

17th International Symposium on Parameterized and Exact Computation

IPEC 2022, September 7–9, 2022, Potsdam, Germany

Edited by

Holger Dell

Jesper Nederlof



Editors

Holger Dell 

Goethe University Frankfurt, Germany
dell@uni-frankfurt.de

Jesper Nederlof 

Utrecht University, The Netherlands
j.nederlof@uu.nl

ACM Classification 2012

Theory of computation → Parameterized complexity and exact algorithms

ISBN 978-3-95977-260-0

Published online and open access by

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <https://www.dagstuhl.de/dagpub/978-3-95977-260-0>.

Publication date

December, 2022

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.2022.0

ISBN 978-3-95977-260-0

ISSN 1868-8969

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

LIPICs – Leibniz International Proceedings in Informatics

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

Editorial Board

- Luca Aceto (*Chair*, Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Mikolaj Bojanczyk (University of Warsaw, PL)
- 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)

ISSN 1868-8969

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

■ Contents

Preface	
<i>Holger Dell and Jesper Nederlof</i>	0:ix
Program Committees	
.....	0:xi
List of External Reviewers	
.....	0:xiii
List of Authors	
.....	0:xv

Regular Papers

A Finite Algorithm for the Realizability of a Delaunay Triangulation	
<i>Akanksha Agrawal, Saket Saurabh, and Meirav Zehavi</i>	1:1–1:16
Parameterized Complexity of Perfectly Matched Sets	
<i>Akanksha Agrawal, Sutanay Bhattacharjee, Satyabrata Jana,</i> <i>and Abhishek Sahu</i>	2:1–2:13
On the Hardness of Generalized Domination Problems Parameterized by Mim-Width	
<i>Brage I. K. Bakkane and Lars Jaffke</i>	3:1–3:19
FPT Approximation for Fair Minimum-Load Clustering	
<i>Sayan Bandyopadhyay, Fedor V. Fomin, Petr A. Golovach, Nidhi Purohit,</i> <i>and Kirill Simonov</i>	4:1–4:14
On Sparse Hitting Sets: From Fair Vertex Cover to Highway Dimension	
<i>Johannes Blum, Yann Disser, Andreas Emil Feldmann, Siddharth Gupta,</i> <i>and Anna Zych-Pawlewicz</i>	5:1–5:23
On the Complexity of Problems on Tree-Structured Graphs	
<i>Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Marcin Pilipczuk,</i> <i>and Michał Pilipczuk</i>	6:1–6:17
On the Parameterized Complexity of Computing Tree-Partitions	
<i>Hans L. Bodlaender, Carla Groenland, and Hugo Jacob</i>	7:1–7:20
XNLP-Completeness for Parameterized Problems on Graphs with a Linear Structure	
<i>Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Lars Jaffke,</i> <i>and Paloma T. Lima</i>	8:1–8:18
Twin-Width VIII: Delineation and Win-Wins	
<i>Édouard Bonnet, Dibyayan Chakraborty, Eun Jung Kim, Noleen Köhler,</i> <i>Raul Lopes, and Stéphan Thomassé</i>	9:1–9:18
Obstructions to Faster Diameter Computation: Asteroidal Sets	
<i>Guillaume Ducoffe</i>	10:1–10:24

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof



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

On the Parameterized Complexity of Symmetric Directed Multicut <i>Eduard Eiben, Clément Rambaud, and Magnus Wahlström</i>	11:1–11:17
Computing Generalized Convolutions Faster Than Brute Force <i>Barış Can Esmer, Ariel Kulik, Dániel Marx, Philipp Schepper, and Karol Węgrzycki</i>	12:1–12:22
Exact Exponential Algorithms for Clustering Problems <i>Fedor V. Fomin, Petr A. Golovach, Tanmay Inamdar, Nidhi Purohit, and Saket Saurabh</i>	13:1–13:14
Domination and Cut Problems on Chordal Graphs with Bounded Leafage <i>Esther Galby, Dániel Marx, Philipp Schepper, Roohani Sharma, and Prafullkumar Tale</i>	14:1–14:24
Slim Tree-Cut Width <i>Robert Ganian and Viktoriia Korchemna</i>	15:1–15:18
A Fixed-Parameter Algorithm for the Schrijver Problem <i>Ishay Haviv</i>	16:1–16:16
Towards Exact Structural Thresholds for Parameterized Complexity <i>Falko Hegerfeld and Stefan Kratsch</i>	17:1–17:20
Hardness of Interval Scheduling on Unrelated Machines <i>Danny Hermelin, Yuval Itzhaki, Hendrik Molter, and Dvir Shabtay</i>	18:1–18:16
Vertex Cover and Feedback Vertex Set Above and Below Structural Guarantees <i>Leon Kellerhals, Tomohiro Koana, and Pascal Kunz</i>	19:1–19:14
Parameterized Local Search for Vertex Cover: When Only the Search Radius Is Crucial <i>Christian Komusiewicz and Nils Morawietz</i>	20:1–20:18
Parameterized Complexity of a Parallel Machine Scheduling Problem <i>Maher Mallem, Claire Hanen, and Alix Munier-Kordon</i>	21:1–21:21
Anti-Factor Is FPT Parameterized by Treewidth and List Size (But Counting Is Hard) <i>Dániel Marx, Govind S. Sankar, and Philipp Schepper</i>	22:1–22:23
Parameterized Complexity of Maximum Happy Set and Densest k -Subgraph <i>Yosuke Mizutani and Blair D. Sullivan</i>	23:1–23:18
Parameterized Complexity of Streaming Diameter and Connectivity Problems <i>Jelle J. Oostveen and Erik Jan van Leeuwen</i>	24:1–24:16
Applying a Cut-Based Data Reduction Rule for Weighted Cluster Editing in Polynomial Time <i>Hjalmar Schulz, André Nichterlein, Rolf Niedermeier, and Christopher Weyand</i>	25:1–25:14

PACE Solver Descriptions

The PACE 2022 Parameterized Algorithms and Computational Experiments Challenge: Directed Feedback Vertex Set <i>Ernestine Großmann, Tobias Heuer, Christian Schulz, and Darren Strash</i>	26:1–26:18
PACE Solver Description: DiVerSeS – A Heuristic Solver for the Directed Feedback Vertex Set Problem <i>Sylwester Swat</i>	27:1–27:3
PACE Solver Description: Mount Doom – An Exact Solver for Directed Feedback Vertex Set <i>Sebastian Angrick, Ben Bals, Katrin Casel, Sarel Cohen, Tobias Friedrich, Niko Hastrich, Theresa Hradilak, Davis Issac, Otto Kießig, Jonas Schmidt, and Leo Wendt</i>	28:1–28:4
PACE Solver Description: Hust-Solver – A Heuristic Algorithm of Directed Feedback Vertex Set Problem <i>YuMing Du, QingYun Zhang, JunZhou Xu, ShunGen Zhang, Chao Liao, ZhiHuai Chen, ZhiBo Sun, ZhouXing Su, JunWen Ding, Chen Wu, PinYan Lu, and ZhiPeng Lv</i>	29:1–29:3
PACE Solver Description: GraPA-JAVA <i>Moritz Bergenthal, Jona Dirks, Thorben Freese, Jakob Gahde, Enna Gerhard, Mario Grobler, and Sebastian Siebertz</i>	30:1–30:4
PACE Solver Description: DreyFVS <i>Gabriel Bathie, Gaétan Berthe, Yoann Coudert–Osmond, David Desobry, Amadeus Reinald, and Mathis Rocton</i>	31:1–31:4
PACE Solver Description: DAGer – Cutting out Cycles with MaxSAT <i>Rafael Kiesel and André Schidler</i>	32:1–32:4

■ 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. Started in 2004 as a biennial workshop, it 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 Hong Kong, China
- 2021 virtual / Lisbon, Portugal

This volume contains the papers presented at IPEC 2022: the 17th International Symposium on Parameterized and Exact Computation. IPEC 2022 was held on September 7–9. It was a part of the ALGO 2022 congress, and took place in Potsdam, Germany. In response to the call for papers, 47 extended abstracts were submitted and 25 of them were ultimately selected for presentation at the conference and inclusion in these proceedings. Each considered submission received at least 3 reviews. The reviews were performed in a double-blind fashion by the 16 regular members of the program committee and by 19 external reviewers, together contributing 141 full reviews.

The **Best Paper Award** was given to Hans L. Bodlaender (Utrecht University), Carla Groenland (Utrecht University), Hugo Jacob (ENS Paris-Saclay), Lars Jaffke (University of Bergen) and Paloma de Lima (IT University of Copenhagen) for their paper “*XNLP-completeness for Parameterized Problems on Graphs with a Linear Structure*”.

The **Best Student Paper Award** was given to Jelle Oostveen (Utrecht University) and Erik Jan van Leeuwen (Utrecht University) for their paper “*Parameterized Complexity of Streaming Diameter and Connectivity Problems*”.

The **EATCS-IPEC Nerode Prize** was given to Bruno Courcelle for his papers “*The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs*” and “*The Monadic Second-Order Logic of Graphs III: Tree-Decompositions, Minors and Complexity Issues*”. IPEC 2022 hosted an award ceremony with a plenary talk given by Bruno Courcelle. The Nerode Prize committee consisted of Anuj Dawar (University of Cambridge), Fedor Fomin (University of Bergen), and Thore Husfeldt (IT University of Copenhagen).

Eun Jung Kim (Université Paris-Dauphine, PSL Research University, CNRS) presented an **invited tutorial** on “*Directed Flow-augmentation*”. Finally, IPEC 2022 hosted the award ceremony of the seventh Parameterized Algorithms and Computational Experiments (PACE) challenge. These proceedings contain a report on the **PACE 2022** challenge and brief communications of the winners about their solvers.

