mathcal{R} \setminus \mathcal{W}$. If \mathcal{S}' consists only of rectangles of height at most $N_1/(\delta k)$, then we can again immediately obtain a packing of size k by stacking the rectangles of $\mathcal{S}' \cup \mathcal{W}$ vertically; again they fit in the vertical dimension, because $k \cdot N_1/(\delta k) \leq N_2$. Otherwise,

we modify \mathcal{S}' by removing any single rectangle R present in \mathcal{S}' whose height (and therefore also width) is at least $N_1/(\delta k)$, and putting all the rectangles of \mathcal{W} into the space freed by the removal of R , by simply stacking them horizontally. They fit horizontally because $w \cdot N_1/(\delta k^2) \leq k \cdot N_1/(\delta k^2) = N_1/(\delta k) \leq w(R)$, and their heights are not greater than the height of R . The obtained modified packing \mathcal{S}' is returned by the algorithm.

It is clear that the algorithm outputs a packing and that when it concludes that there is no packing of size k , this conclusion is correct. What remains to show is that the packing eventually output by the algorithm has always size at least $(1 - 2\varepsilon)k$. And indeed, the algorithm always is able to pack all rectangles packed in \mathcal{S}' , except for possibly one rectangle removed to accommodate \mathcal{W} , and all rectangles of \mathcal{W} . Hence, the packing output by the algorithm has always size at least

$$(1 - \varepsilon)k' - 1 + w = (1 - \varepsilon)k - (1 - \varepsilon)w - 1 + w \geq (1 - \varepsilon)k - 1 > (1 - 2\varepsilon)k,$$

because $\varepsilon k > 1$ due to $k > 1/\varepsilon$. ◀

Now, Theorem 1 follows immediately by combining the algorithm of Lemma 27 with the reduction of Lemma 28. Observe that the running time is $\delta(B)^{\mathcal{O}(k)} \cdot (k + 1/\varepsilon)^{\mathcal{O}(k+1/\varepsilon^2)}$. $(|\mathcal{R}|||B||)^{\mathcal{O}(1/\varepsilon^2)}$, as promised.

6 Conclusion

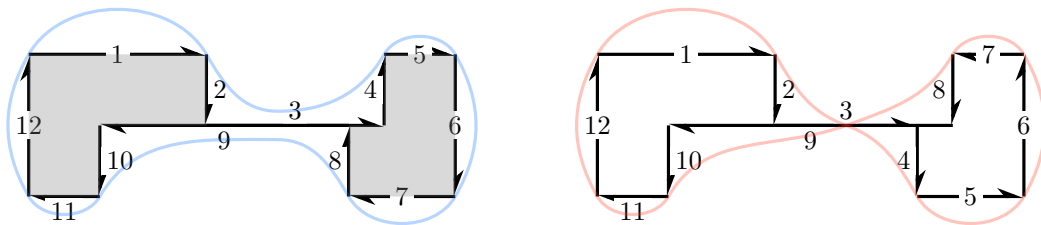
The correctness of our entire algorithm heavily relies on the assumption that every input rectangle is wide. Indeed, this assumption is used in the greedy arguments in the proof of Lemma 28, which allows us to reduce to the case when every rectangle has a substantial width: at least $N_1/\text{poly}(\delta(B), k)$. This assumption is again heavily used later on: in the proof of Lemma 9 it ensures that upon removing the rectangles corresponding to an st -separator in the conflict graph, there is enough space available for vertical shifting. This eventually leads to rounding the rectangles so that there are only $\text{poly}(\delta(B), k)$ different possible widths, and thus effectively bounding the number of candidate rectangles to $\text{poly}(\delta(B), k)$. So while the original problem – the existence of a parameterized approximation scheme for 2D KNAPSACK – remains open, we hope that the new structural techniques proposed in this work might give insight leading to its resolution.

References

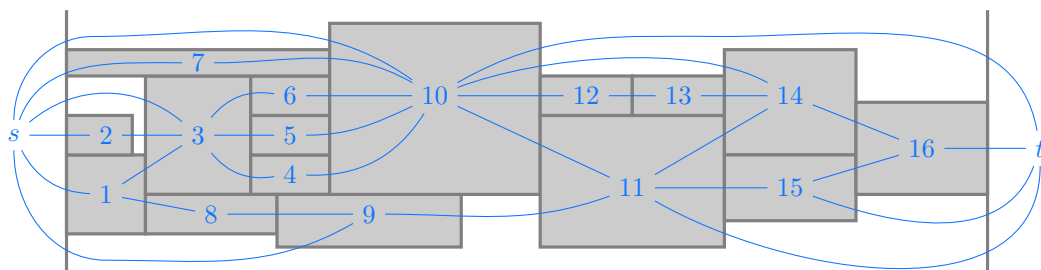
- 1 Anna Adamaszek and Andreas Wiese. A quasi-PTAS for the two-dimensional geometric knapsack problem. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 1491–1505. SIAM, 2015. doi:10.1137/1.9781611973730.98.
- 2 Hsien-Chih Chang, Jeff Erickson, and Chao Xu. Detecting weakly simple polygons. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 1655–1670. SIAM, 2015. doi:10.1137/1.9781611973730.110.
- 3 Erik D Demaine and Joseph O'Rourke. *Geometric folding algorithms: linkages, origami, polyhedra*. Cambridge university press, 2007.
- 4 Manfredo P Do Carmo. *Differential geometry of curves and surfaces: revised and updated second edition*. Courier Dover Publications, 2016.
- 5 Waldo Gálvez, Fabrizio Grandoni, Salvatore Ingala, Sandy Heydrich, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via L-packings. *ACM Trans. Algorithms*, 17(4), 2021. doi:10.1145/3473713.

- 6 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. In *37th International Symposium on Computational Geometry, SoCG 2021*, volume 189 of *LIPIcs*, pages 39:1–39:17. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.SocG.2021.39.
- 7 Fabrizio Grandoni, Stefan Kratsch, and Andreas Wiese. Parameterized approximation schemes for Independent Set of Rectangles and Geometric Knapsack. In *27th Annual European Symposium on Algorithms, ESA 2019*, volume 144 of *LIPIcs*, pages 53:1–53:16. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.53.
- 8 Klaus Jansen and Guochuan Zhang. On rectangle packing: maximizing benefits. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004*, pages 204–213. SIAM, 2004. URL: <http://dl.acm.org/citation.cfm?id=982792.982822>.
- 9 Yoshiyuki Kusakari, Hitoshi Suzuki, and Takao Nishizeki. A shortest pair of paths on the plane with obstacles and crossing areas. *International Journal of Computational Geometry & Applications*, 9(02):151–170, 1999.
- 10 Joseph Y.-T. Leung, Tommy W. Tam, C. S. Wong, Gilbert H. Young, and Francis Y. L. Chin. Packing squares into a square. *J. Parallel Distributed Comput.*, 10(3):271–275, 1990. doi:10.1016/0743-7315(90)90019-L.
- 11 Moni Naor, Leonard J Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 182–191. IEEE, 1995.

A Figures

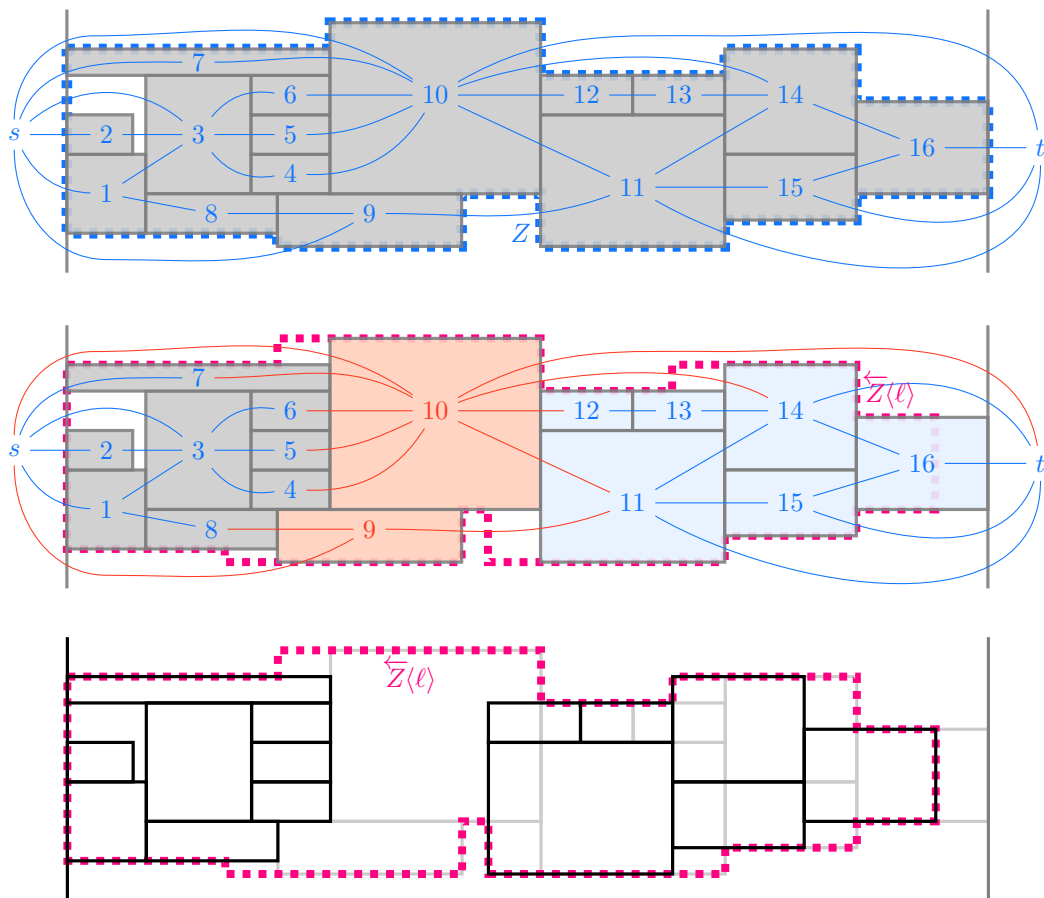


■ **Figure 1** Left panel: an example of a container where the order of the segments is given by the numbers, and some injective candidate γ_ϵ in light blue. Right panel: an example of a non-container (the paths cross in the middle), and some candidate γ_ϵ in light red, that is not injective.

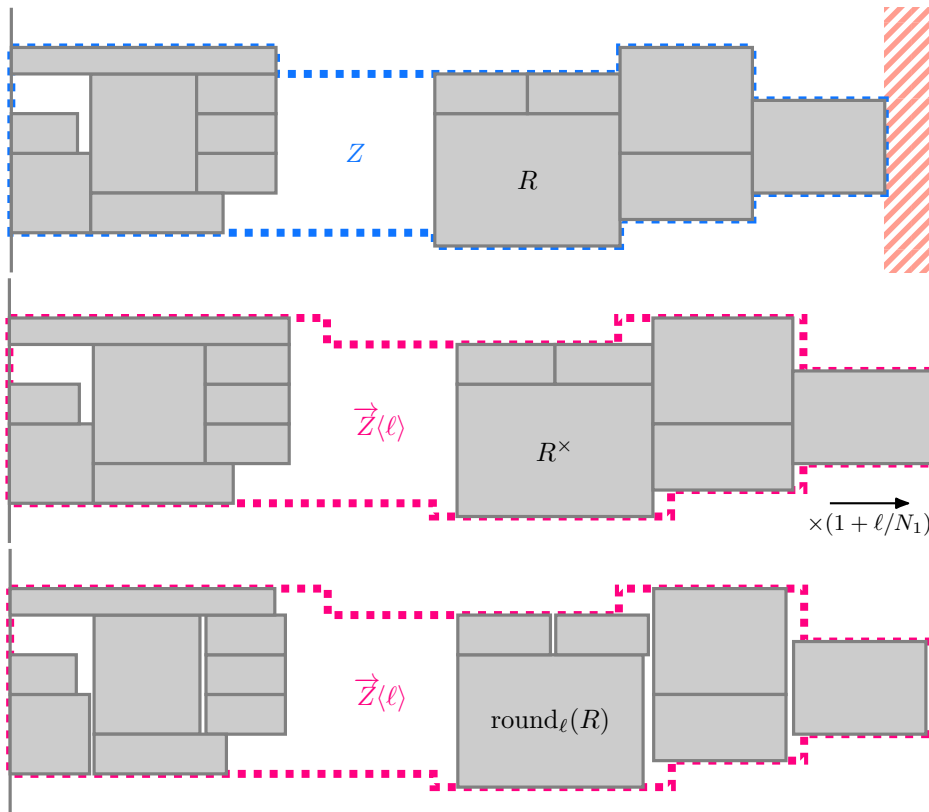


■ **Figure 2** Example of a conflict graph.

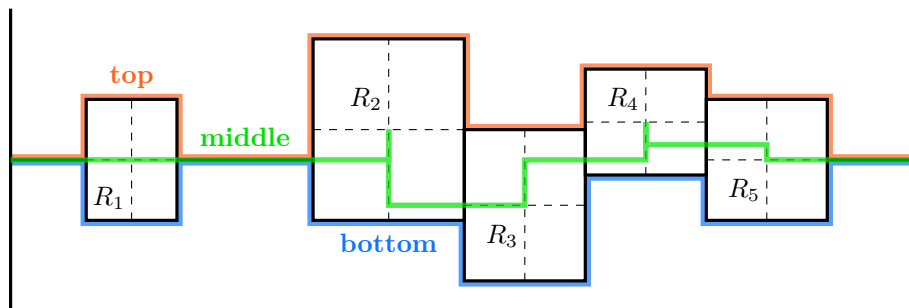
33:16 Parameterized Approximation Scheme for Geometric Knapsack with Wide Items



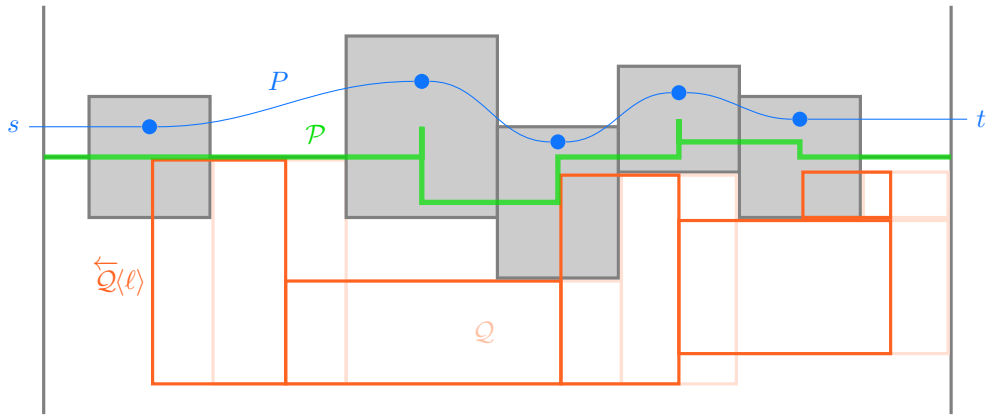
■ **Figure 3** Illustration of the proof of Proposition 15. From top to bottom: First, a blue conflict graph of the gray packing. The packing is entirely in Z , delimited by the blue border. Secondly, we remove the orange separator C and want to pack the leftover rectangles inside the red region $\overline{Z}(\ell)$. The set of rectangles referred to as Y in the proof is in light blue, and X is left gray. Finally, we pack in $\overline{Z}(\ell)$ by shifting the rectangles at the right of the separator by ℓ to the left.



■ **Figure 4** Illustration of the proof of Proposition 16. From top to bottom: First, \mathcal{Q} is packed into Z (blue zone). Then, the rectangles in \mathcal{Q} are scaled horizontally by a factor $\lambda = 1 + \ell/N_1$. We argue in the proof that these scaled-up rectangles are packed in $\vec{Z}(\ell)$ (red zone). Finally, we replace each scaled-up rectangle by its rounded version, which has smaller width.



■ **Figure 5** In orange, the top polyline of the st -path formed by R_1, R_2, R_3, R_4 and R_5 . In blue, its bottom polyline, and in green, its middle polyline. The dashed lines split their respective rectangles into 4 equal parts.



■ **Figure 6** In green, the middle polyline \mathcal{P} of the blue st -path P constituted of the gray rectangles. In light orange, \mathcal{Q} , and in orange, $\overleftarrow{\mathcal{Q}}(\ell)$. Notice that the green polyline does not cross any orange rectangles.

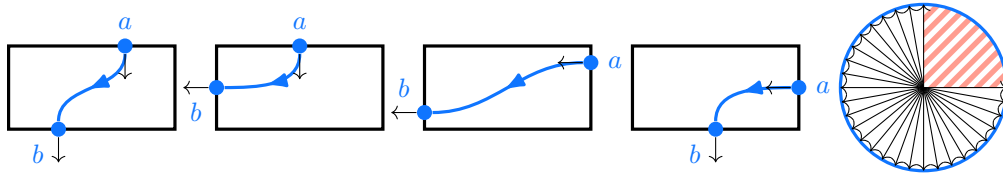
B Omitted proofs

Proof of Proposition 6. Create a directed graph D with vertex set \mathcal{R} , where there is an edge (R, R') if the bottom side of R intersects the top side of R' on more than a single point, or if the left side of R intersects the right side of R' on more than a single point. Let us show that D has no directed cycle, so for contradiction suppose R_1, \dots, R_ℓ is a directed cycle in D . In what follows, all indices behave cyclically modulo ℓ .

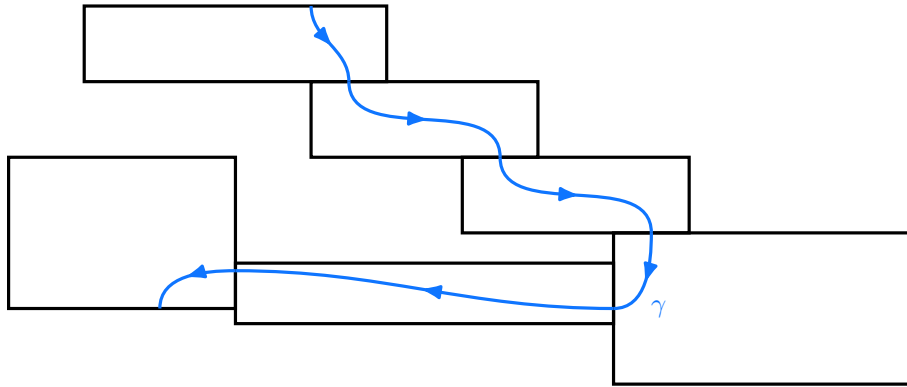
For each $i \in [\ell]$, select an arbitrary point p_i in the intersection of R_{i-1} and R_i that is neither a corner of R_{i-1} nor a corner of R_i . Further, observe that one can construct a curve $\gamma_i: [0, d_i] \rightarrow R_i$, where d_i is the length of γ_i , such that:

- γ_i is smooth (formally, C^1) and monotone in both directions,
- $\|\gamma'_i(t)\|_2 = 1$ for all $t \in [0, d_i]$,
- $\gamma_i(0) = p_i$ and $\gamma_i(d_i) = p_{i+1}$, and
- the tangent of γ_i at p_i and p_{i+1} is perpendicular to the respective side and faces the inside (resp. outside) of R_i . For instance if p_i is on the top side of R_i , we require $\gamma'_i(0) = (0, -1)$, and if p_{i+1} is on the left side of R_i , we require $\gamma'_i(d_i) = (-1, 0)$.

An example of such a construction is shown below.



Concatenating all the curves γ_i in order yields a smooth closed curve $\gamma: S \rightarrow \bigcup_{i=1}^{\ell} R_i$ without self-crossings such that $\|\gamma'(t)\|_2 = 1$ for all $t \in S$, where S is the circle of length $\sum_{i=1}^{\ell} d_i$. Here is the crucial observation: by the way we oriented the arcs in D , the vector $\gamma'(t)$ is never in the positive orthant (i.e. $\gamma'(t)$ has not both coordinates positive), for any $t \in S$.



However by Theorem 2 of [4, section 5-7, page 402], a smooth closed curve in the plane without self-crossings has rotation index ± 1 , where the rotation index of a curve is the number of times its tangent vector turns around the origin. This means that by the intermediate value theorem, for every $\alpha \in]0, 2\pi[$, there exists a point of the curve where the tangent vector is at angle α with the x -axis, and hence belongs to the positive orthant. This is a contradiction.

We conclude that D is acyclic, hence it has a sink R – a rectangle with out-degree 0. Therefore, R is the rectangle we want: its left side intersects a vertical segment of container and its bottom side intersects a horizontal segment of the container. ◀

Proof of Lemma 12. Let G be the conflict graph of \mathcal{Q} . For a rectangle $R \in \mathcal{Q}$, we denote its associated vertex in G by v_R . We define a planar embedding of G as follows. We define the position of a vertex v_R to be the center $c(R) = (x(R) + w(R)/2, y(R) + h(R)/2)$ of the corresponding rectangle. If there is an edge $e = v_R v_{R'}$, choose $y \in \mathbb{R}$ such that the horizontal segment $s = [x(R) + w(R), x(R')] \times \{y\}$ witnesses that R and R' that see each other, where we assume w.l.o.g. that $x(R) + w(R) \leq x(R')$. We define the embedding γ_e of e as the union of 3 internally disjoint segments:

- $s_e^1 = [c(R), (x(R) + w(R), y)]$,
- $s_e^2 = s$,
- $s_e^3 = [(x(R'), y), c(R')]$.

It is straightforward to check that all the curves γ_e $e \in E(G)$ are pairwise internally disjoint, hence they constitute a planar embedding of G . ◀

Proof of Proposition 15. Let G be the conflict graph of \mathcal{Q} . Since C is an st -separator in G , we may partition \mathcal{Q} into three disjoint sets X , C , and Y so that vertices of X are not connected to t , vertices of Y are not connected to s , and no vertex of X is adjacent to any vertex of Y . Now, construct a new set of placed rectangles \mathcal{Q}' by removing all rectangles of C and shifting every rectangle of Y by ℓ to the left. It remains to prove that \mathcal{Q}' is a packing and that all rectangles of \mathcal{Q}' are entirely contained in $\overleftarrow{Z}(\ell)$.

For the second assertion, we need to prove that (i) no $R' \in \mathcal{Q}'$ crosses the left side of the box, i.e., no $R' \in \mathcal{Q}'$ is such that $x(R') < 0$, and (ii) no rectangle $R' \in \mathcal{Q}'$ contains a point with horizontal coordinate larger than $N_2 - \ell$, i.e. $x(R') + w(R') > N_2 - \ell$. To prove (i), suppose for the sake of a contradiction that there exists $R' \in Y$ such that $x(R') < 0$. We must have $R' \in Y$ because R' was shifted, and hence R' cannot see the left side of the box. Let $R \in \mathcal{Q}$ be R' before shifting. We know that $x(R) < \ell$, therefore as every rectangle has width at least ℓ , there is no rectangle in \mathcal{Q} that would be placed between R and the left side of the box. Therefore, the R must see the left side of the box, which is a contradiction because $R \in Y$. A symmetric argument involving the right side of the box proves (ii).