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 2022 for the local arrangements.

Holger Dell and Jesper Nederlof
Frankfurt and Utrecht, October 2022



■ Program Committees

IPEC 2022 Program Committee

- Christian Komusiewicz (Marburg University)
- Christophe Paul (Laboratoire d'Informatique Robotique et Microélectronique de Montpellier)
- Cornelius Brand (Technische Universität Wien)
- Édouard Bonnet (ENS Lyon)
- Holger Dell (Goethe University Frankfurt, ITU Copenhagen, and BARC, co-chair)
- Jesper Nederlof (Utrecht University, co-chair)
- Karol Węgrzycki (Saarland University and Max Planck Institute for Informatics)
- M.S. Ramanujan (University of Warwick)
- Marvin Künnemann (TU Kaiserslautern)
- Michael Lampis (Paris Dauphine University)
- Neeldhara Misra (IIT Gandhinagar)
- Paweł Rzażewski (Warsaw University of Technology and University of Warsaw)
- Radu Curticapean (ITU Copenhagen and BARC)
- René van Bevern (Huawei Cloud Technologies Co., Ltd.)
- Robert Ganian (Technische Universität Wien)
- Roohani Sharma (Max Planck Institute for Informatics)
- Sándor Kisfaludi-Bak (Aalto University)
- Valia Mitsou (Research Institute on the Foundations of Computer Science (IRIF) and Paris Diderot University)

IPEC 2022 Steering Committee

- Dániel Marx (2020 – 2023)
- Eun Jung Kim (2019 – 2022)
- Fedor Fomin (2021 – 2024)
- Holger Dell (2021 – 2024)
- Jesper Nederlof (2021 – 2024)
- Marcin Pilipczuk (2019 – 2022, chair)
- Meirav Zehavi (2020 – 2023)
- Petr Golovach (2020 – 2023)
- Yixin Cao (2019 – 2022)

PACE 2022 Program Committee

- Christian Schulz (chair) (Universität Heidelberg)
- Ernestine Großmann (Universität Heidelberg)
- Tobias Heuer (Karlsruher Institut für Technologie)
- Darren Strash (Hamilton College)



■ List of External Reviewers

- Ararat Harutyunyan
- Archontia Giannopoulou
- Evangelos Protopapas
- Frank Sommer
- Giannos Stamoulis
- Kirill Simonov
- Laure Morelle
- Liana Khazaliya
- Mamadou Moustapha Kanté
- Manuel Lafond
- Mathias Weller
- Mathis Rocton
- Peter Rossmanith
- Philipp Schepper
- Rémy Belmonte
- Sebastian Ordyniak
- Stefan Mengel
- Thekla Hamm
- Viktoriia Korchemna



■ List of Authors

Akanksha Agrawal  (1, 2)
Indian Institute of Technology Madras,
Chennai, India

Sebastian Angrick (28)
Hasso Plattner Institut,
Universität Potsdam, Germany

Brage I. K. Bakkane (3)
University of Bergen, Norway

Ben Bals (28)
Hasso Plattner Institut,
Universität Potsdam, Germany

Sayan Bandyopadhyay (4)
Department of Informatics,
University of Bergen, Norway

Gabriel Bathie (31)
École Normale Supérieure de Lyon, France

Moritz Bergenthal  (30)
Universität Bremen, Germany

Gaëtan Berthe (31)
École Normale Supérieure de Lyon, France

Sutanay Bhattacharjee (2)
Indian Institute of Technology Madras,
Chennai, India

Johannes Blum  (5)
Universität Konstanz, Germany

Hans L. Bodlaender  (6, 7, 8)
Department of Information and Computing
Sciences, Utrecht University, The Netherlands

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

Katrin Casel  (28)
Hasso Plattner Institut,
Universität Potsdam, Germany

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

ZhiHuai Chen (29)
Huawei TCS Lab Shanghai, China

Sarel Cohen  (28)
The Academic College of Tel Aviv-Yaffo, Israel

Yoann Coudert-Osmont (31)
Université de Lorraine, CNRS, Inria, LORIA,
Nancy, France

David Desobry (31)
Université de Lorraine, CNRS, Inria, LORIA,
Nancy, France

JunWen Ding (29)
School of Computer Science and Technology,
Huazhong University of Science & Technology,
China

Jona Dirks (30)
Universität Bremen, Germany

Yann Disser  (5)
Technische Universität Darmstadt, Germany

YuMing Du (29)
School of Computer Science and Technology,
Huazhong University of Science & Technology,
China

Guillaume Ducoffe  (10)
National Institute of Research and Development
in Informatics, Bucharest, Romania;
University of Bucharest, Romania

Eduard Eiben  (11)
Department of Computer Science, Royal
Holloway, University of London, Egham, UK

Barış Can Esmer  (12)
CISPA Helmholtz Center for Information
Security, Saarbrücken, Germany

Andreas Emil Feldmann  (5)
Charles University, Prague, Czechia

Fedor V. Fomin  (4, 13)
Department of Informatics,
University of Bergen, Norway

Thorben Freese (30)
Universität Bremen, Germany

Tobias Friedrich  (28)
Hasso Plattner Institut,
Universität Potsdam, Germany

Jakob Gahde (30)
Universität Bremen, Germany

Esther Galby (14)
TU Hamburg, Germany

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).
Editors: Holger Dell and Jesper Nederlof



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

- Robert Ganian  (15)
Algorithms and Complexity Group,
TU Wien, Austria
- Enna Gerhard  (30)
Universität Bremen, Germany
- Petr A. Golovach  (4, 13)
Department of Informatics,
University of Bergen, Norway
- Mario Grobler  (30)
Universität Bremen, Germany
- Carla Groenland  (6, 7, 8)
Department of Information and Computing
Sciences,
Utrecht University, The Netherlands
- Ernestine Großmann  (26)
Universität Heidelberg, Germany
- Siddharth Gupta  (5)
University of Warwick, Coventry, UK
- Claire Hanen  (21)
Sorbonne Université, CNRS, LIP6,
F-75005 Paris, France;
Université Paris Nanterre, UPL,
92000 Nanterre, France
- Niko Hastrich (28)
Hasso Plattner Institut,
Universität Potsdam, Germany
- Ishay Haviv (16)
School of Computer Science, The Academic
College of Tel Aviv-Yaffo, Israel
- Falko Hegerfeld  (17)
Humboldt-Universität zu Berlin, Germany
- Danny Hermelin (18)
Department of Industrial Engineering and
Management, Ben-Gurion University of the
Negev, Beer-Sheva, Israel
- Tobias Heuer  (26)
Karlsruhe Institute of Technology, Germany
- Theresa Hradilak (28)
Hasso Plattner Institut,
Universität Potsdam, Germany
- Tanmay Inamdar  (13)
Department of Informatics,
University of Bergen, Norway
- Davis Issac  (28)
Hasso Plattner Institut,
Universität Potsdam, Germany
- Yuval Itzhaki (18)
Department of Industrial Engineering and
Management, Ben-Gurion University of the
Negev, Beer-Sheva, Israel
- Hugo Jacob  (6, 7, 8)
ENS Paris-Saclay, France
- Lars Jaffke (3, 8)
University of Bergen, Norway
- Satyabrata Jana  (2)
The Institute of Mathematical Sciences, HBNI,
Chennai, India
- Leon Kellerhals  (19)
Faculty IV, Institute of Software Engineering
and Theoretical Computer Science, Algorithmics
and Computational Complexity,
Technische Universität Berlin, Germany
- Rafael Kiesel  (32)
TU Wien, Austria
- Eun Jung Kim  (9)
Université Paris-Dauphine, PSL University,
CNRS UMR7243, LAMSADE, Paris, France
- Otto Kißig  (28)
Hasso Plattner Institut,
Universität Potsdam, Germany
- Tomohiro Koana  (19)
Faculty IV, Institute of Software Engineering
and Theoretical Computer Science, Algorithmics
and Computational Complexity,
Technische Universität Berlin, Germany
- Christian Komusiewicz  (20)
Fachbereich Mathematik und Informatik,
Philipps-Universität Marburg, Germany
- Viktoriia Korchemna  (15)
Algorithms and Complexity Group,
TU Wien, Austria
- Stefan Kratsch  (17)
Humboldt-Universität zu Berlin, Germany
- Ariel Kulik (12)
CISPA Helmholtz Center for Information
Security, Saarbrücken, Germany
- Pascal Kunz  (19)
Faculty IV, Institute of Software Engineering
and Theoretical Computer Science, Algorithmics
and Computational Complexity,
Technische Universität Berlin, Germany

Noleen Köhler  (9)
 Université Paris-Dauphine, PSL University,
 CNRS UMR7243, LAMSADE, Paris, France

Chao Liao (29)
 Huawei TCS Lab Shanghai, China

Paloma T. Lima  (8)
 IT University of Copenhagen, Denmark

Raul Lopes  (9)
 Université Paris-Dauphine, PSL University,
 CNRS UMR7243, LAMSADE, Paris, France

PinYan Lu (29)
 Huawei TCS Lab Shanghai, China

ZhiPeng Lv (29)
 School of Computer Science and Technology,
 Huazhong University of Science & Technology,
 China

Maher Mallem  (21)
 Sorbonne Université, CNRS, LIP6,
 F-75005 Paris, France

Dániel Marx  (12, 14, 22)
 CISA Helmholtz Center for Information
 Security, Saarbrücken, Germany

Yosuke Mizutani  (23)
 School of Computing, University of Utah,
 Salt Lake City, UT, USA

Hendrik Molter (18)
 Department of Industrial Engineering and
 Management, Ben-Gurion University of the
 Negev, Beer-Sheva, Israel