For the first assertion, we need to prove that no two rectangles in \mathcal{Q}' overlap. The only case when this could a priori happen is if $R_1 \in X$ and $R_2 \in Y$ are overlapping after the shift. This would mean that $x(R_1) + w(R_1) < x(R_2) - \ell$. However, again in \mathcal{Q} there cannot be any rectangle lying in between R_1 and R_2 , because every rectangle has width at least ℓ . Therefore, the R_1 and R_2 must see each other, which is a contradiction because $R_1 \in X$ and $R_2 \in Y$. ◀

Proof of Proposition 16. First, scale horizontally every rectangle in \mathcal{Q} by a factor $\lambda = 1 + \ell/N_1$, i.e., for a rectangle $R = [x, x + w] \times [y, y + h]$ we define the rectangle $R^\times = [\lambda x, \lambda(x + w)] \times [y, y + h]$. These rectangles fit inside $\text{round}_\ell(Z)$. Indeed, the maximum possible displacement of a point is $N_1 \cdot \ell/N_1 = \ell$, i.e. the image of a point under scaling is at horizontal distance at most ℓ to the right of the original point. Next, observe that every rectangle $\text{round}_\ell(R)$ can be entirely placed inside the corresponding rectangle R^\times , because $\lambda w = w + w\ell/N_1 \geq w + \ell^2/N_1 = \ell' + w = \ell'(1 + w/\ell') \geq \ell' \lceil w/\ell' \rceil$. (Recall here that we assumed all rectangles to have width at least ℓ .) Now, \mathcal{Q}' can be obtained from \mathcal{Q} by replacing each $R \in \mathcal{Q}'$ with R^\times , fitting $\text{round}_\ell(R)$ inside R^\times , and finally shifting all rectangles to the left so that they have integer coordinates. The last step is always possible as every rectangle has integer length. ◀

Proof of Proposition 20. Suppose \mathcal{P} crosses $\overleftarrow{R}(\ell)$ for some $R \in \mathcal{Q}$. Then there exists $R_1, R_2 \in V(P)$ (which are possibly the left or the right side of the box) such that R_1 and R_2 see each other through a segment $s = [x(R_1) + w(R_1), x(R_2)] \times \{y\}$ and R crosses one of the following segments:

1. $\{x(R_1) + w(R_1)/2\} \times [\min\{y(R_1) + h(R_1)/2, y\}, \max\{y(R_1) + h(R_1)/2, y\}]$,
2. $[x(R_1) + w(R_1)/2, x(R_2) + w(R_2)/2] \times \{y\}$,
3. $\{x(R_2) + w(R_2)/2\} \times [\min\{y, y(R_2) + h(R_2)/2\}, \max\{y, y(R_2) + h(R_2)/2\}]$.

We show that every case leads to a contradiction.

1. Assume case 1. \mathcal{P} crosses $\overleftarrow{R}(\ell)$ but not R so $x(R) \geq x(R_1) + w(R_1)/2$ and $x(R) - \ell \leq x(R_1) + w(R_1)/2$. Therefore $x(R_1) \leq x(R) \leq x(R_1) + w(R_1)/2 + \ell \leq x(R_1) + w(R_1)$ because $w(R_1) \geq 2\ell$. Moreover, $[\min\{y(R_1) + h(R_1)/2, y\}, \max\{y(R_1) + h(R_1)/2, y\}] \subseteq [y(R_1), y(R_1) + h(R_1)]$ by the definition of y . This means that R and R_1 intersect at $(x(R), y')$ where $y' \in [y(R), y(R) + h(R)] \cap [\min\{y(R_1) + h(R_1)/2, y\}, \max\{y(R_1) + h(R_1)/2, y\}]$, which is not possible.
2. Assume case 2. This would mean that $y \in [y(R), y(R) + h(R)]$, $x(R) \geq x(R_2) + w(R_2)/2$ and $x(R) - \ell \leq x(R_2) + w(R_2)/2$ because $|[x(R_1) + w(R_1)/2, x(R_2) + w(R_2)/2]| \geq 2\ell$ and \mathcal{P} crosses $\overleftarrow{R}(\ell)$ but not R . Therefore $x(R_2) \leq x(R) \leq x(R_2) + w(R_2)/2 + \ell \leq x(R_2) + w(R_2)$ because $w(R_2) \geq 2\ell$. This means that R and R_2 intersect at $(x(R), y)$, which is not possible.
3. Assume case 3. This is a similar argument as case 1, replacing R_1 by R_2 . ◀

Proof of Lemma 25. Let \mathcal{Q} be the assumed packing of $k' \leq k$ rectangles from \mathcal{R} of pairwise different colors in the zone Z . Note that if \mathcal{Q} contains some rectangle of $R \in \mathcal{R} \setminus \text{reduce}_k(\mathcal{R}, \text{col})$, then there exists another rectangle $R' \in \text{reduce}_k(\mathcal{R}, \text{col})$ with $w(R') = w(R)$, $\text{col}(R') = \text{col}(R)$ and $h(R') \leq h(R)$ such that R' was not used in the packing \mathcal{Q} . Hence, we can substitute R with R' in the packing \mathcal{Q} , fitting R' within the area freed by removing R from the packing. By applying such substitutions exhaustively, we obtain a packing in Z consisting of k' rectangles from $\text{reduce}_k(\mathcal{R}, \text{col})$. ◀

A Contraction-Recursive Algorithm for Treewidth

Hisao Tamaki  

Meiji University, Kawasaki, Japan

Abstract

Let $\text{tw}(G)$ denote the treewidth of graph G . Given a graph G and a positive integer k such that $\text{tw}(G) \leq k + 1$, we are to decide if $\text{tw}(G) \leq k$. We give a certifying algorithm RTW (“R” for recursive) for this task: it returns one or more tree-decompositions of G of width $\leq k$ if the answer is YES and a minimal contraction H of G such that $\text{tw}(H) > k$ otherwise. Starting from a greedy upper bound on $\text{tw}(G)$ and repeatedly improving the upper bound by this algorithm, we obtain $\text{tw}(G)$ with certificates.

RTW uses a heuristic variant of Tamaki’s PID algorithm for treewidth (ESA2017), which we call HPID. Informally speaking, PID builds potential subtrees of tree-decompositions of width $\leq k$ in a bottom up manner, until such a tree-decomposition is constructed or the set of potential subtrees is exhausted without success. HPID uses the same method of generating a new subtree from existing ones but with a different generation order which is not intended for exhaustion but for quick generation of a full tree-decomposition when possible. RTW, given G and k , interleaves the execution of HPID with recursive calls on G/e for edges e of G , where G/e denotes the graph obtained from G by contracting edge e . If we find that $\text{tw}(G/e) > k$, then we have $\text{tw}(G) > k$ with the same certificate. If we find that $\text{tw}(G/e) \leq k$, we “uncontract” the bags of the certifying tree-decompositions of G/e into bags of G and feed them to HPID to help progress. If the question is not resolved after the recursive calls are made for all edges, we finish HPID in an exhaustive mode. If it turns out that $\text{tw}(G) > k$, then G is a certificate for $\text{tw}(G') > k$ for every G' of which G is a contraction, because we have found $\text{tw}(G/e) \leq k$ for every edge e of G . This final round of HPID guarantees the correctness of the algorithm, while its practical efficiency derives from our methods of “uncontracting” bags of tree-decompositions of G/e to useful bags of G , as well as of exploiting those bags in HPID.

Experiments show that our algorithm drastically extends the scope of practically solvable instances. In particular, when applied to the 100 instances in the PACE 2017 bonus set, the number of instances solved by our implementation on a typical laptop, with the timeout of 100, 1000, and 10000 seconds per instance, are 72, 92, and 98 respectively, while these numbers are 11, 38, and 68 for Tamaki’s PID solver and 65, 82, and 85 for his new solver (SEA 2022).

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases graph algorithm, treewidth, exact computation, BT dynamic programming, contraction, certifying algorithms

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.34

Related Version *Full Version:* <https://arxiv.org/abs/2307.01318>

Supplementary Material *Software (Source Code):* <https://github.com/twalgor/RTW>

1 Introduction

Treewidth is a graph parameter introduced and extensively studied in the graph minor theory [14]. A *tree-decomposition* of graph G is a tree with each node labeled by a vertex set of G , called a *bag*, satisfying certain conditions (see Section 2) so that those bags form a tree-structured system of vertex-separators of G . The width $w(T)$ of a tree-decomposition T is the maximum cardinality of a bag in T minus one and the treewidth $\text{tw}(G)$ of graph G is the smallest k such that there is a tree-decomposition of G of width k .



© Hisao Tamaki;

licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 34; pp. 34:1–34:15

Leibniz International Proceedings in Informatics



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

The impact of the notion of treewidth on the design of combinatorial algorithms is profound: there are a huge number of NP-hard graph problems that are known to be tractable when parameterized by treewidth: they admit an algorithm with running time $f(k)n^{O(1)}$, where n is the number of vertices, k is the treewidth of the given graph, and f is some typically exponential function (see [10], for example). Those algorithms typically perform dynamic programming based on the system of separators provided by the tree-decomposition. To make such algorithms practically useful, we need to compute the treewidth, or a good approximation of the treewidth, together with an associated tree-decomposition.

Computing the treewidth $\text{tw}(G)$ of a given graph G is NP-complete [2], but is fixed-parameter tractable [14, 4]. In particular, the algorithm due to Bodlaender [4] runs in time linear in the graph size with a factor of $2^{O(\text{tw}(G)^3)}$. Unfortunately, this algorithm does not seem to run efficiently in practice.

In more practical approaches to treewidth computation, triangulations of graphs play an important role. A *triangulation* of graph G is a chordal graph H with $V(G) = V(H)$ and $E(G) \subseteq E(H)$. For every tree-decomposition T of G , filling every bag of T into a clique gives a triangulation of G . Conversely, for every triangulation H of G , there is a tree-decomposition of G in which every bag is a maximal clique of H . Through this characterization of tree-decompositions in terms of triangulations, we can enumerate all relevant tree-decompositions by going through the total orderings on the vertex set, as each total ordering defines a triangulation for which the ordering is a perfect elimination order (see [7], for example). Practical algorithms in the early stage of treewidth research performed a branch-and-bound search over these total orderings [7]. Dynamic programming on this search space results in a $2^n n^{O(1)}$ time algorithms [5], which works well in practice for graphs with a small number of vertices. It should also be noted that classical upper bound algorithms, such as min-deg or min-fill, which heuristically choose a single vertex ordering defining a tree-decomposition, are fast and often give a good approximation of the treewidth in a practical sense [7].

Another important link between chordal graphs and treewidth computation was established by Bouchitté and Todinca [9]. They introduced the notion of potential maximal cliques (PMCs, see below in "Our approach" paragraph for a definition) and gave an efficient dynamic programming algorithm working on PMCs (BT dynamic programming) to find a minimal triangulation of the given graph that corresponds to an optimal tree-decomposition. They showed that their algorithm runs in polynomial time for many special classes of graphs. BT dynamic programming is also used in an exponential time algorithm for treewidth that runs in time $O(1.7549^n)$ [12].

BT dynamic programming had been considered mostly of theoretical interest until 2017, when Tamaki presented its positive-instance driven (PID) variant, which runs fast in practice and significantly outperforms previously implemented treewidth algorithms [18]. Further efforts on treewidth computation based on or around his approach have been made since then, with some incremental successes [17, 16, 19, 1].

In his most recent work [20], Tamaki introduced another approach to treewidth computation, based on the use of contractions to compute tight lower bounds on the treewidth. For edge e of graph G , the *contraction* of G by e , denoted by G/e , is a graph obtained from G by replacing e by a new single vertex v_e and let v_e be adjacent to all neighbors of the ends of e in $V(G) \setminus e$. A graph H is a *contraction* of G if H is obtained from G by zero or more successive contractions by edges. It is well-known and easy to see that $\text{tw}(H) \leq \text{tw}(G)$ for every contraction H of G . This fact has been used to quickly compute reasonably good lower bounds on the treewidth of a graph, typically to be used in branch-and-bound algorithms mentioned above [7, 8]. Tamaki [20] gave a heuristic method of successively improving

contraction based lower bounds which, together with a separate heuristic method for upper bounds, quite often succeeds in computing the exact treewidth of instances that are hard to solve for previously published solvers.

Our approach

Our approach is based on the observation that contractions are useful not only for computing lower bounds but also for computing upper bounds. Suppose we have a tree-decomposition T of G/e of width k for some edge $e = \{u, v\}$ of G . Let v_e be the vertex to which e contracts. Replacing each bag X of T by X' , where $X' = X \setminus \{v_e\} \cup \{u, v\}$ if $v_e \in X$ and $X' = X$ otherwise, we obtain a tree-decomposition T' of G of width $\leq k + 1$, which we call the *uncontraction* of T . In a fortunate case where every bag X of T with $v_e \in X$ has $|X| \leq k$, the width of T' is k . To increase the chance of having such fortunate cases, we deal with a set of tree-decompositions rather than a single tree-decomposition. We represent such a set of tree-decompositions by a set of potential maximal cliques as follows.

A vertex set of G is a *potential maximal clique* (PMC for short) if it is a maximal clique of some minimal triangulation of G . Let $\Pi(G)$ denote the set of all PMCs of G . For each $\Pi \subseteq \Pi(G)$, let $\mathcal{T}_\Pi(G)$ denote the set of all tree-decompositions of G whose bags all belong to Π . Let $\text{tw}_\Pi(G)$ denote the smallest k such that there is a tree-decomposition in $\mathcal{T}_\Pi(G)$ of width k ; we set $\text{tw}_\Pi(G) = \infty$ if $\mathcal{T}_\Pi(G) = \emptyset$. Bouchitté and Todinca [9] showed that $\mathcal{T}_{\Pi(G)}(G)$ contains a tree-decomposition of width $\text{tw}(G)$ and developed a dynamic programming algorithm (BT dynamic programming) to find such a tree-decomposition. Indeed, as Tamaki [17] noted, BT dynamic programming can be used for arbitrary $\Pi \subseteq \Pi(G)$ to compute $\text{tw}_\Pi(G)$ in time linear in $|\Pi|$ and polynomial in $|V(G)|$.

A set of PMCs is a particularly effective representation of a set of tree-decompositions for our purposes, because BT dynamic programming can be used to work on $\Pi \subseteq \Pi(G)$ and find a tree-decomposition in $\mathcal{T}_\Pi(G)$ that minimizes a variety of width measures based on bag weights. In our situation, suppose we have $\Pi \subseteq \Pi(G/e)$ such that $\text{tw}_\Pi(G/e) = k$. Using appropriate bag weights, we can use BT dynamic programming to decide if $\mathcal{T}_\Pi(G/e)$ contains T such that the uncontraction T' of T has width k and find one if it exists.

These observations suggest a recursive algorithm for improving an upper bound on treewidth. Given a graph G and k such that $\text{tw}(G) \leq k + 1$, the task is to decide if $\text{tw}(G) \leq k$. Our algorithm certifies the YES answer by $\Pi \subseteq \Pi(G)$ with $\text{tw}_\Pi(G) \leq k$. It uses heuristic methods to find such Π and, when this goal is hard to achieve, recursively solves the question if $\text{tw}(G/e) \leq k$ for edge e of G . Unless $\text{tw}(G/e) = k + 1$ and hence $\text{tw}(G) = k + 1$, the recursive call returns $\Pi \subseteq \Pi(G/e)$ such that $\text{tw}_\Pi(G/e) \leq k$. We use the method mentioned above to look for $T \in \mathcal{T}_\Pi(G/e)$ whose uncontraction has width $\leq k$. If we are successful, we are done for G . Even when this is not the case, the uncontractions of tree-decompositions in $\mathcal{T}_\Pi(G/e)$ may be useful for our heuristic upper bound method in the following manner.

In [17], Tamaki proposed a local search algorithm for treewidth in which a solution is a set of PMCs rather than an individual tree-decomposition and introduced several methods of expanding $\Pi \subseteq \Pi(G)$ into $\Pi' \supset \Pi$ in hope of having $\text{tw}_{\Pi'}(G) < \text{tw}_\Pi(G)$. His method compares favourably with existing heuristic algorithms but, like typical local search methods, is prone to local optima. To let the search escape from a local optimum, we would like to inject “good” PMCs to the current set Π . It appears that tree-decompositions in $\mathcal{T}_{\Pi'}(G/e)$ such that $\text{tw}_{\Pi'}(G/e) \leq k$, where $k = \text{tw}_\Pi(G) - 1$, are reasonable sources of such good PMCs: we uncontract $T \in \mathcal{T}_{\Pi'}(G/e)$ into a tree-decomposition T' of G and extract PMCs of G from T' . Each such PMC appears in a tree-decomposition of width $\leq k + 1$ and may appear in a tree-decomposition of width $\leq k$. It is also important that Π' is obtained, in a loose sense, independently of Π and not under the influence of the local optimum around which Π stays.

Our algorithm for deciding if $\text{tw}(G) \leq k$ interleaves the execution of a local search algorithm with recursive calls on G/e for edges e of G and injects PMCs obtained from the results of the recursive calls. This process ends in either of the following three ways.

1. The local search succeeds in finding Π with $\text{tw}_\Pi(G) \leq k$.
2. A recursive call on G/e finds that $\text{tw}(G/e) = k + 1$: we conclude that $\text{tw}(G) = k + 1$ on the spot.
3. Recursive calls G/e have been tried for all edges e and it is still unknown if $\text{tw}(G) \leq k$.

We invoke a conventional exact algorithm for treewidth to settle the question.

Note that, when the algorithm concludes that $\text{tw}(G) = k + 1$, there must be a contraction H of G somewhere down in the recursion path from G such that Case 3 applies and the exact computation shows that $\text{tw}(H) = k + 1$. In this case, H is a minimal contraction of G that certifies $\text{tw}(G) = k + 1$, as the recursive calls further down from H have shown $\text{tw}(H/e) \leq k$ for every edge e of H .

As the experiments in Section 11 show, this approach drastically extends the scope of instances for which the exact treewidth can be computed in practice.

Organization

The rest of this paper is organized as follows. After the preliminaries in Section 2, the main algorithm in its basic form is described in Section 3. Sections 4, 6, 7, 8, 9, and 10 describe some details of the techniques used to make the algorithm run fast in practice. Section 11 presents experimental results and Section 12 offers some concluding remarks.

The source code of the implementation of our algorithm used in the experiments is available at <https://github.com/twalgor/RTW>.

2 Preliminaries

Graphs and treewidth

In this paper, all graphs are simple, that is, without self loops or parallel edges. Let G be a graph. We denote by $V(G)$ the vertex set of G and by $E(G)$ the edge set of G . As G is simple, each edge of G is a subset of $V(G)$ with exactly two members that are adjacent to each other in G . The *complete graph* on V , denoted by $K(V)$, is a graph with vertex set V in which every vertex is adjacent to all other vertices. The subgraph of G induced by $U \subseteq V(G)$ is denoted by $G[U]$. We sometimes use an abbreviation $G \setminus U$ to stand for $G[V(G) \setminus U]$. A vertex set $C \subseteq V(G)$ is a *clique* of G if $G[C]$ is a complete graph. For each $v \in V(G)$, $N_G(v)$ denotes the set of neighbors of v in G : $N_G(v) = \{u \in V(G) \mid \{u, v\} \in E(G)\}$. For $U \subseteq V(G)$, the *open neighborhood of U in G* , denoted by $N_G(U)$, is the set of vertices adjacent to some vertex in U but not belonging to U itself: $N_G(U) = (\bigcup_{v \in U} N_G(v)) \setminus U$.

We say that vertex set $C \subseteq V(G)$ is *connected in G* if, for every $u, v \in C$, there is a path in $G[C]$ between u and v . It is a *connected component* or simply a *component* of G if it is connected and is inclusion-wise maximal subject to this condition. We denote by $\mathcal{C}(G)$ the set of all components of G . When the graph G is clear from the context, we denote $\mathcal{C}(G[U])$ by $\mathcal{C}(U)$. A vertex set $S \subseteq V(G)$ is a *separator* of G if $G \setminus S$ has more than one component. A graph is a *cycle* if it is connected and every vertex is adjacent to exactly two vertices. A graph is a *forest* if it does not have a cycle as a subgraph. A forest is a *tree* if it is connected.

A *tree-decomposition* of G is a pair (T, \mathcal{X}) where T is a tree and \mathcal{X} is a family $\{X_i\}_{i \in V(T)}$ of vertex sets of G , indexed by the nodes of T , such that the following three conditions are satisfied. We call each X_i the *bag* at node i .

1. $\bigcup_{i \in V(T)} X_i = V(G)$.
 2. For each edge $\{u, v\} \in E(G)$, there is some $i \in V(T)$ such that $u, v \in X_i$.
 3. For each $v \in V(G)$, the set of nodes $I_v = \{i \in V(T) \mid v \in X_i\} \subseteq V(T)$ is connected in T .
- The *width* of this tree-decomposition is $\max_{i \in V(T)} |X_i| - 1$. The *treewidth* of G , denoted by $\text{tw}(G)$ is the smallest k such that there is a tree-decomposition of G of width k .

For each pair (i, j) of adjacent nodes of a tree-decomposition (T, \mathcal{X}) of G , let $T(i, j)$ denote the subtree of T consisting of nodes of T reachable from i without passing j and let $V(i, j) = \bigcup_{k \in V(T(i, j))} X_k$. Then, it is well-known and straightforward to show that $X_i \cap X_j = V(i, j) \cap V(j, i)$ and there are no edges between $V(i, j) \setminus V(j, i)$ and $V(j, i) \setminus V(i, j)$; $X_i \cap X_j$ is a separator of G unless $V(i, j) \subseteq V(j, i)$ or $V(j, i) \subseteq V(i, j)$. We say that T *uses* separator S if there is an adjacent pair (i, j) such that $S = X_i \cap X_j$. In this paper, we assume G is connected whenever we consider a tree-decomposition of G .

In this paper, most tree-decompositions are such that $X_i = X_j$ only if $i = j$. Because of this, we use a convention to view a tree-decomposition of G as a tree T whose nodes are bags (vertex sets) of G .

Triangulations, minimal separators, and Potential maximal cliques

Let G be a graph and S a separator of G . For distinct vertices $a, b \in V(G)$, S is an *a-b separator* if there is no path between a and b in $G \setminus S$; it is a *minimal a-b separator* if it is an *a-b separator* and no proper subset of S is an *a-b separator*. A separator is a *minimal separator* if it is a minimal *a-b separator* for some $a, b \in V(G)$.

Graph H is *chordal* if every induced cycle of H has exactly three vertices. H is a *triangulation of graph G* if it is chordal, $V(G) = V(H)$, and $E(G) \subseteq E(H)$. A triangulation H of G is *minimal* if there is no triangulation H' of G such that $E(H')$ is a proper subset of $E(H)$. It is known (see [13] for example) that if H is a minimal triangulation of G then every minimal separator of H is a minimal separator of G . In fact, the set of minimal separators of H is a maximal set of pairwise non-crossing minimal separators of G , where two separators S and R *cross each other* if at least two components of $G \setminus S$ intersects R .

Triangulations and tree-decompositions are closely related. For a tree-decomposition T of G , let $\text{fill}(G, T)$ denote the graph obtained from G by filling every bag of T into a clique. Then, it is straightforward to see that $\text{fill}(G, T)$ is a triangulation of G . Conversely, for each chordal graph H , consider a tree on the set \mathcal{K} of all maximal cliques of H such that if $X, Y \in \mathcal{K}$ are adjacent to each other then $X \cap Y$ is a minimal separator of H . Such a tree is called a *clique tree* of H . It is straightforward to verify that a clique tree T of a triangulation H of G is a tree-decomposition of G and that $\text{fill}(G, T) = H$.

We call a tree-decomposition T of G *minimal* if it is a clique tree of a minimal triangulation of G . It is clear that there is a minimal tree-decomposition of G of width $\text{tw}(G)$, since for every tree-decomposition T of G , there is a minimal triangulation H of G that is a subgraph of $\text{fill}(G, T)$ and every clique tree T' of H has $w(T') \leq w(T)$.

A vertex set $X \subseteq V(G)$ is a *potential maximal clique*, PMC for short, of G , if X is a maximal clique in some minimal triangulation of G . We denote by $\Pi(G)$ the set of all potential maximal cliques of G . By definition, every bag of a minimal tree-decomposition of G belongs to $\Pi(G)$.

Bouchitté-Todinca dynamic programming

For each $\Pi \subseteq \Pi(G)$, say that Π *admits* a tree-decomposition T of G if every bag of T belongs to Π . Let $\mathcal{T}_\Pi(G)$ denote the set of all tree-decompositions of G that Π admits and let $\text{tw}_\Pi(G)$ denote the smallest k such that there is $T \in \mathcal{T}_\Pi(G)$ of width k ; we set $\text{tw}_\Pi(G) = \infty$ if $\mathcal{T}_\Pi(G) = \emptyset$. The treewidth algorithm of Bouchitté and Todinca [9] is based on the observation that $\text{tw}(G) = \text{tw}_{\Pi(G)}(G)$. Given G , their algorithm first constructs $\Pi(G)$ and then search through $\mathcal{T}_{\Pi(G)}(G)$ by dynamic programming (BT dynamic programming) to find T of width $\text{tw}_{\Pi(G)}(G)$. As observed in [17], BT dynamic programming can be used to compute $\text{tw}_\Pi(G)$ for an arbitrary subset Π of $\Pi(G)$ to produce an upper bound on $\text{tw}(G)$. As we extensively use this idea, we describe how it works here.

Fix $\Pi \subseteq \Pi(G)$ such that $\mathcal{T}_\Pi(G)$ is non-empty. To formulate the recurrences in BT dynamic programming, we need some definitions. A vertex set B of G is a *block* if B is connected and either $N_G(B)$ is a minimal separator or is empty. As we are assuming that G is connected, $B = V(G)$ in the latter case. A *partial tree-decomposition* of a block B in G is a tree-decomposition of $G[B \cup N_G(B)]$ that has a bag containing $N_G(B)$, called the *root bag* of this partial tree-decomposition. Note that a partial tree-decomposition of block $V(G)$ is a tree-decomposition of G . For a graph G and a block B , let $\mathcal{P}_\Pi(B, G)$ denote the set of all partial tree-decompositions of B in G all of whose bags belong to Π and, when this set is non-empty, let $\text{tw}_\Pi(B, G)$ denote the smallest k such that there is $T \in \mathcal{P}_\Pi(B, G)$ with $w(T) = k$; if $\mathcal{P}_\Pi(B, G)$ is empty we set $\text{tw}_\Pi(B, G) = \infty$.

A PMC X of G is a *cap* of block B if $N_G(B) \subseteq X$ and $X \subseteq B \cup N_G(B)$. Note that a cap of B is a potential root bag of a partial tree-decomposition of B . For each block B , let $\mathcal{B}_\Pi(B)$ denote the set of all caps of B belonging to Π . Recall that, for each vertex set $U \subseteq V(G)$, $\mathcal{C}(U)$ denotes the set of components of $G[U]$. The following recurrence holds.

$$\text{tw}_\Pi(B, G) = \min_{X \in \mathcal{B}_\Pi(B)} \max\{|X| - 1, \max_{C \in \mathcal{C}(B \setminus X)} \text{tw}_\Pi(C, G)\} \quad (1)$$

BT dynamic programming evaluates this recurrence for blocks in the increasing order of cardinality and obtains $\text{tw}_\Pi(G) = \text{tw}_\Pi(V(G), G)$. Tracing back the recurrences, we obtain a tree-decomposition $T \in \mathcal{T}_\Pi(G)$ with $w(T) = \text{tw}_\Pi(G)$.

Tamaki's PID algorithm [18], unlike the original algorithm of Bouchitté and Todinca [9], does not construct $\Pi(G)$ before applying dynamic programming. It rather uses the above recurrence to generate relevant blocks and PMCs. More precisely, PID is for the decision problem whether $\text{tw}(G) \leq k$ for given G and k and it generates all blocks C with $\text{tw}(C, G) \leq k$ using the recurrence in a bottom up manner. We have $\text{tw}(G) \leq k$ if and only if $V(G)$ is among those generated blocks.

Contractors and contractions

To extend the notation G/e of a contraction by an edge to a contraction by multiple edges, we define contractors. A *contractor* γ of G is a partition of $V(G)$ into connected sets. For contractor γ of G , the contraction of G by γ , denoted by G/γ , is the graph obtained from G by contracting each part of γ to a single vertex, with the adjacency inherited from G . For notational convenience, we also view a contractor γ as a mapping from $V(G)$ to $\{1, 2, \dots, m\}$, the index set of the parts of the partition γ . In this view, the vertex set of G/γ is $\{1, 2, \dots, m\}$ and $\gamma(v)$ for each $v \in V(G)$ is the vertex of G/γ into which v is contracted. For each $w \in V(G/\gamma)$, $\gamma^{-1}(w)$ is the part of the partition γ that contracts to w . For $U \subseteq V(G/\gamma)$, we define $\gamma^{-1}(U) = \bigcup_{w \in U} \gamma^{-1}(w)$.

3 Main algorithm

The pseudo code in Algorithm 1 shows the main iteration of our treewidth algorithm. It starts from a greedy upper bound and repeatedly improves the upper bound by algorithm RTW. The call $RTW(G, k, \Pi)$, where $\Pi \subseteq \Pi(G)$ and $tw_{\Pi}(G) \leq k + 1$, decides if $tw(G) \leq k$. If $tw(G) \leq k$, it returns YES with certificate $\Pi' \subseteq \Pi(G)$ such that $tw_{\Pi'}(G) \leq k$; otherwise it returns NO with certificate H , a minimal contraction of G such that $tw(H) = k + 1$.

■ **Algorithm 1** Main iteration for computing $tw(G)$.

Ensure: compute $tw(G)$ for given G

```

1:  $T \leftarrow$  a minimal tree-decomposition of  $G$  obtained by a greedy algorithm
2:  $\Pi \leftarrow$  the set of bags of  $T$ 
3:  $k \leftarrow w(T)$ 
4: while true do
5:   call  $RTW(G, k - 1, \Pi)$ 
6:   if the call returns NO with certificate  $H$  then
7:     stop:  $tw(G)$  equals  $k$  with  $tw(G) \leq k$  certified by  $\Pi$  and  $tw(G) \geq k$  certified by  $H$ 
8:   else
9:      $k \leftarrow k - 1$ 
10:     $\Pi \leftarrow$  the certificate of the YES answer
11:  end if
12: end while

```

The pseudo code in Algorithm 2 describes RTW in its basic form. We sketch here the functions of subalgorithms used in this algorithm. More details can be found in subsequent sections.

Our method of local search in the space of sets of PMCs is a heuristic variant, which we call HPID, of the PID algorithm due to Tamaki [18]. PID constructs partial tree-decompositions of width $\leq k$ using the recurrence of BT dynamic programming in a bottom up manner to exhaustively generate all partial tree-decompositions of width $\leq k$, so that we have a tree-decomposition of width $\leq k$ if and only if $tw(G) \leq k$. HPID uses the same recurrence to generate partial tree-decompositions of width $\leq k$ but the aim is to quickly generate a tree-decomposition of G of width $\leq k$ and the generation order it employs does not guarantee exhaustive generation. The state of HPID computation is characterized by the set Π of root bags of the generated partial tree-decompositions. Recall that the bags of the set of partial tree-decompositions generated by the BT recurrence are PMCs, so $\Pi \subseteq \Pi(G)$. Using BT dynamic programming, we can reconstruct the set of partial tree-decompositions from Π , if needed, in time linear in $|\Pi|$ and polynomial in $|V(G)|$. Thus, we may view HPID as performing a local search in the space of sets of PMCs. This view facilitates communications between HPID and external upper bound heuristics. Those communications are done through the following operations.

We consider each invocation of HPID as an entity having a state. Let s denote such an invocation instance of HPID for G and k . Let $\Pi(s)$ denote the set of PMCs that are root bag of the partial tree-decompositions generated so far by s . The following operations are available.

$s.width()$ returns $tw_{\Pi(s)}(G)$.

$s.usefulPMCs()$ returns the set of PMCs that are the root bags of the partial tree-decompositions of width $\leq s.width()$ generated so far by s .

■ **Algorithm 2** Procedure $RTW(G, k, \Pi)$.

Require: $\Pi \subseteq \Pi(G)$ and $\text{tw}_{\Pi}(G) \leq k + 1$
Ensure: returns YES with $\Pi \subseteq \Pi(G)$ such that $\text{tw}_{\Pi}(G) \leq k$ if $\text{tw}(G) \leq k$; NO with a minimal contraction H of G such that $\text{tw}(H) = k + 1$ otherwise

- 1: create an HPID instance s for G and k
- 2: s .IMPORTPMCS(Π)
- 3: **if** s .width() $\leq k$ **then**
- 4: **return** YES with s .USEFULPMCS()
- 5: **end if**
- 6: order the edges of G appropriately as e_1, e_2, \dots, e_m .
- 7: **for** $i = 1, \dots, m$ **do**
- 8: $\Theta \leftarrow \text{CONTRACTPMCS}(s$.USEFULPMCS(), G, e_i)
- 9: call $RTW(G/e_i, k, \Theta)$
- 10: **if** the call returns NO with certificate H **then**
- 11: **return** NO with certificate H
- 12: **else**
- 13: $\Psi \leftarrow$ the certificate for the YES answer
- 14: $\Psi' \leftarrow \text{UNCONTRACTPMCS}(\Psi, G, e)$
- 15: s .IMPORTPMCS(Ψ')
- 16: s .IMPROVE($UNIT_BUDGET \times i$)
- 17: **if** s .width() $\leq k$ **then**
- 18: **return** YES with s .USEFULPMCS()
- 19: **end if**
- 20: **end if**
- 21: **end for**
- 22: s .FINISH()
- 23: **if** s .width() $\leq k$ **then**
- 24: **return** YES with s .USEFULPMCS()
- 25: **else**
- 26: **return** NO with certificate G
- 27: **end if**

s .importPMCS(Π) updates $\Pi(s)$ to $\Pi(s) \cup \Pi$ and updates the set of partial tree-decompositions by BT dynamic programming.

s .improve(budget) generates more partial tree-decompositions under the specified budget, in terms of the number of search steps spent for the generation.

s .finish() exhaustively generates remaining partial decompositions of width $\leq k$, thereby deciding if $\text{tw}(G) \leq k$.

See Section 4 for some details of these procedures.

We use two additional procedures.

uncontractPMCS(Π, G, e), where e is an edge of G and $\Pi \subseteq \Pi(G/e)$, returns $\Pi' \subseteq \Pi(G)$ such that $\text{tw}_{\Pi'}(G) \leq \text{tw}_{\Pi}(G/e) + 1$ and possibly $\text{tw}_{\Pi'}(G) \leq \text{tw}_{\Pi}(G/e)$

contractPMCS(Π, G, e), where e is an edge of G and $\Pi \subseteq \Pi(G)$, returns $\Pi' \subseteq \Pi(G/e)$ such that $\text{tw}_{\Pi'}(G/e) \leq \text{tw}_{\Pi}(G)$ and possibly $\text{tw}_{\Pi'}(G/e) \leq \text{tw}_{\Pi}(G) - 1$

See Sections 6 and 7 for details of these procedures.

The correctness of this algorithm can be proved by straightforward induction and does not depend on the procedures EXPAND, CONTRACTPMCS, or UNCONTRACTPMCS except that the procedure CONTRACTPMCS(Π, G, e) must return Θ such that $\text{tw}_{\Theta}(G/e) \leq \text{tw}_{\Pi}(G)$

as promised. On the other hand, practical efficiency of this algorithm heavily depends on the performances of these procedures. If they collectively work really well, then we expect that the **for** loop would exit after trying only a few edges, assuming $\text{tw}(G) \leq k$, and $s.\text{FINISH}()$ would be called only if $\text{tw}(G) = k + 1$ and $\text{tw}(G/e) \leq k$ for every edge e . On the other extreme of perfect incapability of these procedures, the **for** loop would always run to the end and $s.\text{FINISH}()$ would be called in every call of $\text{RTW}(G, k, \Pi)$, making the recursion totally meaningless. Our efforts are devoted to developing effective methods for these procedures.

4 Heuristic PID

In this section, we give some details of the HPID algorithm. In particular, we describe in some details how the procedures $\text{IMPROVE}(\text{budget})$ and $\text{FINISH}()$ work.

We first describe how we use Recurrence (1) to generate a new partial tree-decomposition from existing ones. The method basically follows that of PID [18] but differs in the way we view tree-decompositions as rooted-decompositions. The differences are motivated by the need of HPID to interact with external upper bound components through PMCs.

Fix G and k . We assume a total order $<$ on $V(G)$ and say that $U \subseteq V(G)$ is *larger* than $V \subseteq V(G)$ if $|U| > |V|$ or $|U| = |V|$ and U is lexicographically larger than V . We say a block B is *small* if there is some block B' with $N_G(B') = N_G(B)$ such that $B' > B$. We say that a block B of G is *feasible* if $\text{tw}(B, G) \leq k$. We use Recurrence (1), with Π set to $\Pi(G)$, to generate feasible blocks that are small.

Each HPID instance s maintains a set \mathcal{F} of small feasible blocks. To generate a new feasible block to add to \mathcal{F} , it invokes a backtrack search procedure $\text{SEARCHNEWFEASIBLE}(B)$ on a block $B \in \mathcal{F}$ which enumerates $\mathcal{B} \subseteq \mathcal{F}$ such that

1. $B \in \mathcal{B}$ and B is the largest block in \mathcal{B} and
2. there is a block $B_{\mathcal{B}}$ that is either small or is equal to $V(G)$ and a PMC $X_{\mathcal{B}} \in \Pi(G)$ such that $\mathcal{C}(B_{\mathcal{B}} \setminus X_{\mathcal{B}}) = \mathcal{B}$.

For each such \mathcal{B} found, we add $B_{\mathcal{B}}$ to \mathcal{F} since the Recurrence (1) shows that $B_{\mathcal{B}}$ is feasible.

Procedure $s.\text{IMPROVE}(\text{budget})$ uses this search procedure as follows. It uses a priority queue Q of small feasible blocks, in which larger blocks are given higher priority. It first put all blocks in \mathcal{F} to Q . Then, it dequeues a block B , call $\text{SEARCHNEWFEASIBLE}(B)$, and add newly generated feasible blocks to Q . This is repeated until either Q is empty or the cumulative number of search steps exceeds budget. Because of the queuing policy, there is a possibility of $V(G)$ found feasible, when it is indeed feasible, even with a small budget.

The role of procedure $s.\text{FINISH}()$ is to complete the PID computation by exhaustively generating partial tree-decompositions. The implementation used in our experiment uses another variant of PID called SemiPID [16] for this task.

5 Minimalizing tree-decompositions

Given a graph G and a triangulation H of G , *minimalizing* H means finding a minimal triangulation H' of G such that $E(H') \subseteq E(H)$. Minimalizing a tree-decomposition T of G means finding a minimal tree-decomposition T' of G whose bags are maximal cliques of the minimalization of $\text{fill}(G, T)$. We want to minimalize a tree-decomposition for two reasons. One is our decision to represent a set of tree-decompositions by a set of PMCs. Whenever we get a tree-decomposition T by some method that may produce non-minimal tree-decompositions, we minimalize it to make all bags PMCs. Another reason is that minimalization may reduce the width. We have two procedures for minimalization. When

the second reason is of no concern, we use $\text{MINIMALIZE}(T)$ which is an implementation of one of the standard triangulation minimalization algorithm due to Blair *et al* [3]. When the second reason is important, we use $\text{MINIMALIZEOPTIMALLY}(T)$, which finds a minimalization of T of the smallest width. This task is NP-hard, but the following algorithm works well in practice.

Say a minimal separator of G is *admissible for T* if it is a clique of $\text{fill}(G, T)$. Observe that, for every minimalization T' of T , every separator used by T' is a minimal separator of G admissible for T . We first construct the set of all minimal separators of G admissible for T . Then we apply the SemiPID variant of BT dynamic programming, due to Tamaki [16], to this set and obtain a tree-decomposition of the smallest width, among those using only admissible minimal separators. Because of the admissibility constraint, the number of minimal separators is much smaller and both the enumeration part and the SemiPID part run much faster in practices than in the general case without such constraints.

6 Uncontracting PMCs

In this section, we develop an algorithm for procedure $\text{UNCONTRACTPMCs}(G, \Pi, e)$. In fact, we generalize this procedure to $\text{UNCONTRACTPMCs}(G, \Pi, \gamma)$, where the third argument is a general contractor of G .

Given a graph G , $\Pi \subseteq \Pi(G)$, and a contractor γ of G , we first find tree-decompositions $T \in \mathcal{T}_\Pi$ that minimize $w(\gamma^{-1}(T))$. This is done by BT dynamic programming over $\mathcal{T}_\Pi(G/\gamma)$, using bag weights defined as follows. For each weight function ω that assigns weight $\omega(U)$ to each vertex set U , define the width of tree-decomposition T with respect to ω , denoted by $\text{tw}(G, \omega)$, to be the maximum of $\omega(X)$ over all bags of T . Thus, if ω is defined by $\omega(U) = |U| - 1$ then $\text{tw}(G, \omega) = \text{tw}(G)$. A natural choice for our purposes is to set $\omega(X) = |\gamma^{-1}(X)| - 1$. Then, the width of a tree decomposition T of G/γ with respect to this bag weight is $w(\gamma^{-1}(T))$. Therefore, BT dynamic programming with this weight function ω gives us the desired tree-decomposition in $\mathcal{T}_\Pi(G/r)$.

We actually use a slightly modified weight function, considering the possibility of reducing the weight of $\gamma^{-1}(T)$ by minimalization.

Let $T \in \mathcal{T}_\Pi(G/\gamma)$ and X a bag of T . If $X' = \gamma^{-1}(X)$ is a PMC of G , then every minimalization of $\gamma^{-1}(T)$ must contain X' as a bag. Therefore, if $|X'| > k + 1$ then it is impossible that the width of $\gamma^{-1}(T)$ is reduced to k by minimalization. On the other hand, if X' is not a PMC, then no minimalization of $\gamma^{-1}(T)$ has X' as a bag and there is a possibility that there is a minimalization of $\gamma^{-1}(T)$ of width k even if $|X'| > k + 1$. These considerations lead to the following definition of our weight function ω .

$$\omega(U) = 2|\gamma^{-1}(U)| \quad \text{if } \gamma^{-1}(U) \text{ is a PMC of } G \quad (2)$$

$$\omega(U) = 2|\gamma^{-1}(U)| - 1 \quad \text{otherwise} \quad (3)$$

Algorithm 3 describes the main steps of procedure $\text{UNCONTRACTPMCs}(\Pi, G, \gamma)$.

7 Contracting PMCs

The algorithm for procedure CONTRACTPMCs is similar to that for UNCONTRACTPMCs . Given a graph G , $\Pi \subseteq \Pi(G)$, and a contractor γ of G , we first find tree-decompositions $T \in \mathcal{T}_\Pi(G/\gamma)$ that minimize $w(\gamma(T))$. This is done by BT dynamic programming with the following weight function ω .

$$\omega(U) = 2|\gamma(U)| \quad \text{if } \gamma(U) \text{ is a PMC of } G/\gamma$$

$$\omega(U) = 2|\gamma(U)| - 1 \quad \text{otherwise}$$

■ **Algorithm 3** Procedure UNCONTRACTPMCS(Π, G, γ).

Require: $\Pi \subseteq \Pi(G/\gamma)$

Ensure: returns $\Pi' \subseteq \Pi(G)$ that results from uncontracting Π and then minimalizing

- 1: let ω be the weight function on $2^{V(G/\gamma)}$ defined by equations 2 and 3
- 2: use BT dynamic programming to obtain tree-decompositions T_i , $1 \leq i \leq m$, of G/γ such that $w(T_i, \omega) = \text{tw}_\Pi(G, \omega)$
- 3: **for** each i , $1 \leq i \leq m$ **do**
- 4: $T'_i \leftarrow \text{MINIMIZEOPTIMALLY}(\gamma^{-1}(T_i))$
- 5: $\Pi_i \leftarrow$ the set of bags of T'_i
- 6: **end for**
- 7: **return** $\bigcup_i \Pi_i$

Then, we minimalize those tree-decompositions and collect the bags of those minimalized tree-decompositions.

8 Safe separators

Bodlaender and Koster [6] introduced the notion of safe separators for treewidth. Let S be a separator of a graph G . We say that S is *safe for treewidth*, or simply *safe*, if $\text{tw}(G) = \text{tw}(G \cup K(S))$. As every tree-decomposition of $G \cup K(S)$ must have a bag containing S , $\text{tw}(G)$ is the larger of $|S| - 1$ and $\max\{\text{tw}(G[C \cup N_G(C)] \cup K(N_G(C)))\}$, where C ranges over all the components of $G \setminus S$. Thus, the task of computing $\text{tw}(G)$ reduces to the task of computing $\text{tw}(G[C \cup N_G(C)] \cup K(N_G(C)))$ for every component C of $G \setminus S$. The motivation for looking at safe separators of a graph is that there are sufficient conditions for a separator being safe and those sufficient conditions lead to an effective preprocessing method for treewidth computation. We use the following two sufficient conditions.

A vertex set S of G is an *almost-clique* if $S \setminus \{v\}$ is a clique for some $v \in S$. Let R be a vertex set of G . A contractor γ of G is *rooted on R* if, for each part C of γ , $|C \cap R| = 1$.

► **Theorem 1** (Bodlaender and Koster [6]).

1. If S is an almost-clique minimal separator of G , then S is safe.
2. Let lb be a lower bound on $\text{tw}(G)$. Let $C \subseteq V(G)$ be connected and let $S = N_G(C)$. Suppose (1) $\text{tw}(G[C \cup S] \cup K(S)) \leq lb$ and (2) $G[C \cup S]$ has a contractor γ rooted on S such that $G[C \cup S]/\gamma$ is a complete graph. Then, S is safe.

We use safe separators both for preprocessing and during recursion. For preprocessing, we follow the approach of [19]: to preprocess G , we fix a minimal triangulation H of G and test the sufficient conditions in the theorem for each minimal separator of H . Since deciding if the second condition holds is NP-complete, we use a heuristic procedure. Let \mathcal{S} be the set of all minimal separators of H that are confirmed to satisfy the first or the second condition of the theorem. Let \mathcal{A} be a tree-decomposition of G that uses all separators of \mathcal{S} but no other separators. Then, \mathcal{A} is what is called a *safe-separator decomposition* in [6]. A tree-decomposition of G of width $\text{tw}(G)$ can be obtained from \mathcal{A} by replacing each bag X of \mathcal{A} by a tree-decomposition of $G[X] \cup \bigcup_{C \in \mathcal{C}(G \setminus X)} K(N_G(C))$, the graph obtained from the subgraph of G induced by X by filling the neighborhood of every component of $G \setminus X$ into a clique.