Nils Morawietz (20)
 Fachbereich Mathematik und Informatik,
 Philipps-Universität Marburg, Germany

Alix Munier-Kordon  (21)
 Sorbonne Université, CNRS, LIP6,
 F-75005 Paris, France

André Nichterlein  (25)
 Algorithmics and Computational Complexity,
 Technische Universität Berlin, Germany

Rolf Niedermeier  (25)
 Algorithmics and Computational Complexity,
 Technische Universität Berlin, Germany

Jelle J. Oostveen (24)
 Department of Information and Computing
 Sciences, Utrecht University, The Netherlands

Marcin Pilipczuk  (6)
 University of Warsaw, Poland

Michał Pilipczuk  (6)
 University of Warsaw, Poland

Nidhi Purohit (4, 13)
 Department of Informatics,
 University of Bergen, Norway

Clément Rambaud (11)
 DIENS, École Normale Supérieure, CNRS,
 PSL University, Paris, France

Amadeus Reinald  (31)
 École Normale Supérieure de Lyon, France

Mathis Rocton  (31)
 École Normale Supérieure de Lyon, France

Abhishek Sahu (2)
 Indian Institute of Technology Madras,
 Chennai, India

Govind S. Sankar  (22)
 Duke University, Durham, NC, USA

Saket Saurabh  (1, 13)
 The Institute of Mathematical Sciences, HBNI,
 Chennai, India;
 University of Bergen, Norway

Philipp Schepper  (12, 14, 22)
 CISA Helmholtz Center for Information
 Security, Saarbrücken, Germany

André Schidler  (32)
 TU Wien, Austria

Jonas Schmidt (28)
 Hasso Plattner Institut,
 Universität Potsdam, Germany

Christian Schulz  (26)
 Universität Heidelberg, Germany

Hjalmar Schulz (25)
 Algorithmics and Computational Complexity,
 Technische Universität Berlin, Germany

Dvir Shabtay (18)
 Department of Industrial Engineering and
 Management, Ben-Gurion University of the
 Negev, Beer-Sheva, Israel

Roohani Sharma  (14)
 Max Planck Institute for Informatics, SIC,
 Saarbrücken, Germany

Sebastian Siebertz  (30)
 Universität Bremen, Germany

Kirill Simonov (4)
 Algorithms and Complexity Group,
 TU Wien, Austria

Darren Strash  (26)
Hamilton College, Clinton, NY, USA

ZhouXing Su (29)
School of Computer Science and Technology,
Huazhong University of Science & Technology,
China

Blair D. Sullivan  (23)
School of Computing, University of Utah,
Salt Lake City, UT, USA

ZhiBo Sun (29)
School of Computer Science and Technology,
Huazhong University of Science & Technology,
China

Sylwester Swat  (27)
Institute of Computing Science,
Poznań University of Technology, Poland

Prafullkumar Tale  (14)
Indian Institute of Science Education and
Research, Pune, India

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

Erik Jan van Leeuwen (24)
Department of Information and Computing
Sciences, Utrecht University, The Netherlands

Magnus Wahlström  (11)
Department of Computer Science, Royal
Holloway, University of London, Egham, UK

Leo Wendt (28)
Hasso Plattner Institut,
Universität Potsdam, Germany

Christopher Weyand (25)
Karlsruhe Institute of Technology, Germany

Chen Wu (29)
Huawei TCS Lab Shanghai, China

Karol Węgrzycki  (12)
Saarland University, Saarbrücken, Germany;
Max Planck Institute for Informatics,
Saarbrücken, Germany

JunZhou Xu (29)
Huawei TCS Lab Shanghai, China

Meirav Zehavi  (1)
Ben-Gurion University of the Negev,
Beer-Sheva, Israel

QingYun Zhang (29)
School of Computer Science and Technology,
Huazhong University of Science & Technology,
China

ShunGen Zhang (29)
School of Computer Science and Technology,
Huazhong University of Science & Technology,
China

Anna Zych-Pawlewicz  (5)
University of Warsaw, Poland

A Finite Algorithm for the Realizability of a Delaunay Triangulation

Akanksha Agrawal ✉ 

Indian Institute of Technology Madras, Chennai, India

Saket Saurabh ✉ 

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

Meirav Zehavi ✉ 

Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

The *Delaunay graph* of a point set $P \subseteq \mathbb{R}^2$ is the plane graph with the vertex-set P and the edge-set that contains $\{p, p'\}$ if there exists a disc whose intersection with P is exactly $\{p, p'\}$. Accordingly, a triangulated graph G is *Delaunay realizable* if there exists a triangulation of the Delaunay graph of some $P \subseteq \mathbb{R}^2$, called a *Delaunay triangulation* of P , that is isomorphic to G . The objective of DELAUNAY REALIZATION is to compute a point set $P \subseteq \mathbb{R}^2$ that realizes a given graph G (if such a P exists). Known algorithms do not solve DELAUNAY REALIZATION as they are non-constructive. Obtaining a constructive algorithm for DELAUNAY REALIZATION was mentioned as an open problem by Hiroshima et al. [19]. We design an $n^{\mathcal{O}(n)}$ -time constructive algorithm for DELAUNAY REALIZATION. In fact, our algorithm outputs sets of points with *integer* coordinates.

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

Keywords and phrases Delaunay Triangulation, Delaunay Realization, Finite Algorithm, Integer Coordinate Realization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.1

Related Version *Full Version*: <https://arxiv.org/abs/2210.03932>

Funding *Akanksha Agrawal*: Supported by New Faculty Initiation Grant no. NFIG008972.

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

Meirav Zehavi: Supported by Israel Science Foundation grant no. 1176/18, and United States – Israel Binational Science Foundation grant no. 2018302.



1 Introduction

We study Delaunay graphs – through the lens of the well-known DELAUNAY REALIZATION problem – which are defined as follows. Given a point set $P \subseteq \mathbb{R}^2$, the *Delaunay graph*, $\mathcal{DG}(P)$, of P is the graph with vertex-set P and edge-set that consists of every pair (p, p') of points in P that satisfies the following condition: there exists a disc whose boundary intersects P only at p and p' , and whose interior does not contain any point in P . The point set $P \subseteq \mathbb{R}^2$ is in general position if it contains no four points from P on the boundary of a disc. If P is in general position, $\mathcal{DG}(P)$ is a triangulation, called a *Delaunay triangulation*, denoted by $\mathcal{DT}(P)$.¹ Otherwise, Delaunay triangulation and the notation $\mathcal{DT}(P)$, may refer

¹ We assume that $|P| \geq 4$, as otherwise, the problem that we consider, is solvable in polynomial time.



to any triangulation obtained by adding edges to $\mathcal{DG}(P)$. Thus, Delaunay triangulation of a point set P is unique if and only if $\mathcal{DG}(P)$ is a triangulation. An alternate characterization of Delaunay triangulations is that in such a triangulation, for any three points of a triangle of an interior face, the unique disc whose boundary contains these three points does not contain any other point in P .

The Delaunay graph of a point set is a planar graph [7], and triangulations of such graphs form an important subclass of the class of triangulations of a point set, also known as the class of maximal planar sub-divisions of the plane. Accordingly, efficient algorithms for computing a Delaunay triangulation for a given point set have been developed (see [7, 9, 18]). One of the main reasons underlying the interest in Delaunay triangulations is that any angle-optimal triangulation of a point set is actually a Delaunay triangulation of the point set. Here, optimality refers to the maximization of the smallest angle [7, 12]. This property is particularly useful when it is desirable to avoid “slim” triangles – this is the case, for example, when approximating a geographic terrain. Another main reason underlying the interest in Delaunay triangulations is that these triangulations are the duals of “Voronoi diagrams” (see [27]).

We are interested in a well-known problem which, in a sense, is the “opposite” of computing a Delaunay triangulation for a given point set. Here, rather than a point set, we are given a triangulated graph G . The graph G is *Delaunay realizable* if there exists $P \subseteq \mathbb{R}^2$ such that $\mathcal{DJ}(P)$ is isomorphic to G . Specifically, a point set $P \subseteq \mathbb{R}^2$ is said to *realize* G (as a Delaunay triangulation) if $\mathcal{DJ}(P)$ is isomorphic to G .² The problem of finding a point set that realizes G is called DELAUNAY REALIZATION. This problem is important not only theoretically, but also practically (see, e.g., [26, 32, 33]). Formally, it is defined as follows.

DELAUNAY REALIZATION

Input: A triangulation G on n vertices.

Output: If G is realizable as a Delaunay triangulation, then output $P \subseteq \mathbb{R}^2$ that realizes G (as a Delaunay triangulation). Otherwise, output NO.

Dillencourt [14] established necessary conditions for a triangulation to be realizable as a Delaunay triangulation. On the other hand, Dillencourt and Smith [16] established sufficient conditions for a triangulation to be realizable as a Delaunay triangulation. Dillencourt [15] gave a constructive proof showing that any triangulation where all vertices lie on the outer face is realizable as a Delaunay triangulation. Their approach, which results in an algorithm that runs in time $\mathcal{O}(n^2)$, uses a criterion concerning angles of triangles in a hypothetical Delaunay triangulation. In 1994, Sugihara [31] gave a simpler proof that all outerplanar triangulations are realizable as Delaunay triangulations. Later, in 1997, Lambert [22] gave a linear-time algorithm for realizing an outerplanar triangulation as a Delaunay triangulation. More recently, Alam et al. [3] gave yet another constructive proof for outerplanar triangulations.