Safe separators are also useful during the recursive computation. Given G , we wish to find a contractor γ of G such that $\text{tw}(G/\gamma) = \text{tw}(G)$, so that we can safely recurse on G/γ . The second sufficient condition in Theorem 1 is useful for this purpose. Let C, S ,

34:12 A Contraction-Recursive Algorithm for Treewidth

and γ be as in the condition. We construct γ' such that $\text{tw}(G/\gamma') = \text{tw}(G)$ as follows. The proof of this sufficient condition is based on the fact that we get a clique on S when we apply the contractor γ on $G[C \cup S]$. Thus, we may define a contractor γ' on G such that $G/\gamma' = (G \setminus C) \cup K(S)$. As each tree-decomposition of $\text{tw}(G/\gamma)$ can be extended to a tree-decomposition of G , using the tree-decomposition of $G[C \cup S] \cup K(S)$ of width at most $lb \leq \text{tw}(G)$, we have $\text{tw}(G/\gamma') = \text{tw}(G)$ as desired. When the recursive call on $\text{tw}(G/\gamma')$ returns a certificate $\Pi \subseteq \Pi(G/\gamma')$ such that $\text{tw}_\Pi(G/\gamma') \leq k$, we need to "uncontract" Π into a $\Pi' \subseteq \Pi(G)$ such that $\text{tw}_{\Pi'}(G) \leq k$. Fortunately, this can be done without invoking the general uncontraction procedure. Observe first that each PMC in Π naturally corresponds to a PMC of $(G \setminus C) \cup K(S)$, which in turn corresponds to a PMC of G contained in $V(G) \setminus C$. Let Π_1 be the set of those PMCs of G to which a PMC in Π corresponds in that manner. Let $\Pi_2 \subseteq \Pi(G[C \cup S] \cup K(S))$ be such that $\text{tw}_{\Pi_2}(G[C \cup S] \cup K(S)) \leq lb$. Similarly as above, each PMC of Π_2 corresponds to a PMC of G contained $C \cup S$. Let Π'_2 denote the set of those PMCs of G to which a PMC in Π_2 corresponds. As argued above, a tree-decomposition in $\mathcal{T}_\Pi((G \setminus C) \cup K(S))$ of $(G \setminus C) \cup K(S)$ and a tree-decomposition in $\mathcal{T}_{\Pi_2}(G[C \cup S] \cup K(S))$ of $G[C \cup S] \cup K(S)$ can be combined into a tree-decomposition belonging to $\mathcal{T}_{\Pi'_2}(G)$ of width $\leq k$. Thus, Π'_2 is a desired certificate for $\text{tw}(G) \leq k$.

9 Edge ordering

We want an edge e such that $\text{tw}(G/e) = \text{tw}(G)$, if such exists, to appear early in our edge order. Heuristic criteria for such an ordering have been studied in the classic work on contraction based lower bounds [8]. Our criterion is similar to those but differs in that it derives from a special case of safe separators. The following is a simple corollary of Theorem 1.

► **Proposition 2.** *Let $e = \{u, v\}$ be an edge of G and let $S = N_G(v)$. Suppose $S \setminus \{u\}$ is a clique of G . Then, we have $\text{tw}(G/e) = \text{tw}(G)$.*

If e satisfies the above condition, then we certainly put e first in the order. Otherwise, we evaluate e in terms of its closeness to this ideal situation. Define the *deficiency* of graph H , denoted by $\text{defic}(H)$, to be the number of edges of its complement graph. For each ordered pair (u, v) of adjacent vertices of G , let $\text{defic}_G(u, v)$ denote $\text{defic}(G[N_G(v) \cup \{v\}]/\{u, v\})$. Note that $\text{defic}_G(u, v) = 0$ means that the condition of the above proposition is satisfied with $S = N_G(v)$. Thus, we regard $e = \{u, v\}$ preferable if either $\text{defic}_G(u, v)$ or $\text{defic}_G(v, u)$ is small. We relativize the smallness with respect to the neighborhood size, so the *value* of edge $e = \{u, v\}$ is $\min\{\text{defic}_G(u, v)/|N_G(v)|, \text{defic}_G(v, u)/|N_G(u)|\}$. We order edges so that this value is non-decreasing.

10 Suppressed edges

Consider the recursive call on G/e from the call of RTW on G , where e is an edge of G . Suppose there is an ancestor call on G' such that $G = G'/\gamma$ and edge e' of G' such that γ maps the ends of e' to the ends of e . If the call on G'/e' has been made and it is known that $\text{tw}(G'/e') \leq k$ then we know that $\text{tw}(G/e) \leq k$, since G/e is a contraction of G'/e' . In this situation, we say that e is *suppressed* by the pair (G', e') . We may omit the recursive call on G/e without compromising the correctness if e is suppressed. For efficiency, however, it is preferable to obtain the certificate $\Pi \subseteq \Pi(G/e)$ for $\text{tw}(G/e) \leq k$ and feed the uncontraction of Π to the HPID instance on G to help progress. Fortunately, this can be done without making

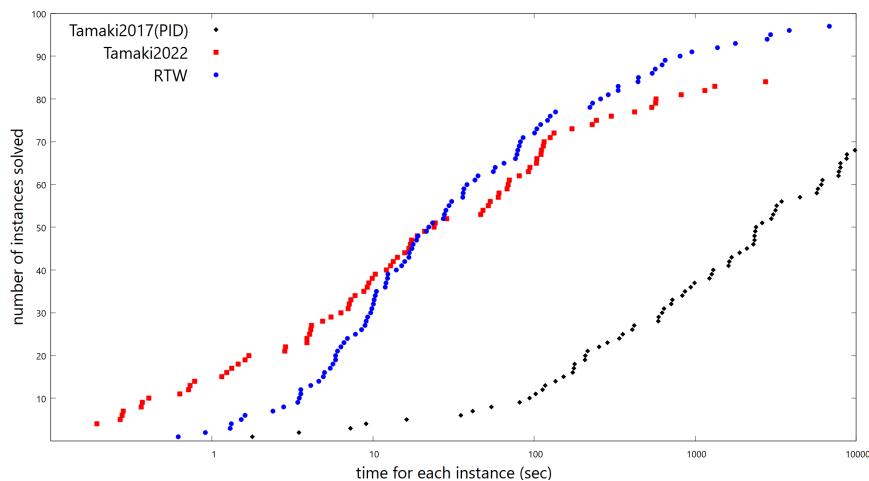
the recursive call on G as follows. Suppose e is suppressed by (G', e') and let $\Pi' \subseteq \Pi(G'/e')$ such that $\text{tw}_{\Pi'}(G'/e') \leq k$. Let γ' be the contractor of G'/e' such that $G'/e'/\gamma' = G/\gamma/e$: such γ' is straightforward to obtain from γ . Letting $\Pi = \text{CONTRACTPMCS}(\Pi, G'/e', \gamma')$, we obtain $\Pi \subseteq \Pi(G/e)$ such that $\text{tw}_{\Pi}(G/e) \leq k$.

11 Experiments

We have implemented RTW and evaluated it by experiments. The computing environment for our experiments is as follows. CPU: Intel Core i7-8700K, 3.70GHz; RAM: 64GB; Operating system: Windows 10Pro, 64bit; Programming language: Java 1.8; JVM: jre1.8.0_271. The maximum heap size is set to 60GB. The implementation uses a single thread except for additional threads that may be invoked for garbage collection by JVM.

Our primary benchmark is the bonus instance set of the exact treewidth track of PACE 2017 algorithm implementation challenge [11]. This set, consisting of 100 instances, is intended to be a challenge for future implementations and, as a set, are hard for the winning solvers of the competition. Using the platform of the competition, about half of the instances took more than one hour to solve and 15 instances took more than a day or were not solvable at all.

We have run our implementation on these instances with the timeout of 10000 seconds each. For comparison, we have run Tamaki's PID solver [18], which is one of the PACE 2017 winners, available at [15] and his new solver [20] available at [21]. Figure 1 summarizes the results on the bonus set. In contrast to PID solver which solves only 68 instances within the timeout, RTW solves 98 instances. Moreover, it solve 72 of them in 100 seconds and 92 of them in 1000 seconds. Thus, we can say that our algorithm drastically extends the scope of practically solvable instances. Tamaki's new solver also quickly solves many instances that are hard for PID solver and is indeed faster than RTW on many instances. However, its performance in terms of the number of instances solvable in practical time is inferior to RTW.

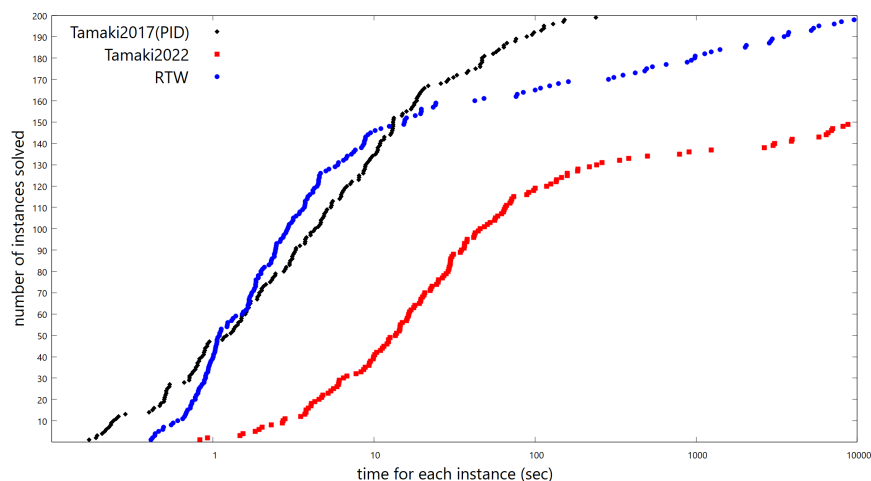


■ **Figure 1** Number of bonus instances solved within a specified time.

We have also run the solvers on the competition set of the exact treewidth track of PACE 2017. This set, consisting of 200 instances, is relatively easy and the two winning solvers of the competitions solved all of the instances within the allocated timeout of 30 minutes for each instance. Figure 2 summarizes the results on the competition set. Somewhat expectedly,

PID performs the best on this instance set. It solves almost all instances in 200 seconds for each instance, while RTW fails to do so on about 30 instances. There are two instances that RTW fails to solve in 10000 seconds and one instance it fails to solve at all. Tamaki’s new solver shows more weakness on this set, failing to solve about 50 instances in the timeout of 10000 seconds.

These results seem to suggest that RTW and PID should probably complement each other in a practical treewidth solver.



■ **Figure 2** Number of competition instances solved within a specified time.

12 Conclusions and future work

We developed a treewidth algorithm RTW that works recursively on contractions. Experiments show that our implementation solves many instances in practical time that are hard to solve for previously published solvers. RTW, however, does not perform well on some instances that are easy for conventional solvers such as PID. A quick compromise would be to run PID first with an affordable timeout and use RTW only when it fails. It would be, however, interesting and potentially fruitful to closely examine those instances that are easy for PID and hard for RTW and, based on such observations, to look for a unified algorithm that avoids the present weakness of RTW.

References

- 1 Ernst Althaus, Daniela Schnurbusch, Julian Wüschner, and Sarah Ziegler. On tamaki’s algorithm to compute treewidths. In *19th International Symposium on Experimental Algorithms (SEA 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.
- 2 Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k -tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.
- 3 Jean RS Blair, Pinar Heggernes, and Jan Arne Telle. A practical algorithm for making filled graphs minimal. *Theoretical Computer Science*, 250(1-2):125–141, 2001.
- 4 Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.
- 5 Hans L Bodlaender, Fedor V Fomin, Arie MCA Koster, Dieter Kratsch, and Dimitrios M Thilikos. On exact algorithms for treewidth. In *Algorithms-ESA 2006: 14th Annual European*

- Symposium, Zurich, Switzerland, September 11-13, 2006. Proceedings*, pages 672–683. Springer, 2006.
- 6 Hans L Bodlaender and Arie MCA Koster. Safe separators for treewidth. *Discrete Mathematics*, 306(3):337–350, 2006.
 - 7 Hans L Bodlaender and Arie MCA Koster. Treewidth computations i. upper bounds. *Information and Computation*, 208(3):259–275, 2010.
 - 8 Hans L Bodlaender and Arie MCA Koster. Treewidth computations ii. lower bounds. *Information and Computation*, 209(7):1103–1119, 2011.
 - 9 Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM Journal on Computing*, 31(1):212–232, 2001.
 - 10 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshтанov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.
 - 11 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The pace 2017 parameterized algorithms and computational experiments challenge: The second iteration. In *12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
 - 12 Fedor V Fomin and Yngve Villanger. Treewidth computation and extremal combinatorics. *Combinatorica*, 32(3):289–308, 2012.
 - 13 Pinar Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.
 - 14 Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986.
 - 15 Hisao Tamaki. PID. <https://github.com/TCS-Meiji/PACE2017-TrackA>, 2017. [github repository].
 - 16 Hisao Tamaki. Computing treewidth via exact and heuristic lists of minimal separators. In *International Symposium on Experimental Algorithms*, pages 219–236. Springer, 2019.
 - 17 Hisao Tamaki. A heuristic use of dynamic programming to upperbound treewidth. *arXiv preprint arXiv:1909.07647*, 2019.
 - 18 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *Journal of Combinatorial Optimization*, 37(4):1283–1311, 2019.
 - 19 Hisao Tamaki. A heuristic for listing almost-clique minimal separators of a graph. *arXiv preprint arXiv:2108.07551*, 2021.
 - 20 Hisao Tamaki. Heuristic Computation of Exact Treewidth. In Christian Schulz and Bora Uçar, editors, *20th International Symposium on Experimental Algorithms (SEA 2022)*, volume 233 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:16, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SEA.2022.17.
 - 21 Hisao Tamaki. twalgor/tw. <https://github.com/twalgor/tw>, 2022. [github repository].

PACE Solver Description: The PACE 2023 Parameterized Algorithms and Computational Experiments Challenge: Twinwidth

Max Bannach  

European Space Agency, Advanced Concepts Team, Noordwijk, The Netherlands

Sebastian Berndt  

Institute for Theoretical Computer Science, University of Lübeck, Germany

Abstract

This article is a report by the challenge organizers on the 8th Parameterized Algorithms and Computational Experiments Challenge (PACE 2023). As was common in previous iterations of the competition, this year's iteration implemented an exact and heuristic track for a parameterized problem that has gained attention in the theory community. This year, the problem was to compute the *twinwidth* of a graph, a recently introduced width parameter that measures the similarity of a graph to a cograph. In the exact track, the competition participants were asked to develop an exact algorithm that can solve as many instances as possible from a benchmark set of 100 instances – with a time limit of 30 minutes per instance. The same task must be accomplished within 5 minutes in the heuristic track. However, the result in this track is not required to be optimal.

As in previous iterations, the organizers handed out awards to the best solutions in both tracks and to the best student submissions. New this year is a dedicated *theory award* that appreciates new theoretical insights found by the participants during the development of their tools.

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

Keywords and phrases Twinwidth, Algorithm Engineering, FPT, Kernelization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.35

Acknowledgements The prize money (€4000) was generously provided by Networks [28], an NWO Gravitation project of the University of Amsterdam, Eindhoven University of Technology, Leiden University and the Center for Mathematics and Computer Science (CWI). We are grateful to the whole optil.io team, led by Szymon Wasik, and especially to Jan Badura and Artur Laskowski for the fruitful collaboration and for hosting the competition at the optil.io online judge system. We also thank André Schidler and Stefan Szeider, who made their exact solver available to the organizers prior to the competition for internal evaluations [29].

1 Introduction: History and Timeline of PACE

The Parameterized Algorithms and Computational Experiments Challenge (PACE) is an algorithm engineering competition conceived in 2015 and held annually since. Its aim is to bridge the theory of parameterized algorithms and their use in practice. The goals are to:

1. bridge the divide between the theory of algorithm design and analysis, and the practice of algorithm engineering,
2. inspire new theoretical developments,
3. investigate the competitiveness of theoretical algorithms from the field of parameterized complexity analysis and related fields in practice,
4. produce universally accessible libraries of implementations and repositories of benchmark instances, and
5. encourage the dissemination of these findings in scientific papers.



© Max Bannach and Sebastian Berndt;

licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 35; pp. 35:1–35:14

Leibniz International Proceedings in Informatics



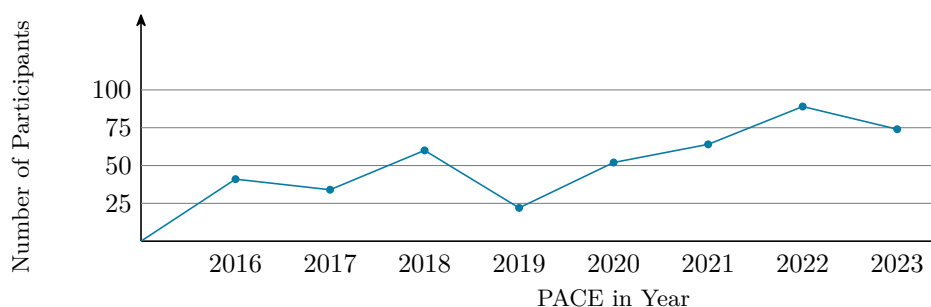
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

35:2 PACE Solver Description: PACE 2023: Twinwidth

In each of the now eight iterations, the participants were asked to provide implementations that either solved small- to medium-sized instances of an NP-complete problem optimally or to solve large-sized instances approximately. In previous iterations, the problems were

PACE 2016	TREEWIDTH and UNDIRECTED-FEEDBACK-VERTEX-SET	[14];
PACE 2017	TREEWIDTH and MINIMUM FILL-IN	[15];
PACE 2018	STEINER-TREE	[12];
PACE 2019	VERTEX-COVER and HYPERTREEWIDTH	[17];
PACE 2020	TREEDEPTH	[22];
PACE 2021	CLUSTER-EDITING	[21];
PACE 2022	DIRECTED-FEEDBACK-VERTEX-SET	[18].

Several of the previous iterations also contained more specialized tracks. Starting with the first iteration of PACE, many participants from all over the world were interested in the challenge and quickly established PACE as a highly competitive challenge. Over the years, the number of PACE participants has constantly grown, as Figure 1 illustrates. Furthermore, papers inspired by concrete implementations were published in prestigious conferences such as ACDA, ALENEX, ESA (Track B), SEA, and WADS. The instances provided by PACE have also often been used to showcase further algorithmic improvements by being used as an established benchmark, ranging also to other competitions such as the famous SAT competition [5].

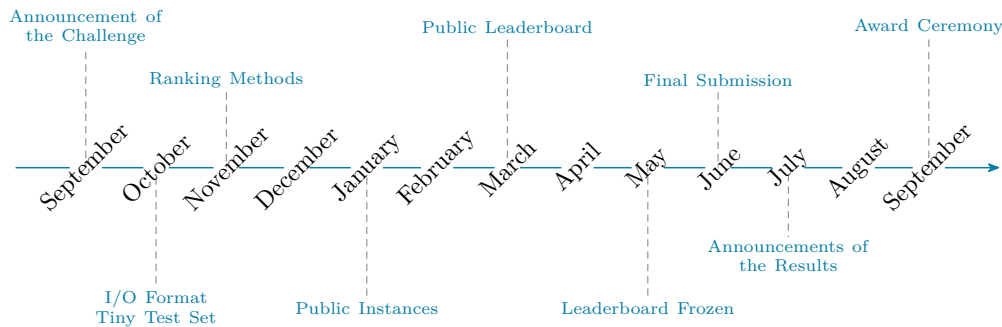


■ **Figure 1** Overview of the number of participants (y -axis) of the PACE challenge over the years.

In this article, we report on the eighth iteration of the PACE challenge. The problem chosen for this year's iteration was *twinwidth*, a relatively young but promising parameter introduced in 2020 by Bonnet et al. [10]. The challenge featured two tracks: an exact track and a heuristic track. In the exact track, the task was to find an optimal solution of a given instance within 30 minutes and a memory limit of 8 GB. In the heuristic track, the task was to compute a valid solution with a width as small as possible within a time limit of 5 minutes and a memory limit of 8 GB.

The PACE 2023 challenge was announced with both tracks in September 2022. Details about the input and output format were provided in October 2022 together with a tiny test set to allow the participants to start with the challenge. One reason being, in particular, that algorithmic lectures that eventually start in this period may integrate PACE into course related projects. The concrete ranking methods for both tracks were published in November 2022. In January 2023, the public instances were made available to the participants, and the public leaderboard on the `optil.io` platform was opened in March 2023. This allowed the participants to test their solvers on the public instances and provided a provisional ranking. The leaderboard was frozen in May 2023, and the final version of the submission was due on the first of June 2023. Afterwards, the submissions were evaluated on the private

instances, which the participants did not know. The results of this evaluation were announced in July 2023, and the award ceremony took place during the International Symposium on Parameterized and Exact Computation (IPEC) 2023 in Amsterdam in September 2023. The complete timeline can be found in Figure 2.



■ **Figure 2** Timeline of the PACE challenge in 2023 (the diagram ranges from September 2022 to September 2023). The next iteration of the PACE for 2024 was announced at the award ceremony.

2 The Problem of the Challenge: Twinwidth

So-called *width measures* are graph parameters that capture the structural complexity of a graph in terms of the minimum cost of an associated type of decomposition of the graph. The poster child of such a parameter is *treewidth*, which measures the distance of a graph to a tree. A recently proposed parameter is *twinwidth*, introduced by Bonnet, Kim, Thomassé, and Watrigant [11]. Informally, the twinwidth of a graph measures the distance to a *cograph*. Cographs are the graphs that can be generated from a single-vertex graph by complementation and disjoint union; which are precisely the P_4 -free graphs. In the context of twinwidth, the crucial property of these graphs is that they always contain a pair of *twins*, that is, vertices sharing the same neighborhood. The contraction of twins in a cograph results again in a cograph. In other words, cographs can be reduced to a single vertex by a sequence of twin contractions.

If the input graph $G = (V, E)$ is not a cograph, this process of contracting twins will eventually get stuck with a graph that contains more than one vertex, but does not contain any twins. We consider the contraction of non-twins (which may or may not be adjacent) as “error” (because the goal is to measure the distance to cographs) and record this error by coloring mixed edges “red,” while referring to the original edges as “black edges”. Precisely, when contracting two vertices u and v into a single fresh vertex z , we form a new graph by removing u and v . For each $x \in V \setminus \{u, v\}$ that was connected to u or v , we add a new edge between x and z . If x was connected to both u and v and both of these edges were black, the edge between x and z is black as well. In all other cases, the edge between x and z is colored red. Figure 3 presents an example: When a and c are contracted in the second picture into the fresh vertex ac , then d gets connected to ac with a black edge (as d was connected to both a and c). In contrast, e was only connected to c but not to a and, hence, a red edge now connects e and ac . Finally, b was not connected to a or to c and hence, no edge is drawn between b and ac .

A *contraction sequence* consists of $|V| - 1$ contractions transforming the input graph (which does not contain any red edges) into a single vertex (which also does not contain red edges). However, intermediate graphs in the sequence may contain red edges. The *width* of a

contraction sequence is the maximal red degree of any vertex encountered in any intermediate graph. The *twinwidth* of a graph G is the minimal width any contraction sequence of G must have. We denote it by $\text{tww}(G)$ and derive the following computational task:

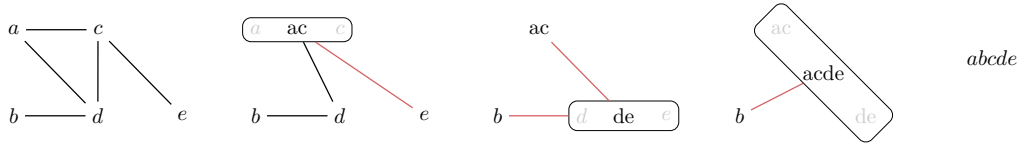
► **Challenge Problem** (TWINWIDTH).

Instance: A undirected graph G given as list of edges.

Task: Compute a contraction sequence S of G and output it as list of pairs.

Exact Track: The width of S must equal $\text{tww}(G)$.

Heuristic Track: Any valid sequence S can be output. The quality is the width of S .



■ **Figure 3** An exemplary contraction sequence of width 2 from the The sequence shows that the input graph (most left) has twinwidth at most 2. The reason being that the third graph has a red degree of 2.