Hodgson et al. [20] gave a polynomial-time algorithm for checking if a graph is realizable as a convex polyhedron with all vertices on a common sphere. Using this, Rivin [30] designed a polynomial-time algorithm for testing if a graph is realizable as a Delaunay triangulation. Independently, Hiroshima et al. [19] found a simpler polynomial-time algorithm, which relies on the proof of a combinatorial characterization of Delaunay realizable graphs. Both these results are non-constructive, i.e., they cannot output a point set P that realizes the input as a Delaunay triangulation, but only answer YES or NO. It is a long standing open problem to design a finite time algorithm for DELAUNAY REALIZATION.

² As G is triangulation, if $\mathcal{DJ}(P)$ is isomorphic to G , then $\mathcal{DJ}(P)$ is unique.

Obtaining a constructive algorithm for DELAUNAY REALIZATION was mentioned as an open problem by Hiroshima et al. [19]. We give the *first* exponential-time algorithm for the DELAUNAY REALIZATION problem. Our algorithm is based on the computation of two sets of polynomial constraints, defined by the input graph G . In both sets of constraints, the degrees of the polynomials are bounded by 2 and the coefficients are integers. The first set of constraints forces the points on the outer face to form a convex hull,³ and the second set of constraints ensures that for each edge in G , there is a disc containing only the endpoints of the edge. Roughly speaking, we prove that a triangulation is realizable as a Delaunay triangulation if and only if a point set realizing it as a Delaunay triangulation satisfies every constraint in our two sets of constraints. We proceed by proving that if a triangulation is realizable as a Delaunay triangulation, then there is $P \subseteq \mathbb{Z}^2$ such that $\mathcal{DT}(P)$ is isomorphic to G . This result is crucial to the design of our algorithm, not only for the sake of obtaining an integer solution, but for the sake of obtaining any solution. In particular, it involves a careful manipulation of a (hypothetical) point set in \mathbb{R}^2 , which allows to argue that it is “safe” to add new polynomials to our two sets of polynomials. Having these new polynomials, we are able to ensure that certain approximate solutions, which we can find in finite time, are actually exact solutions. We show that the special approximate solutions can be computed in polynomial time, and hence we actually solve the problem precisely. To find a solution satisfying our sets of polynomial constraints, our algorithm runs in time $n^{\mathcal{O}(n)}$. All other steps of the algorithm can be executed in polynomial time.

We believe that our contribution is a valuable step forward in the study of algorithms for geometric problems where one is interested in finding a solution rather than only determining whether one exists. Such studies have been carried out for various geometric problems (or their restricted versions) like UNIT-DISC GRAPH REALIZATION [23], LINE-SEGMENT GRAPH REALIZATION [21], PLANAR GRAPH REALIZATION (which is the same as COIN GRAPH REALIZATION) [11], CONVEX POLYGON INTERSECTION GRAPH REALIZATION [24], and DELAUNAY REALIZATION. (The above list is not comprehensive; for more details we refer the readers to given citations and references therein.) We note that the higher dimension analogue of DELAUNAY REALIZATION, called DELAUNAY SUBDIVISIONS REALIZATION, is $\exists\mathbb{R}$ -complete; for details on this generalization, see [1].

2 Preliminaries

In this section, we present basic concepts related to Geometry, Graph Theory and Algorithm Design, and establish some of the notation used throughout.

We refer the reader to the books [7, 28] for geometry-related terms that are not explicitly defined here. We denote the set of natural numbers by \mathbb{N} , the set of rational numbers by \mathbb{Q} and the set of real numbers by \mathbb{R} . By \mathbb{R}^+ we denote the set $\{x \in \mathbb{R} \mid x > 0\}$. For $n \in \mathbb{N}$, we use $[n]$ as a shorthand for $\{1, 2, \dots, n\}$. A point is an element in \mathbb{R}^2 . We work on Euclidean plane and the Cartesian coordinate system with the underlying bijective mapping of points in the Euclidean plane to vectors in the Cartesian coordinate system. For $p, q \in \mathbb{R}^2$, by $\text{dist}(p, q)$ we denote the distance between p and q in \mathbb{R}^2 .

Graphs. We use standard terminology from the book of Diestel [13] for graph-related terms not explicitly defined here. For a graph G , $V(G)$ and $E(G)$ denote the vertex and edge sets of G , respectively. For a vertex $v \in V(G)$, $d_G(v)$ denotes the *degree* of v , i.e the number of edges

³ The convex hull of a point set realizing G forms the outer face of its Delaunay triangulation.

incident on v , in the graph G . For an edge $(u, v) \in E(G)$, u and v are called the *endpoints* of the edge (u, v) . For $S \subseteq V(G)$, $G[S]$, and $G - S$ are the subgraphs of G induced on S and $V(G) \setminus S$, respectively. For $S \subseteq V(G)$, we let $N_G(S)$ and $N_G[S]$ denote the open and closed neighbourhoods of S in G , respectively. That is, $N_G(S) = \{v \mid (u, v) \in E(G), u \in S\} \setminus S$ and $N_G[S] = N_G(S) \cup S$. We drop the sub-script G from $d_G(v)$, $N_G(S)$, and $N_G[S]$ whenever the context is clear. A *path* in a graph is a sequence of distinct vertices v_0, v_1, \dots, v_ℓ such that (v_i, v_{i+1}) is an edge for all $0 \leq i < \ell$. Furthermore, such a path is called a v_0 to v_ℓ path. A graph is *connected* if for all distinct $u, v \in V(G)$, there is a u to v path in G . A graph which is not connected is said to be *disconnected*. A graph G is called *k-connected* if for all $X \subseteq V(G)$ such that $|X| < k$, $G - X$ is connected. A *cycle* in a graph is a sequence of distinct vertices v_0, v_1, \dots, v_ℓ such that $(v_i, v_{(i+1) \bmod (\ell+1)})$ is an edge for all $0 \leq i \leq \ell$. A cycle C in G is said to be a *non-separating cycle* in G if $G - V(C)$ is connected.

Planar Graphs and Plane Graphs. A graph G is called *planar* if it can be drawn on the plane such that no two edges cross each other except possibly at their endpoints. Formally, an embedding of a graph G is an injective function $\varphi : V(G) \rightarrow \mathbb{R}^2$ together with a set \mathcal{C} containing a continuous curve $C_{(u,v)}$ in the plane corresponding to each $(u, v) \in E(G)$ such that $\varphi(u)$ and $\varphi(v)$ are the endpoints of $C_{(u,v)}$. An embedding of a graph G is *planar* if distinct $C, C' \in \mathcal{C}$ intersect only at the endpoints – that is, any point in the intersection of C, C' is an endpoint of both C, C' . A graph that admits a planar embedding is a *planar graph*. Hereafter, whenever we say an embedding of a graph, we mean a planar embedding of it, unless stated otherwise. We often refer to a graph with a fixed embedding on the plane as a *plane graph*. For a plane graph G , the regions in $\mathbb{R}^2 \setminus G$ are called the *faces* of G . We denote the set of faces in G by $F(G)$. Note that since G is bounded and can be assumed to be drawn inside a sufficiently large disc, there is exactly one face in $F(G)$ that is unbounded, which is called the *outer face* of G . A face of G that is not the outer face is called an *inner face* of G . An embedding of a planar graph with the property that the boundary of every face (including the outer face) is a convex polygon is called a *convex drawing*. Below we state propositions related to planar and plane graphs that will be useful later.

► **Proposition 1** (Proposition 4.2.5 [13]). *For a 2-connected plane graph G , every face of G is bounded by a cycle.*

For a graph G and a face $f \in F(G)$, we let $V(f)$ denote the set of vertices in the cycle by which f is bounded. We often refer to $V(f)$ as the *face boundary* of f .

► **Proposition 2** (Proposition 4.2.10 [13]). *For a 3-connected planar graph, its face boundaries are precisely its non-separating induced cycles.*

Note that from Proposition 2, for a 3-connected planar graph and its planar embeddings $G_{\mathcal{P}}$ and $G_{\mathcal{P}'}$, it follows that $F(G_{\mathcal{P}}) = F(G_{\mathcal{P}'})$. (In the above we slightly abused the notation, and think of the sets $F(G_{\mathcal{P}})$ and $F(G_{\mathcal{P}'})$ in terms of their bounding cycles, rather than the regions of the plane.) Hence, it is valid to talk about $F(G)$ for a 3-connected planar graph G , even without knowing its embedding on the plane.

► **Proposition 3** (Tutte's Theorem [34], also see [8, 25]). *A 3-connected planar graph admits a convex embedding on the plane with any face as the outer face. Moreover, such an embedding can be found in polynomial time.*

For a plane graph G and a face $f \in F(G)$, by *stellating* f we mean addition of a new vertex v_f^* inside f and making it adjacent to all $v \in V(f)$. We note that stellating a face of a planar graph results in another planar graph [16].

Triangulations and Delaunay Triangulations. A *triangulation* is a plane graph where each inner face is bounded by a cycle on three vertices. A graph which is isomorphic to a triangulation is called a *triangulated graph*. We state the following simple but useful property of triangulations that will be exploited later.

► **Proposition 4.** *Let G be a triangulation with f^* being the outer face. Then, all the degree-2 vertices in G must belong to $V(f^*)$.*

► **Proposition 5** (Theorem 9.6 [7]). *For a point set $P \subseteq \mathbb{R}^2$ on n points, three points $p_1, p_2, p_3 \in P$ are vertices of the same face of the Delaunay graph of P if and only if the circle through p_1, p_2, p_3 contains no point of P in its interior.*

A *Delaunay triangulation* is any triangulation that is obtained by adding edges to the Delaunay graph. A Delaunay triangulation of a point set P is unique if and only if $\mathcal{DG}(P)$ is a triangulation, which is the case if P is in general position [7]. We refer to the Delaunay triangulation of a point set P by $\mathcal{DT}(P)$ (assuming it is unique, which is the case in our paper). A *triangulated graph* is *Delaunay realizable* if there exists a point set $P \subseteq \mathbb{R}^2$ such that $\mathcal{DT}(P)$ is isomorphic to G . If G has at most three points, then testing if it is Delaunay realizable is solvable in constant time. Also, we can compute an integer representation for it in constant time, if it exists. (Recall that while defining the general position assumption, we assumed that the point set has at least four points. This assumption does not cause any issues because we look for a realization of a graph which has at least four vertices.)