Twinwidth has received wide attention in the three years following its inception: dblp alone lists (at the time of writing this report) 4 papers related to twinwidth in 2020, 15 in 2021, and already 37 in 2022. One of the reasons for its success is that the class of graphs of bounded twinwidth is one of the largest known graph classes that still allows first-order model checking in fpt-time [11]. There is also a growing number of intractable problems for which efficient algorithms on graphs of bounded twinwidth were found [8, 10]. However, all these results require that a contraction sequence of small width is given along with the input, i. e., one needs to solve the aforementioned TWINWIDTH problem as a preprocessing step. Most desirable would, of course, be an algorithm with polynomial running time that computes a contraction sequence of minimal width, or at least of width $\alpha \text{tww}(G)$ for some constant α . Unfortunately, the existence of an exact XP-algorithm or an approximation algorithm with $\alpha < 5/4$ was ruled out by Bergé, Bonnet, and Déprés, who proved that deciding whether $\text{tww}(G)$ is at most 4 is already NP-complete [7]. Nevertheless, nothing is known about $\alpha \geq 5/4$.

This is in strong contrast to problems chosen in previous iterations of PACE, where usually a portfolio of parameterized algorithms was available before the competition. That is, the first aim of PACE mentioned in the introduction, i. e., to *bridge the divide between the theory of algorithm design and analysis, and the practice of algorithm engineering* was interpreted mainly as “evolving theoretical insights into practical tools”. With TWINWIDTH as the problem of the challenge, the direction of this reasoning is inverted in this year’s iteration, i. e., we wondered whether “techniques commonly used in practice may lead to new results in theory”? This is also the reason that the organizers decided to hand out a dedicated theory award along with the usual ranking: To value and highlight theoretical insights gained during the competition in spirit of the second aim of the competition (to *inspire new theoretical developments*).

3 The Setup of PACE 2023

As already mentioned before, this year’s challenge featured two tracks: An *exact* track in which a contraction sequence of minimal width needs to be computed, and a *heuristic* track in which a not necessarily optimal contraction sequence must be computed very quickly.

3.1 The Exact Track (Compute an Optimal Contraction Sequence)

The task of this track is to compute an optimal solution of TWINWIDTH for 100 graphs, which are not known by the participants (they are only presented to the solver during the evaluation in a compartmentalized judge system). For each of the graphs, the solver has to output a solution within a time limit of *30 minutes* and a memory limit of *8 GB*.

The organizers of the competition encouraged submissions that implement provably optimal algorithms, however, this was not a formal requirement. Instead, submissions that output an incorrect solution or a solution that is known to be non-optimal were disqualified from the challenge. Fortunately, no submission was disqualified this year.

Submissions of this track are ranked by the *number of solved instances*. In case of a tie, the winner is determined by the *total time spent on the solved instances*. In particular, there was no need to abort a “hopeless” run early.

3.2 The Heuristic Track (Compute a Contraction Sequence Quickly)

In this track the solver shall compute a good solution quickly. The solver are run on each instance for *5 minutes* and receive the Unix signal SIGTERM afterwards. When receiving this signal, the solver has to output a correct contraction sequence immediately to the standard output and terminate. If the program does not halt in a reasonable time after receiving the signal, it will be stopped via SIGKILL and the instance is counted as time limited exceeded. The memory limit for this track is *8 GB* as well.

For this track solutions do not have to be optimal. However, solvers that produce incorrect solution were disqualified. Fortunately, we did not need to disqualify any solver in this track either. Submissions were ranked by the geometric mean over all instances of

$$100 \cdot \frac{\text{width produced by the solver}}{\text{smallest width known to the PC}}$$

Note that the “smallest width known to the PC” may not be optimal, i. e., may be larger than $\text{tww}(G)$.

3.3 Internal Solver and the Benchmark Set

The fourth aim of the PACE challenge is to *produce universally accessible libraries of implementations and repositories of benchmark instances*. While the first part of this aim is exactly what we expect from the participants, it is the duty of the program committee to produce the benchmark instances. A lot can be said about how to set up a good benchmarkset, and what “good” in this context actually means. In the light of PACE, we found the following points important:

- a) the benchmarkset contains instances from various domains,
- b) it contains easy, medium, hard instances,
- c) it remains a challenging benchmark after the challenge.

The reason for a) being that we would like to evaluate the overall performance of the approaches developed by the participants (and not the performance on, say, a specific graph class). We include b) to make the challenge interesting and fun. We wanted a benchmarkset in which every participant can solve at least a few instances, which should especially encourage student teams to participate as well. The medium instances are the ones that are meant to distinguish the quality of the various solvers, and the hard instances realize c). This last

item is included, as we did not want a benchmarkset that is “completely solved” after the competition. We expected that these hard instances are barely solvable by solvers developed in the time span of the competition and, thus, leave room for further research.

The emerging question of course is: “What is an easy, medium, or hard instance”? This question is not easy to answer, especially if barely anything is known about solving the problem. The organizers used a SAT-based internal solver to answer this question (the solver was developed by André Schidler and Stefan Szeider [30], who provided the organizers with an improved version of the tool before the competition).

The *base set* of graphs was generated from three sources:

Random and synthetic graphs, including: Erdős-Rényi graphs, random hyperbolic graphs, random planar graphs, and graph products of various simple graph classes.

Instances from graph repositories, including: road networks, power grid graphs, graphs from the UAI competition 2014, and max-flow instances from applications in computer vision.

Instances generated by reduction from SAT, including: the instances of the SAT competition 2022 *Anniversary Track*, reduced to VERTEX-COVER and 3-COLORING.

From this base set we generated an *enhanced set* by performing the following steps to every graph multiple times: (i) sample a random vertex, (ii) perform a BFS for ℓ layers to obtain a smaller subgraph, and (iii) add some noise to this subgraph by randomly (with a small probability ϵ) add, remove, or contract edges. By varying ℓ and ϵ , graphs of various sizes and difficulties can be generated. We ended up with an enhanced set of roughly 8000 graphs. To obtain the benchmarkset used in the competition, we let the internal solver run on every instance of the enhanced set for eight hours. We classified an instance as

Easy if the internal solver solved the instance in *30 minutes*;

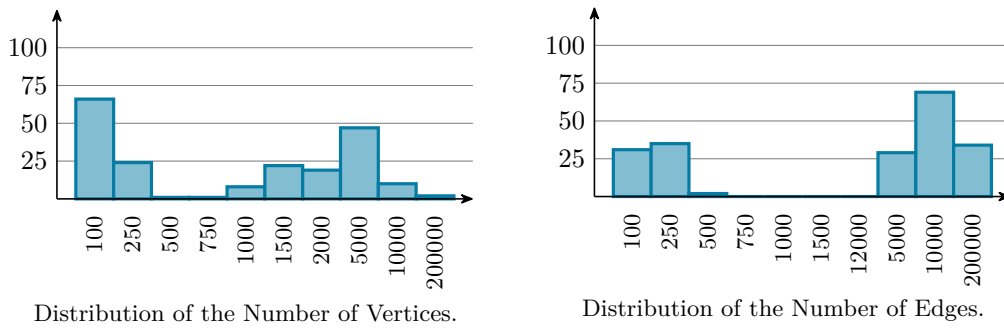
Medium if the internal solver *solved* the instance;

Hard if the internal solver did *not* solve the instance.

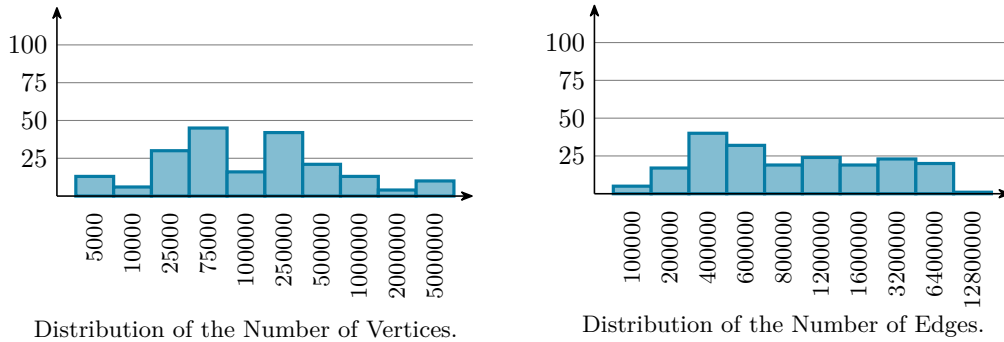
For the *exact track* we sampled 200 instances from this set (50 easy, 50 medium, and 100 hard). We sorted these 200 instances by size. Graphs with an odd number were used as private set to rank the participants, graphs with an even number were released in advance as public test set. Figure 4 illustrates the number of vertices (left) and number of edges (right) that the 200 instances of the *exact track* have. We used the same technique to derive the instances for the heuristic track, but scaled them by varying (the aforementioned parameter) ℓ after the classification into easy, medium, and hard. Figure 5 shows the distribution of the number of vertices and edges of the benchmarkset from the heuristic track. Of course, in the light of parameterized algorithms, not just the size of the instance, but also the actual *twinwidth* is important. All instances from the exact track that were solved during the competition have a twinwidth of at most 10 (see Figure 6 for the distribution). The instances used in the heuristic track have a larger twinwidth, but over 50% of the instances have a twinwidth of at most 60 (see the right plot in Figure 6). Both benchmarksets (all 200 instances per track) are publicly available at <https://pacechallenge.org/2023/>.

4 Participants and Results

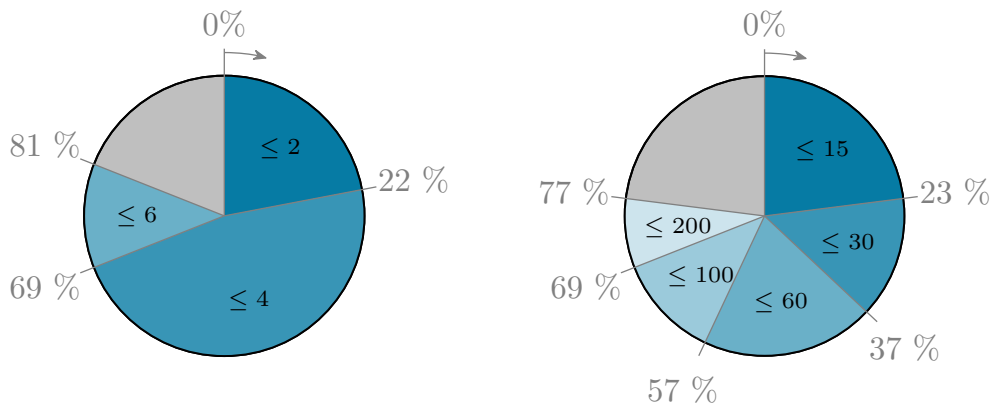
There were 13 and 10 teams that officially submitted a solution to the exact and heuristic track, respectively. There were four teams that submitted solutions to both tracks. Hence, in total there were 19 distinct teams with a total number of 73 participants representing three continents and a wide range of countries, which made this the second largest PACE challenge. The results are listed below.



■ **Figure 4** Distribution of the number of vertices (left) and number of edges (right) of the 200 instances of the PACE 2023 *exact* track. A bar at index x illustrates the number of instances that have at most x vertices (edges), but more than the previous bar contained. That is, every instance is counted in exactly one bar.




■ **Figure 5** Distribution of the number of vertices (left) and number of edges (right) of the 200 instances of the PACE 2023 *heuristic* track. A bar at index x illustrates the number of instances that have at most x vertices (edges), but more than the previous bar contained. That is, every instance is counted in exactly one bar.







■ **Figure 6** All private instances of the exact track that were solved by any solver have a twinwidth of at most 10. The left plot illustrates the twinwidth distribution these instances. The diagram shows clockwise the percentage of instances that fall below a certain twinwidth threshold. The right plot illustrates the same information for the private instances of the heuristic track (the best twinwidth found by any submission was assigned to every instance).


4.1 Ranking of the Exact Track

The ranking for the exact track is listed subsequently; We list the number of solved instances from the 100 private instances. Submissions marked with “” are student submissions after the following rules:

A student is someone who is not and has not been enrolled in a PhD program before the submission deadline. A submission is eligible for a Student Submission Award if either all its authors are students, or besides student co-author(s) there is one non-student co-author that confirms, at the moment of submission, that a clear majority of conceptual and implementation work was done by the student co-author(s).

The first three student submissions obtained a *Student Submission Award*. The submission marked with “” obtained the *Theory Award*, details follow in Section 4.5.

- Rank 1** [Hydra Prime](#) solved 36 instances in 450 minutes.  [16]
Authors Yosuke Mizutani, David Dursteler, and Blair D. Sullivan
Affiliation University of Utah
- Rank 2** [GUTHMi](#) solved 35 instances in 1257 minutes. [13]
Authors Alexander Leonhardt, Holger Dell, Anselm Haak, Frank Kammer, Johannes Meintrup, Ulrich Meyer, and Manuel Penschuck
Affiliation Goethe University Frankfurt and THM, University of Applied Sciences Mittelhessen
- Rank 3** [Touiouidth](#) solved 34 instances in 8885 minutes. [9]
Authors Gaétan Berthe, Yoann Coudert–osmont, Alexander Dobler, Laure Morelle, Amadeus Reinald, and Mathis Rocton
Affiliation LIRMM, CNRS, Université de Montpellier and Université de Lorraine, CNRS, Inria and Algorithms and Complexity Group, TU Wien
- Rank 4** [UAIC Twin Width](#) solved 34 instances in 12781 minutes.  [3]
Authors Andrei Arhire, Matei Chiriac, and Radu Timofte
Affiliation Alexandru Ioan Cuza University of Iasi, and ETH Zürich and University of Würzburg
- Rank 5** [Soapen](#) solved 33 instances in 250 minutes. [33]
Authors Christopher Weyand and Marcus Wilhelm
Affiliation KIT Karlsruhe
- Rank 6** [GBathie](#) solved 31 instances in 1183 minutes. [6]
Authors Gabriel Bathie, Jérôme Boillot, Nicolas Bousquet and Théo Pierron
Affiliation DI ENS, PSL Research University and LaBRI, Université de Bordeaux and Université Lyon
- Rank 7** [Zygosity](#) solved 29 instances in 391 minutes. [23]
Authors Emmanuel Arrighi, Petra Wolf, Pål Grønås Drange, Kenneth Langedal, Farhad Vadiée, and Martin Vatschelle
Affiliation University of Trier and University of Bergen
- Rank 8** [trex-ufmg](#) solved 27 instances in 14760 minutes.  [32]
Authors Alan Cabral Trindade Prado, Emanuel Juliano Morais Silva, Guilherme de Castro Mendes Gomes, Kaio Henrique Masse Vieira and Laila Melo Vaz Lopes
Affiliation Departamento de Ciência da Computação, Universidade Federal de Minas Gerais

- Rank 9** [tuw](#) solved 26 instances in 967 minutes. [30]
Authors André Schidler and Stefan Szeider
Affiliation Algorithms and Complexity Group, TU Wien
- Rank 10** [HeiTwin](#) solved 15 instances in 1187 minutes.  [26]
Authors Thomas Möller, Nikita-Nick Funk, Dennis Jakob and Ernestine Großmann
Affiliation Heidelberg University
- Rank 11** [A Simple Twin-width Searcher](#) solved 14 instances in 1490 minutes. [24]
Authors Alexander Meiburg
Affiliation University of California Santa Barbara
- Rank 12** [GOAT](#) solved 8 instances in 1487 minutes. [1]
Authors Adam Barla, Václav Blažej, Radovan Červený, Michal Dvořák, Dušan Knop, Josef Koleda, Jan Pokorný, Petr Štastný and Ondřej Suchý
Affiliation Faculty of Information Technology, Czech Technical University in Prague and University of Warwick
- Rank 13** [satwin](#) solved 8 instances in 35833 minutes. [20]
Authors Yinon Horev, Shiraz Shay, Sarel Cohen, Tobias Friedrich, Davis Issac, Lior Kamma, Aikaterini Niklanovits and Kirill Simonov
Affiliation School of Computer Science, The Academic College of Tel Aviv-Yaffo and Hasso Plattner Institute, University of Potsdam

4.2 Strategies Used in the Exact Track


We give a short overview on the ideas used by the top three solvers. The interested reader is referred to the corresponding papers published in the proceedings of IPEC 2023.


Winning Team. The winning solver, [Hydra Prime](#), first applies a modular decomposition on the graph and then considers all of the prime graphs and the quotient graph separately. Then, a vast number of different algorithms is applied to derive upper and lower bounds for the twinwidth of the graphs by choosing from four algorithms for the lower bounds and three algorithms for the upper bound. In addition, the solver also contains an optimal solver for trees, a branch-and-bound approach, and a SAT-based tool for the remaining cases. The main contributions are the *timeline encoding* and the *hydra decomposition*. The timeline encoding is a data structure that efficiently computes the width of a given contraction sequence by building up a set of red intervals representing the red edges during the contraction. The hydra decomposition is a refinement strategy that iteratively uses small vertex separators by splitting the graph into a set of heads and tails. This allows to find separators avoiding the heads, to contract the heads, and to join two hydras efficiently.

Runner-Up. The runner-up solver, [GUTHMI](#), is based on the branch-and-bound paradigm, but drastically reduces the potentially large branching vector using a number of heuristics. The authors obtain upper bounds using a heuristic, and lower bounds by sampling carefully chosen subgraphs that are tried to be solved exactly. Finally, to order the remaining branches, the authors use a scoring method to evaluate pairs of vertices. The scoring method aims to minimize the difference between the red degrees of vertices for which a new red edge will appear when contracting the pair, and the red degrees of vertices for which a red edge will be removed. The solver also contains several low-level optimizations such as the caching of small infeasible subproblems, the reusing of scores, and the use of appropriate data structures depending on the structure of the given graph.

Third Place. The third place solver, [Touyouidwidth](#), uses the first 15 minutes to find a good lower bound by considering the twinwidth of several carefully constructed induced subgraphs. Next, two different heuristics are run for five minutes to obtain upper bounds. Finally, the last ten minutes are used for a branch-and-bound algorithm. To reduce the search space, the authors use the observation that two vertices u and v can safely be contracted if the black neighborhood of u is a subset of the black neighborhood of v and, in addition, all neighbors (black and red) of v are contained in the red neighborhood of u (or are equal to u or v). To sort the possible branches for different pairs $\{u, v\}$, the authors first compute the size of the symmetric difference in the neighborhoods of u and v and then perform bucket-sort.

4.3 Ranking of the Heuristic Track

The ranking for the heuristic track is listed subsequently; We list the score for the 100 private instances, computed as described above. The larger the score, the better. As in the exact track, submissions marked with “

Rank 1 [GUTHM](#) got a score of [9103](#) in [30004](#) seconds.  [13]

Authors Alexander Leonhardt, Holger Dell, Anselm Haak, Frank Kammer, Johannes Meintrup, Ulrich Meyer and Manuel Penschuck

Affiliation Goethe University Frankfurt and THM, University of Applied Sciences Mittelhessen

Rank 2 [Zygotity](#) got a score of [8603](#) in [24924](#) seconds. [23]

Authors Emmanuel Arrighi, Petra Wolf, Pål Grønås Drange, Kenneth Langedal, Farhad Vadiie, and Martin Vatschelle

Affiliation University of Trier and University of Bergen

Rank 3 [Red Alert](#) got a score of [7739](#) in [22924](#) seconds. [34]

Authors Édouard Bonnet and Julien Duron

Affiliation Université Lyon

Rank 4 [HATTER](#) got a score of [6169](#) in [30003](#) seconds.  [2]

Authors Aman Jain, Sachin Agarwal, Talika Gupta, and Srinibas Swain

Affiliation Department of Computer Science and Engineering, IIIT Guwahati

Rank 5 [HeiTwin](#) got a score of [6165](#) in [30122](#) seconds.  [27]

Authors Thomas Möller, Nikita-Nick Funk, Dennis Jakob, and Ernestine Großmann

Affiliation Heidelberg University

Rank 6 [UAIC Twin Width](#) got a score of [5937](#) in [28015](#) seconds.  [3]

Authors Andrei Arhire, Matei Chiriac, and Radu Timofte

Affiliation Alexandru Ioan Cuza University of Iasi, and ETH Zürich and University of Würzburg

Rank 7 [Greetwin](#) got a score of [5749](#) in [7265](#) seconds.  [19]

Authors Jippe Hoogeveen

Affiliation Utrecht University

Rank 8 [METATWW](#) got a score of [4265](#) in [27756](#) seconds.  [25]

Authors William Bille Meyling

Affiliation University of Copenhagen

Rank 9 [twin width fmi](#) got a score of [3231](#) in [16081](#) seconds.  [31]

Authors Lucian Trepteanu, Adrian Miclaus, Alexandru Enache, and Alexandru Popa

Affiliation Faculty of Mathematics and Computer Science, University of Bucharest

Rank 10 [jbalasin](#) got a score of 342 in 2345 seconds. [4]

Authors Jonathan Balasingham, Sebastian Wild and Viktor Zamaraev

Affiliation University of Liverpool

4.4 Strategies Used in the Heuristic Track

In the following, we give a short overview on the ideas used in the top three solvers. We refer the interested readers to the corresponding papers in the proceedings of IPEC 2023 for more details.

Winning Team. The winning solver, [GUTHM](#), uses three different strategies that are applied to the connected components of the graph separately. In the first strategy, a very fast heuristic is used to obtain an initial contraction sequence. This strategy greedily selects a vertex u of minimal red degree and then identifies the best partner v according to a scoring method. The scoring method aims to minimize the difference between the red degrees of vertices x , for which a new red edge will appear when contracting u and v , and the red degrees of vertices y , for which a red edge will be removed. The efficiency of this approach stems from the observation that few applications of a two-level breadth-first search can compute this scoring function. Then, depending on the expected twinwidth of the components, one of two possible strategies is applied to improve this first solution. If the twinwidth is sufficiently low, the authors use a sweeping approach where all contractions that do not exceed the width of the contraction sequence beyond a given threshold are performed. This threshold is iteratively adapted by random sampling. If the twinwidth of the connected component is expected to be high, the authors either contract two vertices chosen by the above heuristic (i. e., contracting a vertex of minimal red degree) or apply MinHashing to identify near-twins of large red degree. To speed up their computation, the authors use a data structure that manages the deleted nodes via the union-find data structure.

Runner-Up. The runner-up solver, [Zygosity](#), first contracts all twins and then contracts all trees down to a single pendant vertex connected by a red edge. The remaining time is used for multiple random contractions. Here, the quality of a contraction depends on the maximal red degree in the graph (which should be minimized) and the size of the intersection between the neighborhoods of the two chosen vertices. As the red degree is the more important measure, the intersection size is only used as tiebreaker. Instead of choosing a pair of vertices randomly, the solver chooses one vertex u randomly and then performs a random walk starting from u . The random walk is determined by a biased coin that depends on the density of the given graph. If the graph is sparse, the random walk takes only one step, and otherwise it takes two. To optimize the running time, the authors only use arrays to avoid cache-faults when iterating over the neighborhood of a vertex.

Third Place. The third place solver, [Red Alert](#), uses a randomized approach to generate about 10,000 possible vertex pairs for the contraction from which a certain number is chosen later. The number of chosen pairs directly depends on the remaining time: The less available, the rougher the subroutine to find the pairs. If t denotes the time used in the last iteration and T the remaining time, then about $n' \cdot t/T$ pairs are chosen if the current graph contains n' vertices. To select the vertices, the authors use different approaches based on the density of the remaining graph. If the remaining graph is dense, the pair is chosen uniformly at random. In the sparse case, a first vertex is chosen uniformly at random and then one or two of its

neighbors get sampled. To evaluate a pair, the solver uses a scoring function consisting of three parts: The most crucial part measures whether the contraction decreases the maximum red degree; the second part determines whether the maximum red degree is close to the red degree of the contracted vertex; and the third part counts the total number of red edges. If the time gets short, the authors partition the vertices of the remaining graph into equally sized buckets, which are then contracted.

4.5 Theory Award

To bridge the gap between theory and practice, we gave out a theory award to encourage submissions containing new theoretical insights. The theory award has been granted to the solver [Hydra Prime](#) by Yosuke Mizutani, David Dursteler, and Blair D. Sullivan (highlighted with a “🏆” in Section 4.1). The award was primarily given due to the development of two novel ideas: the timeline encoding and hydra decompositions. The *timeline encoding* is an innovative data structure that enables the efficient computation of the width of a contraction sequence, resulting in enhanced local search capabilities. *Hydra decompositions*, conversely, are a divide-and-conquer strategy that features compact vertex separators.

5 PACE Organization

The program committee of PACE 2023 consisted of the co-chairs Max Bannach and Sebastian Berndt. During the competition, the members of the steering committee were:

- (since 2016) Holger Dell (Goethe University Frankfurt and IT University of Copenhagen)
- (since 2019) Johannes Fichte (Linköping University)
- (since 2019) Markus Hecher (MIT)
- (since 2016) Bart M. P. Jansen (chair) (Eindhoven University of Technology)
- (since 2020) Łukasz Kowalik (University of Warsaw)
- (since 2021) André Nichterlein (Technical University of Berlin)
- (since 2020) Marcin Pilipczuk (University of Warsaw)
- (since 2022) Christian Schulz (Heidelberg University)
- (since 2020) Manuel Sorge (Technische Universität Wien)

6 Conclusion and Future Editions of PACE

We thank all the participants for their impressive work, vital contributions, and patience in case of technical issues. The organizers especially thank the participants who presented their work at IPEC 2023 in the poster session or during the award ceremony. We are pleased about the large number of diverse participants and hope the PACE challenge will continue to build bridges between theory and practice. We welcome anyone interested to add their name to the mailing list on the PACE website to receive updates and join the discussion.

PACE 2024. We look forward to the next edition, which will focus on ONE-SIDED CROSSING MINIMIZATION and will be chaired by Philipp Kindermann. Detailed information will be posted on the website of the competition at pacechallenge.org.

References

- 1 Barla Adam, Blažej Václav, Červený Radovan, Dvořák Michal, Knop Dušan, Koleda Jozef, Pokorný Jan, Šťastný Petr, and Suchý Ondřej. G2oat solver for pace 2023 (twinwidth) exact track, June 2023. doi:10.5281/zenodo.7997817.
- 2 Sachin Agarwal, Aman Jain, Talika Gupta, and Srinibas Swain. Hatter: Hybrid approach to twinwidth by taming red, June 2023. doi:10.5281/zenodo.8045969.
- 3 Andrei Arhire and Matei Chiriac. Uaic_twin_width: An exact twin-width algorithm, June 2023. doi:10.5281/zenodo.8010483.
- 4 Jonathan Balasingham, Sebastian Wild, and Viktor Zamaraev. Pace-2023, May 2023. doi:10.5281/zenodo.7996417.
- 5 Tomas Balyo, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions*. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2023.
- 6 Gabriel Bathie, Jérôme Boillot, Nicolas Bousquet, and Théo Pierron. Pace 2023 - tinywidth solver, May 2023. doi:10.5281/zenodo.7991737.
- 7 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is NP-complete. In *ICALP*, volume 229 of *LIPICs*, pages 18:1–18:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- 8 Pierre Bergé, Édouard Bonnet, Hugues Déprés, and Rémi Watrigant. Approximating highly inapproximable problems on graphs of bounded twin-width. In *STACS*, volume 254 of *LIPICs*, pages 10:1–10:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- 9 Gaétan Berthe, Yoann Coudert–Osmont, Alexander Dobler, Laure Morelle, Amadeus Reinald, and Mathis Rocton. Doblalex/touiouidth: v1.0, June 2023. doi:10.5281/zenodo.8027196.
- 10 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphane Thomassé, and Rémi Watrigant. Twin-width III: max independent set, min dominating set, and coloring. In *ICALP*, volume 198 of *LIPICs*, pages 35:1–35:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 11 Édouard Bonnet, Eun Jung Kim, Stéphane Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. In *FOCS*, pages 601–612. IEEE, 2020.
- 12 Édouard Bonnet and Florian Sikora. The PACE 2018 parameterized algorithms and computational experiments challenge: The third iteration. In *IPEC*, volume 115 of *LIPICs*, pages 26:1–26:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
- 13 Holger Dell, Anselm Haak, Frank Kammer, Alexander Leonhardt, Johannes Meintrup, Meyer Ulrich, and Manuel Penschuck. GUTHM and GUTHMi: Exact and heuristic twin-width solvers, June 2023. doi:10.5281/zenodo.7996074.
- 14 Holger Dell, Thore Husfeldt, Bart M. P. Jansen, Petteri Kaski, Christian Komusiewicz, and Frances A. Rosamond. The first parameterized algorithms and computational experiments challenge. In *IPEC*, volume 63 of *LIPICs*, pages 30:1–30:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- 15 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 parameterized algorithms and computational experiments challenge: The second iteration. In *IPEC*, volume 89 of *LIPICs*, pages 30:1–30:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
- 16 David Dursteler and Yosuke Mizutani. Pace 2023 - exact, June 2023. doi:10.5281/zenodo.7996823.
- 17 M. Ayaz Dzulfikar, Johannes Klaus Fichte, and Markus Hecher. The PACE 2019 parameterized algorithms and computational experiments challenge: The fourth iteration (invited paper). In *IPEC*, volume 148 of *LIPICs*, pages 25:1–25:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- 18 Ernestine Großmann, Tobias Heuer, Christian Schulz, and Darren Strash. The PACE 2022 parameterized algorithms and computational experiments challenge: Directed feedback vertex set. In *IPEC*, volume 249 of *LIPICs*, pages 26:1–26:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

- 19 Jippe Hoogeveen. Greetwin pace 2023, June 2023. doi:10.5281/zenodo.8034100.
- 20 Yinon Horev, Shiraz Shay, Sarel Cohen, Tobias Friedrich, Davis Issac, Lior Kamma, Aikaterini Niklanovits, and Kirill Simonov. Satwin, June 2023. doi:10.5281/zenodo.8047007.
- 21 Leon Kellerhals, Tomohiro Koana, André Nichterlein, and Philipp Zschoche. The PACE 2021 parameterized algorithms and computational experiments challenge: Cluster editing. In *IPEC*, volume 214 of *LIPICs*, pages 26:1–26:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 22 Lukasz Kowalik, Marcin Mucha, Wojciech Nadara, Marcin Pilipczuk, Manuel Sorge, and Piotr Wygocki. The PACE 2020 parameterized algorithms and computational experiments challenge: Treedepth. In *IPEC*, volume 180 of *LIPICs*, pages 37:1–37:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 23 Kenneth Langedal, Emmanuel Arrighi, Pål Grønås Drange, Farhad Vadiée, Martin Vatshelle, and Petra Wolf. Zygosity pace 2023, September 2023. doi:10.5281/zenodo.8370343.
- 24 Alex Meiburg. Timeroot/twinwidth: Pace 2023, June 2023. doi:10.5281/zenodo.8045233.
- 25 William Bille Meyling. Metatww solver for pace-2023, June 2023. doi:10.5281/zenodo.8045541.
- 26 Möller, Funk, Jakob, and Großmann. Heitwin-exact - pace challenge 2023, May 2023. doi:10.5281/zenodo.7988134.
- 27 Thomas Möller, Nikita-Nick Funk, Dennis Jakob, and Ernestine Großmann. Heitwin-heuristic - pace challenge 2023, May 2023. doi:10.5281/zenodo.7986572.
- 28 Networks project, 2017. URL: <https://www.thenetworkcenter.nl>.
- 29 André Schidler and Stefan Szeider. A SAT approach to twin-width. In *ALLENEX*, pages 67–77. SIAM, 2022.
- 30 Andre Schidler and Stefan Szeider. Computing twin-width with branch & bound - PACE Submission, June 2023. doi:10.5281/zenodo.8033459.
- 31 Lucian Trepteanu. luciantrepteanu/pace2023: pace-2023, May 2023. doi:10.5281/zenodo.7991297.
- 32 Kaio Vieira, alanctprado, emanuel juliano morais silva, and Laila Melo Vaz Lopes. emanueljuliano/pace2023: pace-2023, June 2023. doi:10.5281/zenodo.8045380.
- 33 Christopher Weyand and Marcus Wilhelm. soap1, May 2023. doi:10.5281/zenodo.7989779.
- 34 Bonnet Édouard and Duron Julien. Redalert, June 2023. doi:10.5281/zenodo.8079499.

PACE Solver Description: Hydra Prime

Yosuke Mizutani ✉ 

University of Utah, Salt Lake City, UT, USA

David Dursteler ✉ 

University of Utah, Salt Lake City, UT, USA

Blair D. Sullivan ✉ 

University of Utah, Salt Lake City, UT, USA

Abstract

This note describes our submission to the 2023 PACE Challenge on the computation of twin-width. Our solver **Hydra Prime** combines modular decomposition with a collection of upper- and lower-bound algorithms, which are alternatingly applied on the prime graphs resulting from the modular decomposition. We introduce two novel approaches which contributed to the solver's winning performance in the Exact Track: *timeline encoding* and *hydra decomposition*. Timeline encoding is a new data structure for computing the width of a given contraction sequence, enabling faster local search; the hydra decomposition is an iterative refinement strategy featuring a small vertex separator.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Twin-width, PACE 2023

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.36

Supplementary Material *Software*: <https://github.com/TheoryInPractice/hydraprime>
archived at `swh:1:dir:eb6788de444b4d5277f0a400dea4a1affa0e6df7`

Funding *Blair D. Sullivan*: This work was supported in part by the Gordon & Betty Moore Foundation under award GBMF4560.

1 Introduction

The goal of the 2023 Parameterized Algorithms & Computational Experiments (PACE) Challenge (<https://pacechallenge.org/2023/>) was to compute twin-width [2], a structural graph parameter which measures how close a given graph is to a *cograph* – a graph which can be reduced to a single vertex by repeatedly merging (*contracting*) pairs of *twins* – vertices with identical open neighborhoods. More generally, twin-width measures the minimum number of “mistakes” made in such a process when the pairs being contracted are no longer twins. If u and v are being merged, we say uy becomes a *red edge* if y is a neighbor of u but not v (and analogously for edges vy). The width of a contraction sequence is then the maximum number of red edges incident to any vertex (*red degree*) at any time during the process, and the twin-width of a graph is the minimum width of all valid contraction sequences. While graphs with bounded twin-width admit many FPT algorithms, computing the parameter is NP-hard, and prior to the PACE challenge its exact computation had remained impractical even on relatively small graphs.

Most twin-width solvers naturally begin by removing twins, as all groups of twins can be collapsed without incurring any red edges, making it a safe operation. In **Hydra Prime**, we employ a stronger notion of this via *modular decompositions* [4], which decompose a graph into a hierarchy of maximal modules. A key property of these decompositions is that the twin-width of the original graph is exactly the maximum of the twin-width of the twin-free, prime quotient graphs (Theorem 3.1 from [5]). We thus begin by running a



© Yosuke Mizutani, David Dursteler, and Blair D. Sullivan;
licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 36; pp. 36:1–36:5

Leibniz International Proceedings in Informatics



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

re-implemented linear-time modular decomposition solver based on [6], then process each prime graph separately, maintaining a global lower bound. If a prime graph is a tree, we run `PrimeTreeSolver`, otherwise we run a series of lower- and upper-bound algorithms (listed at the end of this section) alternatively until the bounds match, from the quickest algorithms to the slowest. Those algorithms marked with (*) use a SAT solver as a subroutine; the implementation submitted to PACE uses the Kissat solver [1].

Algorithm List.

- Exact algorithms
 - `PrimeTreeSolver`: Linear-time exact solver for trees without twins.
 - `BranchSolver`: Brute-force solver equipped with caching mechanism and reduction rules.
 - `DirectSolver` (*): SAT-based solver implementing the relative encoding presented in [5].
- Lower-bound algorithms
 - `LBGreedy`: Greedily removes a vertex u from the graph G such that $|\Delta(u, v)|$ is minimized for some v . Reports the maximum value of $\min_{u, v \in V(G), u \neq v} |\Delta_G(u, v)|$.
 - `LBCore` (*): SAT-based algorithm to find $\max_{S \subseteq V(G)} \min_{u, v \in S, u \neq v} |\Delta_{G[S]}(u, v)|$.
 - `LBSample`: Sampling-based algorithm. Finds a connected induced subgraph G' of G by random walk and computes the exact or lower-bound twin-width of G' .
 - `LBSeparate` (*): Similar to `LBSample`, but uses the hydra decomposition to find an induced subgraph to check for the lower-bound.
- Upper-bound algorithms
 - `UBGreedy`: Iteratively contract a vertex pair minimizing the weak red potential.
 - `UBLocalSearch`: Using the timeline encoding, we make small changes to the elimination ordering and the contraction tree to see if there is a better solution.
 - `UBSeparate` (*): Iterative refinement algorithm using the hydra decomposition.

In this paper we focus on two additional contributions to solving twin-width which are used in the `LocalSearch` and `Separate` algorithms implemented in `Hydra Prime`: “timeline encoding” and “hydra decomposition”. *Timeline encoding* is a novel data structure which enables faster computation of twin-width by storing red “sources” and “intervals” indicating the cause and window of each red edge. In the `Separate` upper- and lower-bound algorithms, we introduce *hydra decomposition*, an iterative refinement strategy using small vertex separators. After defining necessary notation, we briefly describe these in Sections 2 and 3, respectively. Additional details are in the appendix available on the code repository.

Notation. We follow standard graph-theoretic notation (e.g. found in [3]), the original definition of twin-width [2], and terminology introduced by Schidler and Szeider [5]. Refer to [4] and [6] for the definitions of a *module*, *modular decomposition*, a *prime* graph, etc. We write $u \leftarrow v$ when vertex v is contracted into vertex u . Given a trigraph G , the *weak red potential* of $u, v \in V(G), u \neq v$ is the red degree of u after contraction $u \leftarrow v$. We further define the *unshared neighbors* of vertices u and v , denoted by $\Delta(u, v)$ as $N(u) \Delta N(v) \setminus \{u, v\}$, where Δ denotes the symmetric difference of two sets. We write $[n]$ for $\{1, \dots, n\}$.

2 Timeline Encoding

In this work we developed the *timeline encoding*, a data structure to compute the width of a given contraction sequence. An instance of the timeline encoding stores the following data:

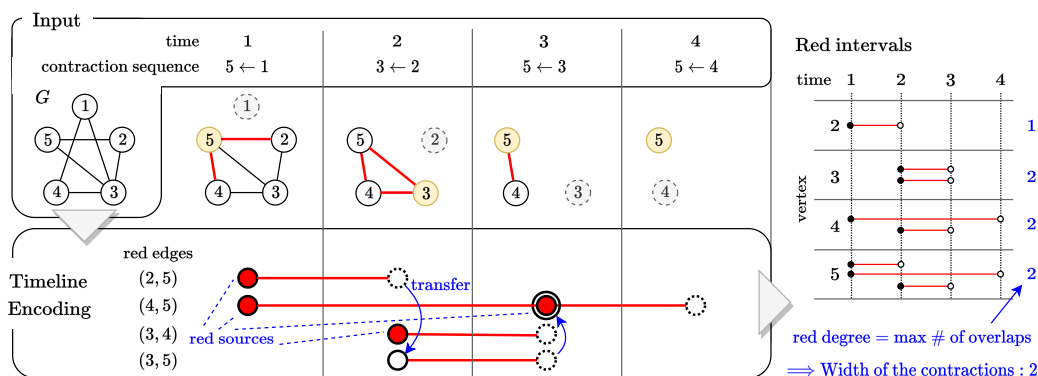


Figure 1 An illustration of the timeline encoding given a graph and its contraction sequence. Vertex labels show the elimination ordering. For each time i with contraction $j \leftarrow i$ ($i < j$), we create red sources $\{k, j\}$ for every $k \in \Delta_{>}(j, i)$, which determines red intervals $[i, \min\{k, j\}]$ that will then disappear or transfer at time $\min\{k, j\}$. The red degree corresponds to the number of overlaps of red intervals aggregated by vertices, and its maximum value is the width of the contraction sequence.

- G : input graph with n vertices.
- $\phi : V(G) \rightarrow [n]$: bijection that encodes an elimination ordering (vertex v is eliminated at time $\phi(v)$ if $\phi(v) < n$).
- $p : [n - 1] \rightarrow [n]$: encoding of a contraction tree. For $i < j$, $p(i) = j$ if vertex $\phi^{-1}(i)$ is merged into vertex $\phi^{-1}(j)$ (i.e. j is the *parent* of i in the contraction tree).

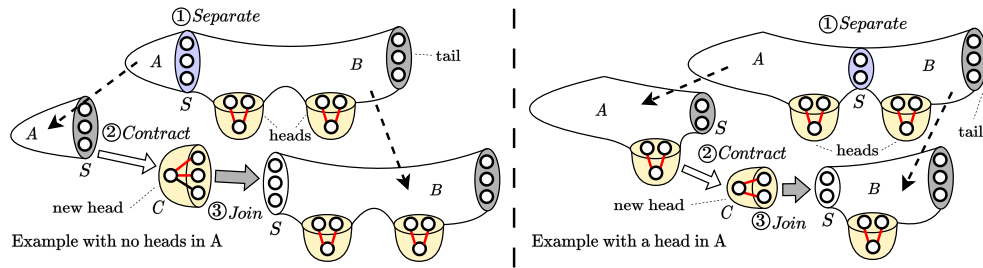
For internal data structures, we introduce a few terms. First, define $\Delta_{>}(j, i) := \{\phi(w) \mid w \in \Delta(\phi^{-1}(i), \phi^{-1}(j)), \phi(w) > i\}$. Then, the *red sources* at time t are a set of red edges introduced at time t , defined as $\{\{p(t), k\} \mid k \in \Delta_{>}(p(t), t)\}$. Red sources determine the *red intervals* – non-overlapping, continuous intervals where an edge is red, defined as follows: for $i < j$, red source (i, j) at time t creates an interval $[t, i)$ (red edge ij disappears at time i). If $p(i) \neq j$, then we recurse this process as if red source $\{p(i), j\}$ was created (red edge ij transfers to $\{p(i), j\}$), as illustrated in Figure 1.

Now we aggregate red intervals by vertices. We maintain a *multiset* of intervals for each vertex such that a red interval of an edge accounts to its both endpoints. The maximum number of the overlaps of such intervals gives the maximum red degree at a vertex over time. Finally, we obtain the width of the contraction sequence by taking the maximum of the red degrees over all vertices.

A key observation is that we can dynamically compute the number of overlaps of a multiset of intervals efficiently with a balanced binary tree (e.g. modification in time $\mathcal{O}(\log n)$, getting the maximum number of overlaps in $\mathcal{O}(1)$, etc.). For local search, we implemented methods for modifying a contraction tree and also updating a bijection ϕ .

3 Hydra Decomposition

We also implemented an iterative refinement strategy which we term *hydra decomposition*, based on finding a small vertex separator. A *hydra* is a structured trigraph which consists of a (possibly empty) set of *heads* and a (possibly empty) vertex set *tail*. Each head is a set of vertices containing one *top vertex* and a nonempty set of *boundary vertices*. The neighbors of the top vertex must be a subset of the boundary vertices. All red edges in the trigraph must be incident to one of the top vertices. Heads must be vertex-disjoint, but the tail may contain boundary vertices (but not a top vertex). A *compact hydra* is a hydra consisting of its tail and



■ **Figure 2** Structure of the hydra and two examples of performing a round of hydra decomposition.

one extra vertex, with no heads. A head of a hydra H can additionally be viewed as a compact hydra C , where the boundary vertices of H are the tail of C . Now that we have defined the parts of a hydra, we will now show the operations performed in hydra decomposition:

1. *Separate*: partitions the vertices of a hydra into three parts S, A, B such that S separates A from B . The part S should not contain any vertices from the heads, and any tail vertices cannot be in A . Figure 2 shows two ways of choosing a separator S of a hydra.
2. *Contract*: takes a hydra and contracts all vertices but its tail. The output is a contraction sequence and the resulting compact hydra.
3. *Join*: combines a compact hydra C and another hydra H such that $V(C) \cap V(H)$ is the tail of C . The output is the union of C and H , where the heads and tail of H remain and C becomes an additional head.

We now present a description of `UBSeparate`. Given a graph H and a target width d for a contraction sequence, `UBSeparate` runs *contract* on the original graph without any heads or tails. The *contract* operation works as follows: If the input H is small enough, or a vertex separator of size at most d is not found, we directly search for a contraction sequence of width at most d for all but tail vertices, which can be done by modified `UBGreedy` and other exact algorithms. Otherwise, we perform *separate* to obtain a partition S, A, B . We recursively call *contract* with $H[A \cup S]$ with S being the tail. Then, we have a contraction sequence s_1 and a compact hydra C . Next, we *join* C with $H[B \cup S]$ and obtain a hydra H' . Notice that the tail of C must be S . We again call *contract* with H' and get a contraction sequence s_2 , resulting in a compact hydra C' with the original tail of H . Finally, C' is returned along with the concatenation of s_1 and s_2 as the result of the original *contract* operation.

A key observation is that since red edges reside only in heads and the size of separators are bounded by d , the red degree of a hydra is also upper-bounded by d , which helps construct a d -contraction sequence part by part. For $d = 1$ we use a linear-time algorithm to find a vertex separator, or a cut vertex (articulation point); for $d \geq 2$, we instead call a SAT solver.

References

- 1 Armin Biere and Mathias Fleury. Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022. In *Proc. of SAT Competition 2022 – Solver and Benchmark Descriptions*, volume B-2022-1 of *Department of Computer Science Series of Publications B*, pages 10–11, 2022.
- 2 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 601–612, 2020.
- 3 Reinhard Diestel. *Graph Theory*. Springer Publishing Company, Inc., 5th edition, 2017.
- 4 Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.


- 5 André Schidler and Stefan Szeider. A SAT Approach to Twin-Width. In *2022 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 67–77, 2022.
- 6 Marc Tedder, Derek Corneil, Michel Habib, and Christophe Paul. Simple, linear-time modular decomposition, 2008. arXiv:0710.3901.

PACE Solver Description: Exact (GUTHMI) and Heuristic (GUTHM)

Alexander Leonhardt ✉
Goethe University Frankfurt, Germany

Anselm Haak ✉
Goethe University Frankfurt, Germany

Johannes Meintrup ✉ 
THM, University of Applied Sciences
Mittelhessen, Gießen, Germany

Manuel Penschuck ✉ 
Goethe University Frankfurt, Germany

Holger Dell ✉ 
Goethe University Frankfurt, Germany

Frank Kammer ✉ 
THM, University of Applied Sciences
Mittelhessen, Gießen, Germany

Ulrich Meyer ✉ 
Goethe University Frankfurt, Germany

Abstract

Twin-width (tww) is a parameter measuring the similarity of an undirected graph to a co-graph [3]. It is useful to analyze the parameterized complexity of various graph problems. This paper presents two algorithms to compute the twin-width and to provide a contraction sequence as witness. The two algorithms are motivated by the PACE 2023 challenge, one for the exact track and one for the heuristic track. Each algorithm produces a contraction sequence witnessing (i) the minimal twin-width admissible by the graph in the exact track (ii) an upper bound on the twin-width as tight as possible in the heuristic track.