Polynomial Constraints. Let us now give some definitions and notation related to polynomials and sets of polynomial constraints (equalities and inequalities). We refer the reader to the books [5, 6] for algebra-related terms that are not explicitly defined here. For $t, n \in \mathbb{N}$ and a set C , a polynomial $\mathcal{P} = \sum_{i \in [t]} a_i \cdot (\prod_{j \in [n]} X_j^{d_j^i})$ on n variables and t terms is said to be a polynomial over C if for all $i \in [t]$, $j \in [n]$ we have $a_i \in C$ and $d_j^i \in \mathbb{N}$. Furthermore, the *degree* of the polynomial \mathcal{P} is defined to be $\max_{i \in [t]} (\sum_{j \in [n]} d_j^i)$. We denote the set of polynomials on n variables X_1, X_2, \dots, X_n with coefficients in C by $C[X_1, X_2, \dots, X_n]$.

A *polynomial constraint* \mathcal{C} on n variables with coefficients from $C \subseteq \mathbb{R}$ is a sequence $\mathcal{P}\Delta 0$, where $\mathcal{P} \in C[X_1, X_2, \dots, X_n]$ and $\Delta \in \{=, \geq, >, \leq, <\}$. The *degree* of such a constraint is the degree of \mathcal{P} , and it is said to be an equality constraint if Δ is ‘=’. We say that the constraint is *satisfied* by an element $(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n) \in \mathbb{R}^n$ if $\mathcal{P}(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)\Delta 0$.⁴ Given a set \mathcal{C} of polynomial constraints on n variables, X_1, X_2, \dots, X_n , and with coefficients from $C \subseteq \mathbb{R}$, we say that an element $(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n) \in \mathbb{R}^n$ *satisfies* \mathcal{C} if for all $\mathcal{C} \in \mathcal{C}$, we have that $(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)$ satisfies \mathcal{C} . In this case, $(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)$ is also called a *solution* of \mathcal{C} . Furthermore, \mathcal{C} is said to be *satisfiable* (in \mathbb{R}) if there exists $(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n) \in \mathbb{R}^n$ satisfying \mathcal{C} .

Below we state a result regarding a method for solving a finite set of polynomial constraints, which will be used by our algorithm. This result is a direct implication of Propositions 3.8.1 and 4.1 in [29] (see also [6]).

► **Proposition 6** (Propositions 3.8.1 and 4.1 in [29]). *Let \mathcal{C} be a set of m polynomial constraints of degree 2 on n variables with coefficients in \mathbb{Z} whose bitsizes are bounded by $\mathcal{O}(1)$. Then, in time $m^{\mathcal{O}(n)}$ we can decide if \mathcal{C} is satisfiable in \mathbb{R} . Moreover, if \mathcal{C} is satisfiable in \mathbb{R} , then in time $m^{\mathcal{O}(n)}$ we can also compute a (satisfiable) set $\hat{\mathcal{C}}$ of n polynomial constraints, $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_n$, with coefficients in \mathbb{Z} , where for all $i \in [n]$, we have that \mathcal{C}_i is an equality constraint on X_i (only), and a solution of $\hat{\mathcal{C}}$ is also a solution of \mathcal{C} .*

⁴ Here, $\mathcal{P}(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)$ is the evaluation of \mathcal{P} , where the variable X_i is assigned the value \bar{x}_i , for $i \in [n]$.

3 Restricted-Delaunay Realization: Generating Polynomials

In this section, we generate a set of polynomials that encodes the realizability of a triangulation as a Delaunay triangulation in the case where the outer face of the Delaunay triangulation is known. More precisely, we suppose that the outer faces of G and the Delaunay triangulation are the same. For the general case where we might not know a priori which is the face in G that is supposed to be the outer face of the Delaunay triangulation (this is the case when G is a maximal planar graph), we will “guess” the outer face and then use our restricted version to solve the problem. Formally, we solve the following problem.

RESTRICTED-DELAUNAY REALIZATION (RES-DR)

Input: A triangulation G with outer face f^* .

Output: A set of polynomial constraints $\text{Const}(G)$ such that $\text{Const}(G)$ is satisfiable if and only if G is realizable as a Delaunay triangulation with f^* as the outer face.

Let (G, f^*) be an instance of RES-DR, and let n denote $|V(G)|$. We denote $V(G)$ by the set $\{v_1, v_2, \dots, v_n\}$. Note that except possibly f^* , each of the faces of G is bounded by a cycle on three vertices. With each $v_i \in V(G)$ we associate two variables, X_i and Y_i , which correspond to the values of the x and y coordinates of v_i in the plane. Furthermore, we let P_i denote the vector (X_i, Y_i) . We let \bar{X} denote the value that some solution of $\text{Const}(G)$ assigns to the variable X . Accordingly, we denote $\bar{P}_i = (\bar{X}_i, \bar{Y}_i)$. For the sake of clarity, we sometimes abuse the notation \bar{P}_i by letting it denote both \bar{P}_i and P_i (this is done in situations where both interpretations are valid).

Our algorithm is based on the computation of two sets of polynomial constraints of bounded degree and integer coefficients. Informally, we have one set of inequalities which ensures that the points to which vertices of f^* are mapped are in convex position, and another set of inequalities which ensures that for each $(v_i, v_j) \in E(G)$, there exists a disc containing (\bar{X}_i, \bar{Y}_i) and (\bar{X}_j, \bar{Y}_j) on its boundary and excluding all other points (\bar{X}_k, \bar{Y}_k) . (While other sets of inequalities may be devised to ensure these properties, we subjectively found the two sets presented here the easiest to employ.)

3.1 Inequalities Ensuring that the Outer Face Forms the Convex Hull

We first generate the set of polynomial constraints ensuring that the points associated with the vertices in f^* form the convex hull of the output point set. Here, we also ensure that the vertices in f^* have the same cyclic ordering (given by the cycle bounding f^*) as the points corresponding to them have in the convex hull. Note that the edges of the convex hull are present in any Delaunay triangulation [7]. Moreover, the convex hull of a point set forms the outer face of its Delaunay triangulation. To formulate our equations, we rely on the notions of *left and right turns*. Their definitions are the same as those in the book [10], which uses *cross product* to determine whether a turn is a left turn or a right turn. For the sake of clarity, we also explain these notions below.

Left and Right Turns. Consider two vectors (or points) \bar{P}_1 and \bar{P}_2 , denoting some (x_1, y_1) and (x_2, y_2) , respectively. The cross product $\bar{P}_1 \times \bar{P}_2$ of \bar{P}_1 and \bar{P}_2 is defined as follows.

$$\bar{P}_1 \times \bar{P}_2 = \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = x_1 y_2 - x_2 y_1.$$

If $\bar{P}_1 \times \bar{P}_2 > 0$, then \bar{P}_1 is said to be *clockwise* from \bar{P}_2 (with respect to the origin $(0, 0)$). Else, if $\bar{P}_1 \times \bar{P}_2 < 0$, then \bar{P}_1 is said to be *counterclockwise* from \bar{P}_2 . Otherwise (if $\bar{P}_1 \times \bar{P}_2 = 0$), \bar{P}_1 and \bar{P}_2 are said to be *collinear*. Given line segments $\overline{P_0 P_1}$ and $\overline{P_1 P_2}$, we

would like to determine the type of turn taken by the angle $\angle P_0P_1P_2$. To this end, we check whether the directed segment $\overline{P_0P_2}$ is clockwise or counterclockwise from $\overline{P_0P_1}$. Towards this, we first compute the cross product $(\bar{P}_2 - \bar{P}_0) \times (\bar{P}_1 - \bar{P}_0)$. If $(\bar{P}_2 - \bar{P}_0) \times (\bar{P}_1 - \bar{P}_0) > 0$, then $\overline{P_0P_2}$ is clockwise from $\overline{P_0P_1}$, and we say that we take a *right turn* at \bar{P}_1 . Else, if $(\bar{P}_2 - \bar{P}_0) \times (\bar{P}_1 - \bar{P}_0) < 0$, then $\overline{P_0P_2}$ is counterclockwise from $\overline{P_0P_1}$, and we say that we take a *left turn* at \bar{P}_1 . Otherwise, we make *no turn* at \bar{P}_1 . Note that the computation of $(\bar{P}_2 - \bar{P}_0) \times (\bar{P}_1 - \bar{P}_0)$ can be done as follows.

$$(\bar{P}_2 - \bar{P}_0) \times (\bar{P}_1 - \bar{P}_0) = \begin{vmatrix} x_2 - x_0 & x_1 - x_0 \\ y_2 - y_0 & y_1 - y_0 \end{vmatrix} = x_2y_1 - x_2y_0 - x_0y_1 - x_1y_2 + x_1y_0 + x_0y_2.$$

The Polynomials. For three vectors (or points) $\bar{P}_0 = (x_0, y_0)$, $\bar{P}_1 = (x_1, y_1)$ and $\bar{P}_2 = (x_2, y_2)$, by $\text{Con}(\bar{P}_0, \bar{P}_1, \bar{P}_2)$ we denote the polynomial $x_2y_1 - x_2y_0 - x_0y_1 - x_1y_2 + x_1y_0 + x_0y_2$. Note that $\text{Con}(\bar{P}_0, \bar{P}_1, \bar{P}_2)$ determines whether we have a right, left or no turn at \bar{P}_1 .