Our heuristic algorithm relies on several greedy approaches with different performance characteristics to find and improve solutions. For large graphs we use locality sensitive hashing to approximately identify suitable contraction candidates. The exact solver follows a branch-and-bound design. It relies on the heuristic algorithm to provide initial upper bounds, and uses lower bounds via contraction sequences to show the optimality of a heuristic solution found in some branch.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases PACE 2023 Challenge, Heuristic, Exact, Twin-Width

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.37

Supplementary Material *Software (Source Code)*: https://github.com/manpen/twin_width
archived at `swh:1:dir:b01c88158d6a76db60eade64dbb8b9f8121127`
Software (Archived Source Code): <https://zenodo.org/record/7996074>

Funding *Johannes Meintrup*: Funded by the Deutsche Forschungsgemeinschaft (DFG) – 379157101.
Manuel Penschuck: Funded by the Deutsche Forschungsgemeinschaft (DFG) – ME 2088/5-1 (FOR 2975 – Algorithms, Dynamics, and Information Flow in Networks).

Acknowledgements The order of authors is alphabetical with the exception that we moved Alexander Leonhardt to the front as he contributed more than a proportional amount to the heuristic solver.

1 Introduction

Twin-width has been a recent focus of researchers in the field of parameterized complexity. It was first introduced by Bonnet et al. [3] in the context of model-checking, and further results by Bonnet et al. followed in a series of publications. Previously, there has been only one work on exactly computing twin-width in practice, which is based on a SAT-formulation [5].



© Alexander Leonhardt, Holger Dell, Anselm Haak, Frank Kammer, Johannes Meintrup, Ulrich Meyer, and Manuel Penschuck;

licensed under Creative Commons License CC-BY 4.0

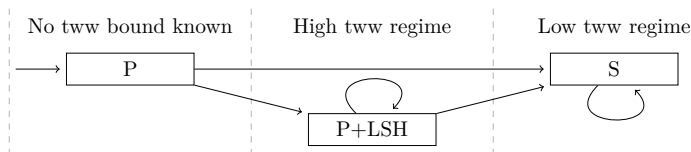
18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 37; pp.37:1–37:7

Leibniz International Proceedings in Informatics



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



■ **Figure 1** Heuristic strategies. The solver always starts with **P** to *quickly* upper bound a connected component's tww. It then attempts to find better solutions with **S** or **P+LSH** based on the best known tww bound. Hence, the solver may only move to the right in the figure.

To our knowledge, there has been no work on heuristics for obtaining contraction sequences of low width. It is currently unknown if there exists an FPT-approximation algorithm for computing the twin-width of a graph, and it is NP-complete to decide whether the twin-width of a graph is at most four [1]. Graphs of twin-width one can be recognized in polynomial time [2], for graphs of twin-width two or three the question remains open.¹

2 Preliminaries

A *tri-graph* $G = (V, B, R)$ is an undirected graph $(V, B \cup R)$ where the edge set is bi-partitioned into so-called *black* edges B and *red* edges R . Let $N_B(v)$, $N_R(v)$ and $N(v)$, denote the open neighborhood of v in the graphs (V, B) , (V, R) , and $(V, B \cup R)$, respectively, and $N[v]$ denote its closed neighborhood in the graph $(V, B \cup R)$. Furthermore, denote by $\text{blkDeg}(v)$ its black degree $|N_B(v)|$ and by $\text{redDeg}(v)$ its red degree $|N_R(v)|$. The *2-neighborhood* of v is defined as the set of v 's neighbors and their neighbors (excluding v itself).

Given a tri-graph, the *contraction* of node v into u removes v and replaces all edges incident to u by the following new edges: A black edge to any node that had a black edge to both u and v . A red edge to any node that had a red edge to either u or v or was only adjacent to either u or v , but not both. Here, self-loops on u are not added. Intuitively, this can be seen as merging u and v and coloring all differences in their incident edges red.

Given a simple and undirected graph $G = (V, E)$, a *contraction sequence* for G is a list of $n - 1$ contractions transforming the all black tri-graph (V, E, \emptyset) into a tri-graph with a single node. The *width* of such a contraction sequence is the maximal red degree of any node in any of the intermediate graphs obtained by only applying a prefix of the sequence. The *twin-width* of a graph G is the minimal width of any contraction sequence for G .

3 GUTHM: Greedily Unifying Twins with Hashing and More

Our heuristic solver GUTHM is greedy in nature as it repeatedly selects the locally best contractions to carry out. Based on various heuristics locally suboptimal contractions may be selected, though. Since there are $\Theta(|V|^2)$ possible merges to consider at each step, a naive greedy approach is prohibitively slow on large graphs.

As summarized in Figure 1, we use three strategies based on the information available to derive a contraction sequence. A strategy is only changed after the input graph has been fully contracted. Based on the information gathered either a new strategy is employed or the same strategy is used again with a different seed.

¹ Open problems for twin-width: <http://perso.ens-lyon.fr/edouard.bonnet/openQuestions.html>

- **(P) Priority based:** *Quickly* constructs a somewhat good initial contraction sequence.
- **(S) Sweeping based:** Primarily used as second stage for low twin-width graphs since its runtime depends on characteristics exhibited by this kind of graphs.
- **(P+LSH) Priority with support for locality sensitive hashing:** Primarily used as second stage for high twin-width graphs extending the greedy approach with locality sensitive hashing.

If the input is disconnected, each connected component (CC) can be processed in isolation. Then, node contractions within a CC preserve connectivity. We start by processing each CC using the solver **P**. After establishing a first trivial contraction sequence, we repeatedly attempt to improve the partial solution for a CC with the currently largest twin-width bound. In the following subsections unless otherwise stated a “best” contraction always refers to a contraction minimizing the score given in Section 3.2.

3.1 P: Priority based solver

The priority based solver **P** always selects a node v with the smallest red degree. It then identifies the best contraction partner for this node (see Section 3.2 for the scoring function). To accelerate the second step, we devised a fast method to find all nodes in the 2-neighborhood of v and rank each by the similarity between its neighbors and v 's neighbors:

► **Observation 1.** *A two-level BFS can be adjusted to calculate the symmetric difference between v 's neighborhood and the neighborhood of the nodes in its 2-neighborhood.*

After calculating the score for all nodes in the 2-neighborhood of v as depicted in Figure 2, one can calculate the cardinality of the symmetric difference of the neighborhoods of u and v

$$\text{SD}(u, v) = \begin{cases} |N[v] \oplus N[u]| = |\deg(v)| + |\deg(u)| - 2 \cdot \alpha_u, & \text{if } u \text{ and } v \text{ are adjacent} \\ |N(v) \oplus N(u)| = |\deg(v)| + |\deg(u)| - 2 \cdot \alpha_u, & \text{otherwise,} \end{cases}$$

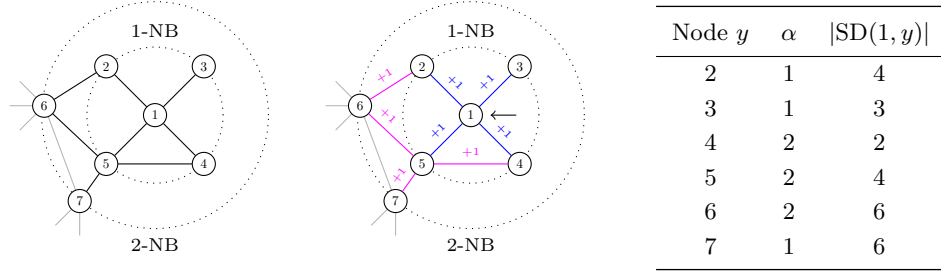
where α_u is the number of visited incoming edges of u during a two-level BFS from v .

► **Lemma 2.** *The calculation of all symmetric neighbor set differences in a graph in the 2-neighborhood of an arbitrary node v is possible in time $\mathcal{O}(|E_{2\text{-NB}}(v)|)$. Here, $|E_{2\text{-NB}}(v)|$ is the total number of edges in the 2-neighborhood.*

Proof. Every visited incoming edge of a node u during a two-level BFS traversal directly corresponds to a shared neighbor with v . The number α_u therefore measures the number of shared neighbors between u and v , which is $|N[u] \cap N[v]|$ for direct neighbors and $|N(u) \cap N(v)|$ otherwise. Since a shared neighbor is present in both u 's and v 's neighborhood, a factor of two is necessary to calculate the symmetric difference of the neighborhoods. The time to traverse all edges in a 2-neighborhood is bounded by $\mathcal{O}(|E_{2\text{-NB}}(v)|)$. ◀

► **Observation 3.** *This approach can be extended to a tri-graph by executing it twice; once on the black induced subgraph and once on the red one. Now a single two-level BFS which only traverses the paths which consist of different colored edges, red-black and black-red, suffices to correct the overestimation provided by the sum of the first two BFS applications.*

The above technique is used to find and rank all possible contraction partners by their neighbor set similarity. After finding and scoring the top- k partners of v a best one is selected as the next contraction partner. If the twin-width is increased due to a contraction involving v , the solver might postpone contracting this node to a later point in time, and, instead, selects the next best candidate.



■ **Figure 2** Application of BFS to order all neighbors in the 2-neighborhood by their symmetric neighbor set difference with the source node. Grey edges depict edges from BFS level ≥ 3 while blue and magenta edges depict edges from BFS level 1 and 2 respectively.

After a successful merge involving v , the solver contracts the newly created leaves in the direct vicinity of v (if there are multiple). If there are further “good” contractions involving node v , they are executed as well before selecting a new v . We continue until the intermediate tri-graph is sufficiently small to run an exhaustive final stage solver. The final stage solver considers *all* possible contractions and greedily selects the best contraction at any time.

3.2 Move selection

If the contractions do not increase the twin-width of the current intermediate tri-graph this heuristic is employed, otherwise the next contraction is greedily chosen as a contraction with the smallest increase in twin-width. We say the *best* contractions are those which minimize the following scoring function:

$$\text{score}(u, v) = \sum_{(v, x) \in R_{\text{new}}} (\text{redDeg}(x) + 1) - \sum_{(v, y) \in R_{\text{rem}}} \text{redDeg}(y),$$

where R_{new} and R_{rem} denote the sets of red edges the contraction of (u, v) introduces and removes, respectively.

3.3 S: Sweeping based solver

The solver **S** sweeps over all nodes in the graph and carries out contractions that do not increase the twin-width above a certain threshold. On one hand, the threshold helps to guide the solver. On the other hand, it also speeds up the computation as it reduces the number of contraction candidates to consider. As such, we only use this strategy on graphs with a sufficiently low upper bound on the twin-width (previously established by **P** or **P+LSH**). On top, we employ random sampling to establish an estimate of the new threshold at the beginning of every round further curbing execution speed at the cost of accuracy. In subsequent calls to this solver the threshold and the random samples are continuously tweaked to improve accuracy at the cost of execution speed.

3.4 P+LSH: Priority with support for locality sensitive hashing

If the solver **P** found a contraction sequence witnessing a high twin-width graph, it is unlikely that the sweeping solver **S** can process this graph within the time budget. Therefore, we attempt to improve the solution quality of the fast solver **P** by adding global information collected via MinHashing [4].

- **Local information:** Just as solver **P**, we initially select the next vertex v based on the smallest red degree. The local information is now derived from the possible contractions involving v and node u selected from the 2-neighborhood of v .
- **Global information:** The local perspective, however, fails to identify near-twins with large red degrees. To overcome this restriction, we use a scheme based on MinHashing to identify “almost twins” by approximately finding a solution for the closest pair problem. From all similar pairs obtained, we order the pairs by their number of collisions in all hash tables and the maximum red degree of the nodes in the pair.

Using MinHashing, we approximately find near twins even in large graphs. Despite the obvious benefits of using MinHashing, we empirically found it is still important to retain the initial approach of selecting a vertex with the lowest red degree and considering contractions involving this vertex. This is because the performance of MinHashing strongly depends on the choice of tuning parameters, which we cannot efficiently adapt during execution due to time and memory constraints. From the ordered similar pairs we only consider the top- k pairs and select a pair that minimizes the score given in Section 3.2. When the graph becomes sufficiently small, we again switch to an exhaustive final stage solver.

Updates of MinHash-based all-pair nearest-neighbor

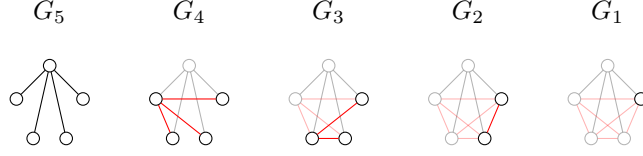
After a contraction of v into u , only neighbors of the survivor u need to be updated in the MinHash table. Observe that at most 2 adjacent edges are changed for any node w in $N(u) = N_B(u) \cup N_R(u)$. Thus, we can preserve the data structure even for large graphs, since the probability of needing to update a neighbor is inversely proportional to its degree.

3.5 Graph reconstruction

Since the heuristic track requires processing of large graphs with only limited memory, we devised a data structure allowing fast reversals of previous contractions. It uses $\mathcal{O}(|V| + |E|)$ memory and has a runtime for reverting contractions proportional to the combined degree of the involved nodes for all practical purposes. As illustrated in Figure 3, any data structure keeping track of all previously existing edges requires $\Omega(|V|^2)$ memory in the worst case. Therefore, any algorithm trying to achieve better memory bounds has to delete previously existing edges at some point, making the reconstruction considerably harder. Other algorithms (*Contraction tree based reconstruction*), which trivially achieve these memory bounds, struggle to reconstruct the previous state with a good time complexity. We present a data structure achieving an acceptable memory consumption in addition to a fast reversal time.

Our data structure keeps track of any deleted node using a union-find data structure. Every red edge keeps track of the node whose removal induced the color switch to red. This value is transferred alongside the edge when it is transferred to a new node. In case of a conflict, that is, when the contracted nodes both have a red edge to the same neighbor, the red edge from the survivor is taken. Furthermore, for every contraction, we use a stack to store certain tokens that allow us to reconstruct the previous state of the graph. Upon a contraction of v into u , v and all incident edges are deleted. For every neighbor $w \in N(v) \cup N(u)$, let $(w_u, w_v) \in \{\emptyset, r, b\}^2$ describe the relationship of u, v with the corresponding node w with $w_x = \emptyset$ if $w \notin N(x)$, $w_x = r$ if $w \in N_R(x)$, and $w_x = b$ otherwise.

From now on the special relationships $w^+ = (w_u, w_v) = (\emptyset, r)$ and $w^- = (w_u, w_v) = (r, \emptyset)$ are distinguished from all other possible relationships. For every neighbor in $N(u) \cup N(v)$ the following case distinction is made alongside the rules to be able to revert a contraction in case of a particular relationship:



■ **Figure 3** Depicts the $\Theta(|V|^2)$ different edges present during the whole lifetime of the graph even though it never exceeds $|E| = \Theta(|V|)$ edges at any point in time.

- $(w_u, w_v) = (\emptyset, b)$ - Put (τ_-, w) on the stack. *Reverse:* $N_R(u) = N_R(u) \setminus \{w\}$.
- $(w_u, w_v) = (b, \emptyset)$ - Put (τ_+, w) on the stack. *Reverse:* $N_R(u) = N_R(u) \setminus \{w\}$ and $N_B(u) = N_B(u) \cup \{w\}$.
- $(w_u, w_v) = (r, b)$ - Put $(\tau_<, w)$ on the stack. *Reverse:* $N_B(v) = N_B(v) \cup w$.
- $(w_u, w_v) = (b, r)$ - Put $(\tau_>, w)$ on the stack. *Reverse:* $N_B(v) = N_B(v) \setminus \{w\}$ and $N_R(v) = N_R(v) \cup \{w\}$.
- $(w_u, w_v) = (r, r)$ - Put $(\tau_=, w)$ on the stack. *Reverse:* $N_R(v) = N_R(v) \cup \{w\}$.

Without going further into detail, every token on the stack either corresponds to a black edge turning red (happens at most **once** for every black edge), or to two edges having a conflict, which necessarily deletes one edge and can therefore only happen $\mathcal{O}(|E|)$ times.

► **Observation 4.** *The cases w^+, w^- correspond to cases where exactly one of the nodes is connected by a red edge to a neighbor w while the other one is not. Determining the source of the red edge is equivalent to looking up the node responsible for the edge turning red in the union-find data structure, which is in the same set as one of the last contraction's nodes.*

By Observation 4 the only potentially memory-wise unbounded cases left can be handled by querying the union-find data structure. Combining the case distinction and the observation above, a contraction can be reverted with the stated memory and time bounds.

► **Lemma 5.** *There exists a data structure using $\mathcal{O}(|E| + |V|)$ memory and supporting the reversal of the last contraction up to the initial graph in $\mathcal{O}(\alpha(|V|) \cdot (\deg(v) + \deg(u)))$ expected time for the reversal of any contraction (u, v) where $\alpha(\cdot)$ is the inverse Ackermann function.*

Proof. The union-find data structure needs at most $\mathcal{O}(|V|)$ memory for the currently deleted nodes. Since all edges are removed from the graph upon a contraction and the number of edges cannot increase by contracting two nodes, the memory needed to store the graph never exceeds $\mathcal{O}(|V| + |E|)$. We only add an item to the stack used to distinguish the different removals from the graph, if two edges are reduced to a single edge or a black edge turns red. Therefore, this can happen at most $|E|$ times. This leads to a total memory consumption dominated by $\mathcal{O}(|V| + |E|)$. Since any reinsertion of edges can be done in linear time, the worst case is having to look up every edge in the union-find data structure. In this case, the running time is bounded by $\mathcal{O}(\alpha(|V|) \cdot (\deg(v) + \deg(u)))$. ◀

The implementation of the heuristic solver often uses approximations of the described techniques to stay within the imposed time bounds.

4 GUTHMI: Germanely Unifying Twins with Hashing and Meticulous Inspection

Our exact solver GUTHMI follows the branch-and-bound paradigm. At each level of the recursion, the algorithm potentially follows $\Theta(|V|^2)$ branches which is prohibitively expensive even for small graphs. We use several heuristics to reduce the search space:

- *Safe contraction of twins*: Before processing an (intermediate) graph, we search for exact twins and contract them. For performance reasons, several rules (e.g., for multiple leaves on the same node, general twins, etc.) are dedicated to this idea.
- *Upper bounds*: Before engaging the exact algorithm, we obtain an upper bound from a heuristic solution. This bound may be repeatedly improved during the runtime of the exact solver. It allows us to prune branches that cannot improve the current best solution.
- *Branching order*: We use scoring methods similar to Section 3 to descend into most promising branches first. Thereby, we often discover improvements quite early in the process. These improved upper bounds then translate into even more aggressive pruning.
- *Lower bounds*: Given a graph $G = (V, E)$ and an induced subgraph G' of G , the twin-width of G is bounded from below by the twin-width of G' . Based on this, we (non-uniformly) randomly sample subgraphs and attempt to solve them exactly in the first 20 seconds of the execution. In many cases (esp. for small graphs with low twin-width), this suffices to prove the optimality of the heuristic solution. Otherwise, we compare any improved solution to the lower bound, which, upon matching allows us to terminate early.
- *“Conditional lower bounds”*: We pass the maximum red degree of the current contraction sequence candidate down the recursion. Amongst others, this allows us to quickly prune subtrees if an improved solution is found.
- *Infeasibility caching*: Since the upper bound is non-increasing during an execution, a subproblem that cannot improve the upper bound at one point in time, cannot do it later. For this reason we cache small infeasible subproblems to avoid recomputing them later. Several implementation details helped to shave-off constant factors.
- While descending into the recursion tree, we attempt to reuse as much of the meta-information (e.g., branch scoring) from the parent as possible.
- We compile dedicated solvers for different graph size ranges using meta-programming. For instance, small graphs are kept in bit matrices on stack, while larger graphs are escalated on to the heap. This way, most set operation implementations are bit-parallel.

5 Conclusion

We presented two solvers for approximately and exactly finding contraction sequences of small width. The solvers are able to handle arbitrary graph classes with varying success, they are generic in that sense. Further research directions might involve using the weighted Jaccard similarity such that one can directly approximate the score in Section 3.2.

References

- 1 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is np-complete. In *ICALP 2022*, volume 229 of *LIPICs*, pages 18:1–18:20, 2022. doi:10.4230/LIPICs.ICALP.2022.18.
- 2 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. *Algorithmica*, 2022. doi:10.1007/s00453-022-00965-5.
- 3 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.
- 4 A. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, page 21, USA, 1997. IEEE. doi:10.1109/SEQUEN.1997.666900.
- 5 André Schidler and Stefan Szeider. A SAT approach to twin-width. In *ALENEX 2022*, pages 67–77. SIAM, 2022. doi:10.1137/1.9781611977042.6.

PACE Solver Description: Touiwidth

Gaétan Berthe ✉ 

LIRMM, CNRS, Université de Montpellier, France

Yoann Coudert–Osmont ✉

Université de Lorraine, CNRS, Inria, LORIA, France

Alexander Dobler ✉ 

Algorithms and Complexity Group, TU Wien, Austria

Laure Morelle ✉ 

LIRMM, CNRS, Université de Montpellier, France

Amadeus Reinald ✉ 

LIRMM, CNRS, Université de Montpellier, France

Mathis Rocton ✉ 

Algorithms and Complexity Group, TU Wien, Austria

Abstract

We describe Touiwidth, a twin-width solver for the exact-track of the 2023 PACE Challenge: TWIN WIDTH. Our solver is based on a simple branch and bound algorithm with search space reductions and is implemented in C++.

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

Keywords and phrases Twinwidth, Pace Challenge

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.38

Supplementary Material *Software (source code)*: <https://doi.org/10.5281/zenodo.8027195>

Funding *Gaétan Berthe*: Supported by the ANR project ELIT (ANR-20-CE48-0008-01).

Alexander Dobler: Supported by the Vienna Science and Technology Fund (WWTF) under grant [10.47379/ICT19035].

Laure Morelle: Supported by the ANR project ELIT (ANR-20-CE48-0008-01) and the French-German Collaboration ANR/DFG Project UTMA (ANR-20-CE92-0027).

Amadeus Reinald: Supported by the ANR project DIGRAPHS (ANR-19-CE48-0013).

Mathis Rocton: Supported by the European Union’s Horizon 2020 research and innovation COFUND programme (LogiCS@TUWien, grant agreement No 101034440), and the FWF Science Fund (FWF project Y1329).

1 Introduction

Twin-width is a graph parameter introduced in 2020 by Bonnet, Kim, Thomassé and Watrigant [3]. The PACE 2023 Exact Challenge asks to compute the exact twin-width of a dataset of 200 graphs. It is known that deciding whether a graph has twin-width 4 is NP-complete [1], and no approximation algorithms are known. On the exact front, there exists a polynomial-time algorithm for deciding whether a given graph has twin-width 1 [2], and the complexity remains open for twin-width 2 and 3. The exact computation of twin-width has already been tackled using SAT solver methods in [4].



© Gaétan Berthe, Yoann Coudert–Osmont, Alexander Dobler, Laure Morelle, Amadeus Reinald, and Mathis Rocton;

licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 38; pp. 38:1–38:4



Leibniz International Proceedings in Informatics

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

A *trigraph* $G = (V, E_b, E_r)$ consists of a vertex set V , a set of black edges E_b , and a set of red edges E_r . All trigraphs considered in this paper are undirected, finite, and without loops or multiple edges (red or black). Let $N_b^G(v) = \{x \mid \{v, x\} \in E_b\}$ be the open black neighbourhood of v in G , and $N_r^G(v) = \{x \mid \{v, x\} \in E_r\}$ be its open red neighbourhood. The *red degree* of $v \in V$ is $\delta_r(v) = |\{e \in E_r \mid v \in e\}|$. The *red degree* of G is $\max_{v \in V} \delta_r(v)$.

A *contraction* in a trigraph G consists of merging any two vertices u and v into a single vertex w , and updating the edges of G as follows. Every vertex $x \in N_r^G(u) \cup N_r^G(v) \cup (N_b^G(u) \Delta N_b^G(v))$ is linked to w by a red edge, and every vertex $x \in N_b(u) \cap N_b(v)$ is linked to w by a black edge. All other edges in the graph (those not incident to u or v) remain unchanged. A *contraction sequence* $(G_1, \dots, G_{|V|})$ for G is a sequence of contractions starting from $G_1 = G$ with no red edge and ending in $G_{|V|}$ consisting of a single vertex, where G_{i+1} is obtained from G_i by performing a contraction. A *k-contraction sequence* is such a sequence in which all graphs G_i have red degree at most k . Then, the *twin-width* of the graph is the minimal k for which there exists a k -contraction sequence.