Before stating the constraints based on these polynomials, let us recall the well-known fact stating that a non-intersecting polygon is convex if and only if every interior angle of the polygon is less than 180° . While we ensure the non-intersecting constraint later, the characterization of each angle being less than 180° is the same as taking a *right* (or *left*) turn at P_j for every three consecutive points P_i, P_j and P_k of the polygon. We will use this characterization to enforce convexity on the points corresponding to the vertices in $V(f^*)$. Let us also recall that f^* is a cycle C^* in G . Next, whenever we talk about consecutive vertices in C^* , we always follow clockwise direction.

For every three consecutive vertices v_i, v_j and v_k in C^* , we add the following inequality:

$$\text{Con}(P_i, P_j, P_k) > 0.$$

These inequalities ensure that in any output point set, the points corresponding to vertices in $V(f^*)$ are in convex position (together with the non-intersecting condition to be ensured later).

Next, we further need to ensure that all the points which correspond to vertices in $V(G) \setminus V(f^*)$ belong to the interior of the convex hull formed by the points corresponding to vertices in $V(f^*)$ and the polygon formed by the points corresponding to $V(f^*)$ is non-self intersecting. For this purpose, we crucially rely on the following property of convex hulls (or convex polygons): For any edge of the convex hull, it holds that all the points, except for the endpoints of the edge, are located in one of the sides of the edge. Using this property, we know that for any two consecutive vertices v_i and v_j in C^* , all points are on one side of the line associated with v_i and v_j . Since at each $v_i \in C^*$ we ensure that we turn right, we must have all the points located on the right of the line defined by the edge (v_i, v_j) . This, in turn, implies that for every pair of consecutive vertices v_i and v_j in C^* , for any vertex $v_k \in V(G) \setminus V(f^*)$, we must be turning left at v_k (according to the ordered triplet (v_i, v_k, v_j)). Hence, we add the following inequalities:

$$\text{Con}(P_i, P_k, P_j) < 0.$$

where v_i and v_j are consecutive vertices of C^* and $v_k \in V(G) \setminus \{v_i, v_j\}$.

We denote the set of inequalities generated above by $\text{Con}(G)$.

3.2 Inequalities Guaranteeing Existence of Edges

For each edge $(v_i, v_j) \in E(G)$, we add two new variables, X_{ij} and Y_{ij} , to indicate the coordinates of the centre of the disc that realizes the edge (v_i, v_j) . There might exist many discs that realize the edge (v_i, v_j) , but we are interested in only one such disc, say C_{ij} .

Note that C_{ij} should contain (\bar{X}_i, \bar{Y}_i) and (\bar{X}_j, \bar{Y}_j) on its boundary, and it should not contain any (\bar{X}_k, \bar{Y}_k) such that $k \notin \{i, j\}$. Towards this, for each edge $(v_i, v_j) \in E(G)$, we add a set of inequalities that we denote by $\text{Dis}(v_i, v_j)$. Note that the radius r_{ij} of C_{ij} is given by $r_{ij}^2 = (X_i - X_{ij})^2 + (Y_i - Y_{ij})^2$ (if (X_i, Y_i) lies on the boundary) and by $r_{ij}^2 = (X_j - X_{ij})^2 + (Y_j - Y_{ij})^2$ (if (X_j, Y_j) lies on the boundary). Therefore, we want to ensure the following.

$$\begin{aligned} (X_j - X_{ij})^2 + (Y_j - Y_{ij})^2 &= (X_i - X_{ij})^2 + (Y_i - Y_{ij})^2 \\ \Rightarrow X_i^2 - X_j^2 + Y_i^2 - Y_j^2 - 2X_{ij}X_i - 2Y_{ij}Y_i + 2X_{ij}X_j + 2Y_{ij}Y_j &= 0. \end{aligned}$$

Hence, we add the above constraint to $\text{Dis}(v_i, v_j)$. Further, we want to ensure that for each $k \in [n] \setminus \{i, j\}$, (X_k, Y_k) does not belong to C_{ij} . Therefore, for each $k \in [n] \setminus \{i, j\}$, the following must hold.

$$\begin{aligned} (X_k - X_{ij})^2 + (Y_k - Y_{ij})^2 - (X_i - X_{ij})^2 - (Y_i - Y_{ij})^2 &> 0 \\ \Rightarrow X_k^2 - X_i^2 + Y_k^2 - Y_i^2 - 2X_{ij}X_k - 2Y_{ij}Y_k + 2X_{ij}X_i + 2Y_{ij}Y_i &> 0 \end{aligned}$$

Hence, we also add the above constraint to $\text{Dis}(v_i, v_j)$ for $k \in [n] \setminus \{i, j\}$. Overall, we denote $\text{Dis}(G) = \bigcup_{(v_i, v_j) \in E(G)} \text{Dis}(v_i, v_j)$. This completes the description of all inequalities relevant to this section.

3.3 Correctness

Let us denote $\text{Const}(G) = \text{Con}(G) \cup \text{Dis}(G)$. We begin with the following observation. Here, to bound the number of variables, we rely on the fact that G is a planar graph, its number of edges is upper bounded by $3n$, and hence in total we introduced less than $8n$ variables.

► **Observation 7.** *The number of constraints in $\text{Const}(G)$ is bounded by $\mathcal{O}(n^2)$ and the total number of variables is bounded by $\mathcal{O}(n)$. Moreover, each constraint in $\text{Const}(G)$ is of degree 2, and its coefficients belong to $\{-2, -1, 0, 1, 2\}$.*

Now, we state the central lemma establishing the correctness of our algorithm for RES-DR.

► **Lemma 8.** *A triangulation G with outer face f^* is realizable as a Delaunay triangulation with f^* as its outer face if and only if $\text{Const}(G)$ is satisfiable.*

Proof. Let G be a triangulation realizable as a Delaunay triangulation with f^* as the outer face of the Delaunay triangulation. Then, there exists $P \subseteq \mathbb{R}^2$ such that $\mathcal{DT}(P)$ is isomorphic to G and f^* is the outer face of $\mathcal{DT}(P)$. Furthermore, for each $(\bar{P}_i, \bar{P}_j) \in E(\mathcal{DT}(P))$, there exists a disc C_{ij} which contains \bar{P}_i and \bar{P}_j on its boundary, and which contains no point \bar{P}_k , $k \in [n] \setminus \{i, j\}$, on neither its boundary nor its interior. We let \bar{P}_{ij} denote the centre of C_{ij} . Let \mathcal{P} be the vector assigning \bar{P}_i to the vertex $v_i \in V(G)$ and \bar{P}_{ij} to the centre of the disc C_{ij} . We note that the vertices of f^* are in convex position in $\mathcal{DT}(P)$. Clearly, we then have that \mathcal{P} satisfies $\text{Const}(G)$. This concludes the proof of the forward direction.

In the reverse direction, consider some \mathcal{P} that satisfies $\text{Const}(G)$. By our polynomial constraints, \mathcal{P} assigns some \bar{P}_i to each vertex $v_i \in V(G)$, such that for each edge $(v_i, v_j) \in E(G)$, it lets \bar{P}_{ij} be the centre of a disc C_{ij} containing \bar{P}_i and \bar{P}_j (on its boundary) and no point \bar{P}_k where $k \notin \{i, j\}$. Further, we let $P = \{\bar{P}_i \mid i \in [n]\}$. By the construction of $\text{Const}(G)$, it follows that if $(v_i, v_j) \in E(G)$, then $(\bar{P}_i, \bar{P}_j) \in \mathcal{DT}(P)$ and the points in P corresponding to vertices in $V(f^*)$ form the convex hull of P . This implies that the points

corresponding to the vertices in $V(f^*)$ are on the outer face of $\mathcal{DT}(P)$. From Theorem 9.1 in [7], it follows that $|E(G)| \leq |E(\mathcal{DT}(P))|$. Thus, $E(G) = E(\mathcal{DT}(P))$. This concludes the proof of the reverse direction. \blacktriangleleft

The next theorem follows from the construction of $\text{Const}(G)$, Observation 7 and Lemma 8.

► Theorem 9. *Let G be a triangulation on n vertices with f^* as the outer face. Then, in time $\mathcal{O}(n^2)$, we can output a set of polynomial constraints $\text{Const}(G)$ such that G is realizable as a Delaunay triangulation with f^* as its outer face if and only if $\text{Const}(G)$ is satisfiable. Moreover, $\text{Const}(G)$ consists of $\mathcal{O}(n^2)$ constraints and $\mathcal{O}(n)$ variables, where each constraint is of degree 2 and with coefficients only from $\{-2, -1, 0, 1, 2\}$.*

4 Restricted-Delaunay Realization: Replacing Points by Discs

Let G be a triangulation on n vertices with f^* as its outer face. Suppose that G is realizable as a Delaunay triangulation where the points corresponding to vertices in $V(f^*)$ belong to the outer face. By Theorem 9, it follows that $\text{Const}(G)$ is satisfiable. Let n^* denote the number of variables of $\text{Const}(G)$. Since $\text{Const}(G)$ is satisfiable, there exists \mathcal{Q} satisfying $\text{Const}(G)$. Let \bar{Q}_i be the value assigned to the vertex $v_i \in V(G)$ for $i \in [n]$. Let $Q = \{\bar{Q}_i \mid v_i \in V(G)\}$. Recall that apart from assigning points in the plane to vertices in $V(G)$, \mathcal{Q} assigns to each $(v_i, v_j) \in E(G)$, a point \bar{Q}_{ij} corresponding to the centre of some disc, say C'_{ij} , containing \bar{Q}_i, \bar{Q}_j on its boundary and excluding all other points in Q .

In this section, we prove that for any given $\beta \in \mathbb{R}^+$, there exists a set of discs of radius β , one for each vertex in $V(G)$, with the following property. If for every $v_i \in V(G)$, we choose some point \bar{P}_{C_i} inside or on the boundary of its disc C_i , we get that $\mathcal{DT}(Q)$ and the Delaunay triangulation of our set of chosen points are isomorphic.

We start with two simple observations, where the second directly follows from the definition of the constraints in $\text{Const}(G)$.

► Observation 10. *Let $(a, b), (x, y) \in \mathbb{R}^2$ be two points and $\alpha \in \mathbb{R}^+$. Then, $\text{dist}((\alpha a, \alpha b), (\alpha x, \alpha y)) = \alpha \cdot \text{dist}((a, b), (x, y))$.*

► Observation 11. *Let G be a triangulation on n vertices with f^* as its outer face. If \mathcal{Q} is a solution of $\text{Const}(G)$, then for any $\alpha \in \mathbb{R}^+$, it holds that $\alpha\mathcal{Q}$ also satisfies $\text{Const}(G)$.*

In what follows, we create a point set P such that $\mathcal{DT}(P)$ is isomorphic to G , where the points corresponding to vertices in $V(f^*)$ form the outer face of $\mathcal{DT}(P)$. We then show that this point set defines a set of discs with the desired property – for each $v_i \in V(G)$, it defines one disc C_i with \bar{P}_i as centre and with radius $r^* \geq \beta > 0$ (to be determined), such that, roughly speaking, each point of C_i is a valid choice for v_i . For this purpose, we first define the real numbers, d_N , d_C , and d_A , which are necessary to determine r^* and P . Informally, d_N ensures that the discs we create around vertices do not intersect, d_C will be used to ensure existence of specific edges, d_A will be used to ensure that “convex hull property” is satisfied. These (positive) real numbers are defined as follows.

- Let $d_N = \min_{i,j \in [n], i \neq j} \{\text{dist}(\bar{Q}_i, \bar{Q}_j)\}$, i.e., d_N is the minimum distance between any pair of distinct points in Q .
- Let $d_C = \min_{i,j,k \in [n], i \neq j, i \neq k, j \neq k} \{\text{dist}(C'_{ij}, \bar{Q}_k)\}$, i.e., d_C denotes the minimum distance between a point corresponding to a vertex in $V(G)$ and a disc realizing an edge non-incident to it. (Recall that C'_{ij} is defined at the beginning of this section.) Note that $d_C > 0$ because in the above definition of d_C , we have only considered those disc and point pairs where the point lies outside the disc.

► **Theorem 13.** *Let G be a triangulation on n vertices with f^* as its outer face, realizable as a Delaunay triangulation where the points corresponding to vertices of f^* lie on the outer face. Moreover, let \mathcal{Q} be a solution of $\text{Const}(G)$ and $\beta \in \mathbb{R}^+$. Then, there is a solution \mathcal{P} of $\text{Const}(G)$, assigning a set of points $P \subseteq \mathbb{R}^2$ to vertices of G , such that for each $v_i \in V(G)$, there exists a disc C_i with centre \bar{P}_i and radius at least β for which the following condition holds. For any $P' = \{\bar{P}'_i \mid \bar{P}'_i \in C_i, i \in [n]\}$, it holds that $\mathcal{DT}(P')$ is isomorphic to $\mathcal{DT}(P)$, and the points corresponding to vertices of f^* lie on the outer face of $\mathcal{DT}(P')$.*

Proof. The proof of theorem follows directly from the construction of r^* , the discs C_i for $i \in [n]$, and Lemma 12. ◀

5 Delaunay Realization: Integer Coordinates

In this section, we prove our main theorem:

► **Theorem 14.** *Given a triangulation G on n vertices, in time $n^{\mathcal{O}(n)}$ we can either output a point set $P \subseteq \mathbb{Z}^2$ such that G is isomorphic to $\mathcal{DT}(P)$, or correctly conclude that G is not Delaunay realizable.*

The Outer Face of the Output. First, we explain how to identify the outer face f^* of the output (in case the output should not be NO). For this purpose, let f_{out} denote the outer face of G (according to the embedding of the triangulation G , given as the input). Recall our assumption that $n \geq 4$. Let us first consider the case where G is not a maximal planar graph, i.e., f_{out} consists of at least four vertices. Suppose that the output is not NO. Then, for any point set $P \subseteq \mathbb{R}^2$ that realizes G as a Delaunay triangulation, it holds that the points corresponding to the vertices of f_{out} form the outer face of $\mathcal{DT}(P)$. Thus, in this case, we simply set $f^* = f_{\text{out}}$. Next, consider the case where G is a maximal planar graph. Again, suppose that the output is not NO. Then, for a point set $P \subseteq \mathbb{R}^2$ that realizes G as a Delaunay triangulation, the outer face of $\mathcal{DT}(P)$ need not be the same as f_{out} . To handle this case, we “guess” the outer face of the output (if it is not NO). More precisely, we examine each face f of G separately, and attempt to solve the “integral version” of RES-DR with f^* set to f , and where G is embedded with f^* , rather than f_{out} , as its outer face. Here, note that a maximal planar graph is 3-connected [35], and therefore, by Proposition 3, we can indeed compute an embedding of G with f^* as the outer face.

The number of iterations is bounded by $\mathcal{O}(n)$ (since the number of faces of G is bounded by $\mathcal{O}(n)$). Thus, from now on, we may assume that we seek only Delaunay realizations of G where the outer face is the same as the outer face of G (that we denote by f^*).

Sieving NO-Instances. We compute the set $\text{Const}(G)$ as described in Section 3. From Theorem 9, we know that G is realizable as a Delaunay triangulation with the points corresponding to f^* on the outer face if and only if $\text{Const}(G)$ is satisfiable. Using Proposition 6, we check whether $\text{Const}(G)$ is satisfiable, and if the answer is negative, then we return NO. Thus, we next focus on the following problem.

INTEGRAL DELAUNAY REALIZATION (INT-DR)

Input: A triangulation G with outer face f^* that is realizable as a Delaunay triangulation with outer face f^* .

Output: A point set $P \subseteq \mathbb{Z}^2$ realizing G as a Delaunay triangulation with outer face f^* .

Similarly, we define the intermediate RATIONAL DELAUNAY REALIZATION (RATIONAL-DR) problem – here, however, $P \subseteq \mathbb{Q}^2$ rather than \mathbb{Z}^2 . To prove Theorem 14, it is sufficient to prove the following result, which is the objective of the rest of this paper.

► **Lemma 15.** *INT-DR is solvable in time $n^{\mathcal{O}(n)}$.*

In what follows, we crucially rely on the fact that by Theorem 13, for all $\beta \in \mathbb{R}^+$, there is a solution \mathcal{P} of $\text{Const}(G)$ that assigns a set of points $P \subseteq \mathbb{R}^2$ to the vertices of G , such that for each $v_i \in V(G)$, there exists a disc C_i with radius at least β , satisfying the following condition: For any $P' = \{\bar{P}'_i \mid P'_i \in C_i, i \in [n]\}$, it holds that $\mathcal{DT}(P')$ is isomorphic to G with points corresponding to the vertices in f^* on the outer face (in the same order as in f^*).

As it would be cleaner to proceed while working with squares, we need the next observation.

► **Observation 16.** *Every disc C with radius at least 2 contains a square of side length at least 2 and with the same centre.*

We next extend $\text{Const}(G)$ to a set $\text{ConstSqu}(G)$, which explicitly ensures that there exists a square around each point in the solution such that the point can be replaced by any point in the square. Thus, rather than discs of radius 2 (whose existence, in some solution, is proven by choosing $\beta = 2$), we consider squares with side length 2 given by Observation 16, and force our constraints to be satisfied at the corner points of the squares. For this purpose, for each $v_i \in V(G)$, apart from adding constraints for the point $P_i = (X_i, Y_i)$ (which can be regarded as a disc of radius 0 in the previous setting), we also have constraints for the corner points of the square of side length 2 whose centre is P_i . For technical reasons, we also add constraints for the intersection points of perpendicular bisectors. For any constraint where P_i appears, we make copies for the points $P_i = (X_i, Y_i)$, $P_i^1 = (X_i - 1, Y_i - 1)$, $P_i^2 = (X_i - 1, Y_i + 1)$, $P_i^3 = (X_i + 1, Y_i - 1)$, $P_i^4 = (X_i + 1, Y_i + 1)$, $P_i^5 = (X_i - 1, Y_i)$, $P_i^6 = (X_i, Y_i + 1)$, $P_i^7 = (X_i + 1, Y_i)$, $P_i^8 = (X_i, Y_i - 1)$.⁶

Inequalities that ensure the outer face forms the convex hull. We generate the set of constraints that ensure the points corresponding to vertices in $V(f^*)$ form a convex hull of the output point set. Let C^* be the cycle of the outer face f^* . Whenever we say consecutive vertices in C^* , we always follow clockwise direction. For three consecutive vertex v_i, v_j and v_k in C^* , for every $Z_i \in \{P_i\} \cup \{P_i^\ell \mid \ell \in [8]\}$, $Z_j \in \{P_j\} \cup \{P_j^\ell \mid \ell \in [8]\}$ and $Z_k \in \{P_k\} \cup \{P_k^\ell \mid \ell \in [8]\}$, we add the inequality $\text{Con}(Z_i, Z_j, Z_k) > 0$. This ensures that the points corresponding to vertices in $V(f^*)$ are in convex position in any output point set. Further, we want all the points which correspond to the vertices in $V(G) \setminus V(f^*)$ to be in the interior of the convex hull formed by the points corresponding to vertices in $V(f^*)$. To achieve this, for each pair of vertices v_i, v_j that are consecutive vertices of C^* , $v_k \in V(G) \setminus \{v_i, v_j\}$, $Z_i \in \{P_i\} \cup \{P_i^\ell \mid \ell \in [8]\}$, $Z_j \in \{P_j\} \cup \{P_j^\ell \mid \ell \in [8]\}$ and $Z_k \in \{P_k\} \cup \{P_k^\ell \mid \ell \in [8]\}$, we add $\text{Con}(Z_i, Z_k, Z_j) < 0$. We call the above set of polynomial constraints $\text{ConSqu}(G)$.