In the following sections we describe our algorithm finding such an optimal contraction sequence. Section 2 describes a simple reduction rule. Section 3 describes the branch and bound algorithm that is used in several parts of our final algorithm, which is described in Section 5.

2 Reduction Rule

The following reduction rule is a generalisation of twins for trigraphs and is applied in all parts of our algorithms.

► **Reduction Rule 1.** *If there exist two vertices $u, v \in V$ such that*

- $N_b^G(u) \subseteq N_b^G(v)$, and
- $(N_r^G(v) \cup N_b^G(v)) \subseteq (N_r^G(u) \cup \{u, v\})$,

then contract u and v .

The correctness of this reduction rule is immediate: Let G' be the graph after contracting u and v . Then $G' = G[V \setminus \{v\}]$ and the twin-width of induced trigraphs can only decrease [3]. Note that u and v are not symmetric in the above description of the reduction rule. But the rule as it is presented above is easier to implement.

3 Main Branch and Bound Algorithm

The main building block of our algorithm is a branch and bound algorithm that tries all pairs of possible contractions in each search tree node. In the following, we present a few optimizations made in our solver.

Search space reduction. Consider a search tree node x of our branch and bound algorithm. Assume we contract vertices u and v in G , and recurse into child node y of the search tree. We do not have to try contracting u and v in the search trees rooted at sibling nodes of y until some contraction “touches” $N_G^2(u)$ or $N_G^2(v)$ where N_G^2 is the closed radius-two neighbourhood in G (considering black and red edges) (see Algorithm 1). We store these so-called forbidden contractions (set S in Algorithm 1) using a global vector of sets.

■ **Algorithm 1** A sketch of the recursive branch and bound algorithm that avoids symmetries. The input is a trigraph G and a set S of *forbidden contractions*. Both are updated during the search tree.

```

1 SearchTree( $G, S$ ):
2   for  $(u, v) \in V(G) \times V(G)$  do
3     if  $u < v \wedge (u, v) \notin S$  then
4        $S' \leftarrow S \setminus ((N_G^2(u) \cup N_G^2(v)) \times V(G)) \setminus (V(G) \times (N_G^2(u) \cup N_G^2(v)))$ ;
5       SearchTree(contract( $G, u, v$ ),  $S'$ );
6        $S \leftarrow S \cup \{(u, v), (v, u)\}$ ;

```

Branching order. To guide the order in which we explore the search tree induced by our branch and bound algorithm, we consider all possible contractions in a search tree node in a given order. Namely, for $u, v \in V$, let $\Delta(u, v) = |(N_b(u) \Delta N_b(v)) \cup (N_r(u) \cup N_r(v)) \setminus \{u, v\}|$. Then we contract u_1 and v_1 before u_2 and v_2 if $\Delta(u_1, v_1) < \Delta(u_2, v_2)$. We sort all pairs using bucket sort, using one bucket for each $k = \Delta(u_1, v_1)$. This is faster than sorting all pairs using standard sorting algorithms when k is small. We observed that our initial upper bounds are small for most instances, so this improved the overall runtime of our algorithm.

Optimisations. We list further optimisations that we apply during the branch and bound process.

- If the current trigraph has at most 64 vertices then a vector of unsigned long long is used to store the current adjacency lists (for red and black edges). If bit j for index i in this data structure is one then this indicates the presence of an edge. This allows for faster set operations on neighbourhoods of vertices.
- At each search tree node, contract twins based on Reduction Rule 1. To find these twins, we do not have to try all pairs of vertices, but only those that are in the neighbourhood of contractions.
- We use a global upper bound and local/global lower bounds to leave infeasible parts of the search tree as early as possible.

4 Upper Bounds

We apply two heuristics that compute contraction sequences yielding us upper bounds. The first, called Heuristic1, iteratively contracts two vertices u, v such that $\Delta(u, v)$ is minimal among all pairs of vertices of the current graph. The second heuristic, called Heuristic2, uses randomness and can be executed several times. Given the current graph and an upper bound T , it contracts a random pair of vertices among all pairs u, v such that, after the contraction of u and v , the red degree of the trigraph does not exceed $T - 1$. If Heuristic2 is able to find a contraction sequence with twin-width less than T then this twin-width can be used as new upper bound for the next iteration.

5 The Complete Algorithm

The complete algorithm consists of three phases: computing a lower bound, computing an upper bound, and then using the branch and bound algorithm. It is run on each connected component of the graph, and each phase is given a time limit as indicated below.

Lower bound (15 minutes). A lower bound is computed based on the twin-width of induced subgraphs of G . We start with $|V|$ induced subgraphs, each containing one distinct vertex of the graph. Then, for each graph G_i , we iteratively add a vertex v ; v has to be connected to G_i and v 's minimum symmetric difference of neighbourhoods (in G) with vertices in G_i is maximal. After adding a vertex, the twin-width of the newly obtained induced subgraph is computed, and the lower bound is updated.

Upper bound heuristics (5 minutes). Next, Heuristic1 is ran once, and Heuristic2 is called multiple times in the remaining time. In a lot of instances, Heuristic2 is able to match the lower bound. So we are done and do not have to go to the next stage of the algorithm.

Exact Branch and Bound (10 minutes). The remaining time the exact branch and bound algorithm is applied to the input trigraph G . The upper bound obtained from the previous algorithm is used to leave infeasible parts of the search tree early. If a contraction sequence with twin-width matching the lower bound from the first phase is found, we report that contraction sequence as the solution. Otherwise, if the twin-width is larger than that lower bound, the complete search space (without infeasible parts due to the upper bound and without symmetries) is explored to find the contraction sequence with the smallest twin-width.

References

- 1 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding Twin-Width at Most 4 Is NP-Complete. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *Proc. International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2022.18.
- 2 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. *Algorithmica*, 84(11):3300–3337, 2022.
- 3 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.
- 4 André Schidler and Stefan Szeider. A SAT approach to twin-width. In Cynthia A. Phillips and Bettina Speckmann, editors, *Proc. Symposium on Algorithm Engineering and Experiments (ALENEX 2022)*, pages 67–77. SIAM, 2022. doi:10.1137/1.9781611977042.6.

PACE Solver Description: Zygoty

Emmanuel Arrighi ✉ 🏠 

University of Trier, Germany

Pål Grønås Drange ✉ 

University of Bergen, Norway

Kenneth Langedal ✉ 

University of Bergen, Norway

Farhad Vadiie ✉ 

University of Bergen, Norway

Martin Vatshelle ✉

University of Bergen, Norway

Petra Wolf ✉ 🏠 

University of Bergen, Norway

Abstract

The graph parameter twin-width was recently introduced by Bonnet et al. Twin-width is a parameter that measures a graph's similarity to a cograph, which is a graph that can be reduced to a single vertex by repeatedly contracting twins. This brief description introduces Zygoty, a heuristic for computing a low-width contraction sequence that achieved second place in the 2023 edition of Parameterized Algorithms and Computational Experiments Challenge (PACE). Zygoty starts by repeatedly contracting twins. Then, any attached trees are contracted down to a single pendant vertex. The remaining graph is then contracted using a randomized greedy algorithm.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Twin-width, randomized greedy algorithm

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.39

Supplementary Material *Software (Source Code)*: <https://doi.org/10.5281/zenodo.8370343>

1 Problem description

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E , where the edge set is partitioned into black edges $B(E)$ and red edges $R(E)$. The neighborhood of a vertex u is denoted by $N(u)$, formally $N(u) = \{v \in V \mid uv \in E\}$. A neighbor v of a vertex u can be black or red, depending on whether uv is in $B(E)$ or $R(E)$. Formally, for a vertex $u \in V$, we let $N^r(u) = \{v \in V \mid uv \in R(E)\}$ be the red neighborhood of u and $N^b(u) = \{v \in V \mid uv \in B(E)\}$ be the black neighborhood of u . The degree of a vertex u is the size of the neighborhood $|N(u)|$ and the black and red degree refers to the size of the black and red neighborhood ($|N^b(u)|$ and $|N^r(u)|$), respectively.

A contraction between two vertices u and v is performed by removing u and v from G , and adding a new vertex w as follow:

- $N^r(w) = N^r(u) \cup N^r(v) \cup (N^b(u) \Delta N^b(v)) \setminus \{u, v\}$ where $N^b(u) \Delta N^b(v)$ is the symmetric difference.
- $N^b(w) = N^b(u) \cap N^b(v)$.

A contraction sequence starts from the original graph and ends with a single vertex. This original graph starts with no red edges. The *width* of a contraction sequence is the highest red degree of any vertex along the way. And finally, the twin-width parameter introduced by Bonnet et al. [1] is the lowest width among all possible contraction sequences for G .



© Emmanuel Arrighi, Pål Grønås Drange, Kenneth Langedal, Farhad Vadiie, Martin Vatshelle, and Petra Wolf;

licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 39; pp. 39:1–39:3

Leibniz International Proceedings in Informatics



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

2 Solver description

Zygoty is a randomized greedy heuristic for computing low-width contraction sequences. Before the randomized part, it performs two preprocessing steps. First, all twins are contracted. Secondly, as described by Bonnet et al. [1], trees are contracted by either contracting two leaves with the same parent or a leaf with its parent. If the whole graph is a tree, this gives a sequence with a width of at most two. Otherwise, the tree will contract down to a single pendant vertex connected by a red edge.

The remaining sequence is constructed by considering multiple random contractions at each step and selecting the best one. The exact number of contractions to consider is updated continuously during the construction. The number is updated every second based on the remaining time and the average time it takes to check one contraction. Zygoty employs two measures to evaluate the quality of a contraction.

- **Red degree:** This red degree refers to the highest red degree of a vertex that increased its red degree due to this contraction. The newly merged vertex is considered increased if its red degree is greater than both of the merged vertices. Intuitively, the red degree is the most important thing to keep low when constructing a low-width contraction sequence.
- **Intersection size:** This is the size of the intersection between the neighborhoods of the two merged vertices. This number also says how many edges will be removed due to this contraction. If an edge connects the two merged vertices, this counts one to the intersection size. The intuition for this measure is that fewer edges make for fewer problems later.

Zygoty prioritizes the smallest red degree and uses intersection size as a tiebreaker. The only exception to this rule is when both contractions being compared have a red degree lower than the current width of the sequence generated so far. In such cases, only the intersection size is taken into account.

Instead of randomly picking a pair of vertices, Zygoty randomly picks the first vertex and then does a random walk from there. The random walk performs either one or two steps, depending on a biased coin toss. The odds on this coin are decided based on the density of the input graph, favoring distance one walks on sparser graphs and distance two on dense ones.

3 Implementation details

For this randomized procedure, it is crucial to quickly compute the intersection and red degree of a potential contraction. Therefore, spending more time elsewhere can be beneficial if it speeds up the contraction checking. For this reason, we don't use any associative data structure but instead store the neighborhood of a vertex continuously in a single array. The black and red edges are kept separate and sorted separately. With this setup, computing the intersection and red degree can be done by scanning through the two neighborhoods in a cache-friendly manner. It is possible to make a single pass using four pointers. However, we noticed that doing a separate pass over the red neighborhoods first was faster. This pass copies the difference to a separate array. Then, a second pass over the black parts and the temporarily stored difference from the red ones. This captures every relation a vertex can have with the contraction in question, and the intersection and max degree are registered accordingly.

References

- 1 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *ACM Journal of the ACM (JACM)*, 69(1):1–46, 2021.

PACE Solver Description: RedAlert - Heuristic Track

Édouard Bonnet   

Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Julien Duron  

Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Abstract

We present **RedAlert**, a heuristic solver for twin-width, submitted to the Heuristic Track of the 2023 edition of the Parameterized Algorithms and Computational Experiments (PACE) challenge.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases twin-width, contraction sequences, heuristic, pair sampling, pair filtering

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.40

Related Version *Full Version*: <https://bitbucket.org/tennobe/red-alert>

Funding This work was supported by the ANR projects TWIN-WIDTH (ANR-21-CE48-0014) and Digraphs (ANR-19-CE48-0013).

1 Twin-width and contraction sequences

To keep our description short, we refer the reader to the first two sections of [2] for the definitions and motivations behind contraction sequences and twin-width. A *trigraph* has two disjoint edge relations: red edges and black edges. Its *total graph* consists of the union of these two relations. The *red degree* (resp. *total degree*) is the degree in the graph formed by the red edges (resp. in the *total graph*). We aim to find a contraction sequence (that iteratively identifies two vertices and updates the color of their incident edge, until there is only one vertex left) with overall maximum red degree as low as possible.

2 Overview of RedAlert

In the search of contraction sequences of low width, a natural subroutine consists of finding a *good pair of vertices*, that is, one whose contraction results in a trigraph with maximum red degree as low as possible. An oracle providing the greedily best pair in 10^{-5} s would have likely won the competition. However, this is far from what is physically possible. In theory, getting a best pair within the allowed 300s is already challenging, since the largest instances had order of 10^7 vertices, and CLOSEST VECTOR PAIR –more or less the *best pair* problem when starting from a graph– (like the task of finding an orthogonal pair of 0,1-vectors) has no truly subquadratic algorithm unless the Strong Exponential-Time Hypothesis fails [3].

We thus resolve to sampling the pairs of vertices as candidates for the next contraction. Based on the remaining time and number of vertices, we compute an integer `minPairs` indicating the minimum number of pairs to be contracted from the sampled pairs (to finish in time). We then contract at least `minPairs` candidate pairs according to a cost function detailed below. Challenging instances produce denser and denser trigraphs as we contract them. This results in a progressive increase of the time to sample and select pairs to contract. In such cases, we may resort to faster (and rougher) subroutines to finish the contraction sequences: `totDegAlert` and `balancedScheme`. We will briefly detail them.



© Édouard Bonnet and Julien Duron;

licensed under Creative Commons License CC-BY 4.0

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).

Editors: Neeldhara Misra and Magnus Wahlström; Article No. 40; pp. 40:1–40:5

Leibniz International Proceedings in Informatics



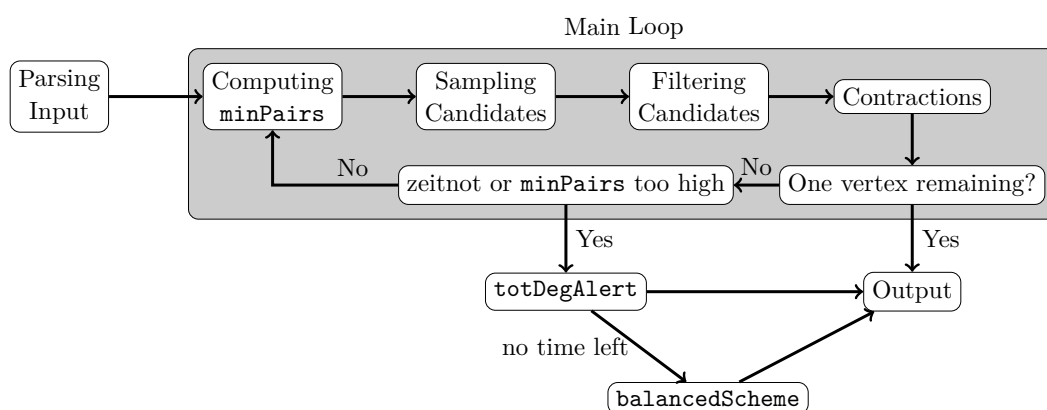
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

40:2 PACE Solver Description: RedAlert - Heuristic Track

To summarize, the main loop of RedAlert is as follows:

1. Estimation of `minPairs`;
2. Sampling of the *candidate pairs*;
3. Selection of at least `minPairs` most favorable pairs among the candidate pairs;
4. Contraction of these pairs;
5. If running out of time, finish with increasingly cruder heuristics.

Also see Figure 1.



■ **Figure 1** Sketch of RedAlert. We exit the main loop when (we have a solution or) the time budget of 260s is depleted or we would have to contract over 40% of our sampled pairs. We then call `totDegAlert` up to second 285. If by then no complete contraction sequence was found, we call `balancedScheme` which takes less than a second to terminate.

3 Dense and Small vs. Sparse and Large Inputs

Before we start parsing the input graph, we decide based on its number of vertices and edges whether we want to work with adjacency matrices or adjacency lists. Typically the former shall be preferred on graphs with few vertices but high edge density, while it is simply not an option when the number of vertices becomes too large. Only the first three or four instances (below 2500 vertices) of the Heuristic Track were such that our treatment with adjacency matrices performed better. Thus we will mainly describe the part of our algorithm using adjacency lists. Nevertheless, let us mention one nice feature of using adjacency matrices: one can test the *quality* of a pair of vertices by sampling an appropriate number of indices, and computing the number of disagreements (potential red edges) at these indices.

In the *sparse* case, we represent our trigraphs with slightly-modified adjacency lists: the neighbors of a vertex are stored in a set. Each vertex has a set for its black neighbors, and a set for its red neighbors. While contracting the trigraph, we maintain other useful elements such as the remaining number of vertices, edges, maximum red degree, overall maximum red degree, list of pairs *vertex/red degree*, list of pairs *vertex/total degree*, and some arrays to keep the conversion between *local* labels and *global* labels. The contraction operation remains reasonably fast and takes a typical 10^{-4} s on the large instances.

4 Sampling candidates

4.1 Computing minPairs

At each iteration of the main loop, we sample around 10000 pairs of vertices, and contract `minPairs` pairs of them. To choose `minPairs`, we compute the time used in the last iteration of the loop, say t and the remaining time say T . In the hypothesis where the next iterations will use the same time t , we can afford T/t more iterations. In practice, not all the iterations take the same amount of time, but the changes are sufficiently gradual for the approximation to work.

If the current number of vertices of the graph is k , then each of those iterations should contract $\frac{k}{T/t} = tk/T$ pairs, which leads to `minPairs` = tk/T . Typical values of `minPairs` are 1 for the smallest instances, 30 for medium ones, and above 1000 for the largest, a problem that we tackle in Section 6. When `minPairs` is too small (1 for example) we increase the sampling size to better use our time. To achieve this, we increase it while `minPairs` is below 30, decrease it if later `minPairs` gets above 70, and reset it to its minimum of 10000 when `minPairs` reaches 500.

4.2 Candidates distribution

In the *sparse* case, our distribution is biased towards pairs of vertices that are close to each other. We pick a vertex v uniformly at random. Then with probability $1/2$, we uniformly pick a first neighbor of v (in the total graph) to complete the pair, and with probability $1/2$, we uniformly choose a second neighbor of v (still in the total graph). This performs well on the sparsest instances. As half of our sampled pairs consist of vertices at distance 2, we naturally find contractions that are decreasing the red degree of high-degree vertices.

We experimented a bit with favoring vertices v with low red degree or low total degree, or adding pairs of red neighbors of a vertex with highest red degree. This did not seem to improve the overall performance of the heuristic, so we opted for this simple distribution.

In the *dense* case, this distribution is not helpful: most pairs of vertices are at distance 2. We thus used the uniform distribution.

5 Filtering candidates

We evaluate the sampled pairs based on a cost function f , defined as follows. If G is the current trigraph, u, v two vertices of G , and G' the resulting trigraph if u and v were contracted into w , we set $f(u, v) = (r, p, e)$ where

- $r \in \{0, 1\}$, and $r = 0$ iff the maximum red degree of G' is smaller than that of G ;
- p is the maximum red degree among vertices of G' in the closed red neighborhood of w ;
- e is the total number of red edges in G' .

When comparing two pairs of vertices, we prefer the one whose image by f is lexicographically smaller. When different sampled pairs share vertices, we cannot contract both of them. In the same way, the contraction of a pair can drastically change the evaluation of another pair. To overcome those issues, we build a min-heap (according to f) of the candidates, and contract them in the following way:

- Pop the minimal candidate c
- If one of the vertices of c does not exist anymore, we continue
- We evaluate $f(c)$ in the current trigraph
- If $f(c)$ is not worst than the previous time it was evaluated, we contract c
- Otherwise we add c to the min-heap with the new value of $f(c)$.

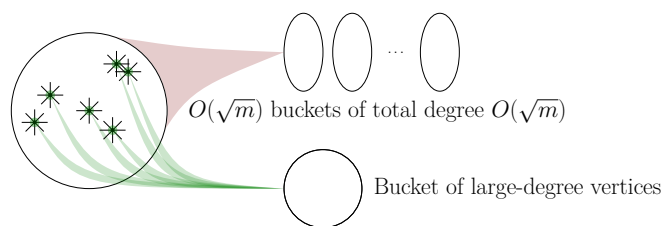
And we loop until we contracted `minPairs` candidates. This procedure is particularly useful in the case of a path: If we did not evaluate again the candidates after one contraction, then we would contract both endpoints of the path with their neighbor, resulting in a sequence of width 2 (instead of 1). We found out that further contracting all pairs tying with the worst contracted pair may advantageously clear some time up, heading to an uncertain and more complicated future.

A major issue with f is that computing $f(u, v)$ is linear in $d(u) + d(v)$ where $d(u)$ and $d(v)$ are the degrees of u and v . This implies that increasing the density of the trigraph increases the time taken to evaluate f . In this case, we cannot afford to be as picky as before in the choice of candidate we contract, which is implemented by the augmentation of `minPairs`.

6 When time gets shorter or `minPairs` gets too large

It can happen that `minPairs`, which is computed based on the number of remaining time and vertices (or contractions), and the time spent in the previous loop iteration per performed contraction, steadily increases. This typically happens when the *densification of the current trigraph accelerates*. This may result in a situation when, to meet its deadline, the heuristic has to contract a large fraction of the sampled pairs, making it close to the random heuristic.

In this case, remark that the initial average degree was quite low, and as the black degree cannot increase, the densification of the trigraph come from red edges. We can deduce from this observation that the degree of a vertex is a good approximation of its red degree. We thus break out of the main loop and call a faster subroutine, `totDegAlert`, which greedily contracts pairs of smallest total degree. This subroutine still requires to explicitly perform contractions, which takes some time on the largest instances. It is thus possible that we run out of time even inside `totDegAlert`. Therefore, when we have only 15 seconds left, we call `balancedScheme`. This subroutine is based on the $O(\sqrt{m})$ -sequence for m -edge graphs (see [1] for a more precise bound). It partitions the vertex set into $O(\sqrt{m})$ buckets of roughly equal sum of total degrees, plus an additional bucket with vertices of total degree $\Omega(\sqrt{m})$ (large degree), see Figure 2.



■ **Figure 2** Partition of the trigraph in buckets yielding an $O(\sqrt{m})$ -sequence.

The idea is then to contract every bucket into a single vertex, finishing with the bucket of large-degree vertices, and end the contraction sequence arbitrarily. What makes `balancedScheme` particularly fast is that we do not need to make these contractions explicitly. As a simple but effective optimization, we contract each bucket in such a way that the *contraction tree* is a balanced binary tree rather than a caterpillar. When `RedAlert` is about to output a solution for which it knows the actual width (i.e., without invoking `balancedScheme`), we first compare it to some small multiple of \sqrt{m} where m is the number of edges of the input graph. In some cases, indeed, running `balancedScheme` from scratch on the original graph gives a better contraction sequence.

References

- 1 Jungho Ahn, Kevin Hendrey, Donggyu Kim, and Sang-il Oum. Bounds for the twin-width of graphs. *SIAM J. Discret. Math.*, 36(3):2352–2366, 2022. doi:10.1137/21m1452834.
- 2 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.
- 3 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.