Inequalities that guarantee existence of edges. For each edge $(v_i, v_j) \in E(G)$, we add three new variables, namely X_{ij}, Y_{ij} and r_{ij} . These newly added variables will correspond to the centre and radius of a disc that realizes the edge (v_i, v_j) . There might exist many such discs, but we are interested in only one such disc. In particular, (X_{ij}, Y_{ij}) corresponds to centre of one such discs, say C_{ij} , with radius r_{ij} , containing all the points in $\{P_i\} \cup \{P_i^\ell \mid \ell \in [8]\}$ and $\{P_j\} \cup \{P_j^\ell \mid \ell \in [8]\}$ but none of the points in $\{P_k \mid k \in [n] \setminus \{i, j\}\} \cup \{P_k^\ell \mid \ell \in [8], k \in [n] \setminus \{i, j\}\}$. Towards this, we add a set of inequalities for each edge $(v_i, v_j) \in E(G)$, which we will denote by $\text{DisSqu}(v_i, v_j)$. For each $Z \in \{P_i, P_j\} \cup \{P_i^\ell, P_j^\ell \mid \ell \in [8]\}$, we add the following inequalities to $\text{DisSqu}(v_i, v_j)$, ensuring that C_{ij} contains $Z = (Z_X, Z_Y)$.

$$Z_X^2 + X_{ij}^2 - 2Z_X X_{ij} + Z_Y^2 + Y_{ij}^2 - 2Z_Y Y_{ij} - r_{ij}^2 \leq 0.$$

⁶ We remark that we do not create new variables for the corresponding x - and y -coordinates for points P_i^ℓ , for $i \in [n]$.

Further, we want to ensure that for each $k \in [n] \setminus \{i, j\}$, $Z \in \{P_k\} \cup \{P_k^\ell \mid \ell \in [8]\}$ does not belong to C_{ij} . Hence, for each such $Z = (Z_X, Z_Y)$, the following must hold.

$$Z_X^2 + X_{ij}^2 - 2Z_X X_{ij} + Z_Y^2 + Y_{ij}^2 - 2Z_Y Y_{ij} - r_{ij}^2 > 0.$$

Hence, we add the above constraint to $\text{DisSqu}(v_i, v_j)$ for $k \in [n] \setminus \{i, j\}$. We denote $\text{DisSqu}(G) = \bigcup_{(v_i, v_j) \in E(G)} \text{DisSqu}(v_i, v_j)$.

This completes the description of all the constraints we need. We let $\text{ConstSqu}(G) = \text{ConSqu}(G) \cup \text{DisSqu}(G)$. We let n^* denote the number of variables appearing in $\text{ConstSqu}(G)$. Note that $n^* = \mathcal{O}(n)$ and the number of constraints in $\text{ConstSqu}(G)$ is bounded by $\mathcal{O}(n^2)$.

► **Theorem 17.** *Let G be a triangulation on n vertices with f^* as the outer face. Then, in time $\mathcal{O}(n^2)$ we can find a set of polynomial constraints $\text{ConstSqu}(G)$ such that G is realizable as a Delaunay triangulation with f^* as its outer face if and only if $\text{ConstSqu}(G)$ is satisfiable. Moreover, $\text{ConstSqu}(G)$ consists of $\mathcal{O}(n^2)$ constraints and $\mathcal{O}(n)$ variables, where each constraint is of degree 2 and with coefficients only from $\{-10, -9, \dots, 10\}$.*

Proof. Follows from the construction of $\text{ConstSqu}(G)$, Lemma 8, and Theorems 13. ◀

Having proved Theorem 17, we use Proposition 6 to decide in time $n^{\mathcal{O}(n)}$ if $\text{ConstSqu}(G)$ is satisfiable. Recall that if the answer is negative, then we returned NO. We compute a “good” approximate solution as we describe next. First, by Proposition 6, in time $n^{\mathcal{O}(n)}$ we compute a (satisfiable) set \mathcal{C} of n^* polynomial constraints, $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_{n^*}$, with coefficients in \mathbb{Z} , where for all $i \in [n]$, we have that \mathcal{C}_i is an equality constraint on the variable indexed i (only), and a solution of \mathcal{C} is also a solution of $\text{ConstSqu}(G)$. Next, we would like to find a “good” rational approximation to the solution of \mathcal{C} . Later we will prove that such an approximate solution is actually an *exact* solution to our problem.

For $\delta > 0$, a δ rational approximate solution \mathcal{S} for a set of polynomial equality constraints is an assignment to the variables, for which there exists a solution \mathcal{S}^* , such that for any variable X , the (absolute) difference between the assignment to X by \mathcal{S} and the assignment to X by \mathcal{S}^* is at most δ .⁷ We follow the approach of Arora et al. [4] to find a δ rational approximation to a solution for a set of polynomial equality constraints with $\delta = 1/2$. This approach states that using Renegar’s algorithm [29] together with binary search, with search range bound given by Grigor’ev and Vorobjov [17], we can find a rational approximation to a solution of a set of polynomial equality constraints with accuracy up to δ in time $(\tau + n' + m'^{n'} + \log(1/\delta))^{\mathcal{O}(1)}$ where τ is the maximum bitsize of a coefficient, n' is the number of variables and m' is the number of constraints. In this manner, we obtain in time $n^{\mathcal{O}(n)}$ a rational approximation \mathcal{S} to the solution of \mathcal{C} with accuracy $1/2$. By Theorem 17, \mathcal{S} is also a rational approximation to a solution of $\text{ConstSqu}(G)$ with accuracy $1/2$. We let $\bar{P}_{S_i} = (\bar{X}_{S_i}, \bar{Y}_{S_i})$ denote the value that \mathcal{S} assigns to (X_i, Y_i) (corresponding to the vertex $v_i \in V(G)$). Further we let $P_S = \{\bar{P}_{S_i} \mid i \in [n]\}$. In the following lemma, we analyze $\mathcal{DT}(P_S)$.

► **Lemma 18 (♠).** *The triangulation G is isomorphic to $\mathcal{DT}(P_S)$ where points corresponding to vertices in f^* form the outer face (in that order). Here, P_S is the point set described above.*

Towards the proof of Lemma 15, we first consider our intermediate problem.

⁷ We note that \mathcal{S} may not be a solution in the sense that it may not satisfy all constraints (but it is close to some solution that satisfies all of them).

► **Lemma 19.** *RATIONAL-DR is solvable in time $n^{\mathcal{O}(n)}$.*

Proof. Our algorithm first computes the set of polynomial constraints $\text{ConstSqu}(G)$ in time $\mathcal{O}(n^2)$. Then, it computes a $1/2$ accurate approximate solution for $\text{ConstSqu}(G)$ by using the approach of Arora et al. [4] in time $n^{\mathcal{O}(n)}$. In Lemma 18, we have shown that such an approximate solution is an exact solution. This concludes the proof. ◀

Finally, we are ready to prove Lemma 15, and thus conclude the correctness of Theorem 14.

Proof of Lemma 15. We use the algorithm given by Lemma 19 to output a point set $P \subseteq \mathbb{Q}^2$ in time $n^{\mathcal{O}(n)}$ such that G is isomorphic to $\mathcal{DT}(P)$ and the points corresponding to vertices in $V(f^*)$ lie on the outer face of $\mathcal{DT}(P)$ in the order in which they appear in the cycle of f^* . We denote by $\bar{P}_i = (\bar{X}_i, \bar{Y}_i)$ the value P assigns to the vertex $v_i \in V(G)$. For $i \in [n]$, since $\bar{X}_i, \bar{Y}_i \in \mathbb{Q}$, we let the representation be $\bar{X}_i = \bar{X}_i^a / \bar{X}_i^b$ and $\bar{Y}_i = \bar{Y}_i^a / \bar{Y}_i^b$, where $\bar{X}_i^a, \bar{X}_i^b, \bar{Y}_i^a, \bar{Y}_i^b \in \mathbb{Z}$. For each edge $(\bar{P}_i, \bar{P}_j) \in E(\mathcal{DT}(P))$, there exists a disc C_{ij} with a centre, say \bar{P}_{ij} , containing only \bar{P}_i and \bar{P}_j from P . These assignments satisfy the constraints $\text{Con}(G)$ and $\text{Dis}(G)$ presented in Section 3. It thus follows that \mathcal{P} satisfies $\text{Const}(G)$. From Observation 11 it follows that for any $\alpha \in \mathbb{R}^+$, we have that $\alpha\mathcal{P}$ satisfies $\text{Const}(G)$. We let $\beta = \prod_{i \in [n]} \bar{X}_i^b \bar{Y}_i^b$. But then $\beta\mathcal{P}$ satisfies $\text{Const}(G)$, and hence $\beta P = \{(\beta\bar{X}_i, \beta\bar{Y}_i \mid i \in [n])\}$ is a point set such that G is isomorphic to $\mathcal{DT}(\beta P)$ where the points corresponding to vertices in $V(f^*)$ lie on the outer face of $\mathcal{DT}(\beta P)$. Therefore, we output a correct point set, βP , with only integer coordinates. This concludes the proof. ◀

6 Conclusion

In this paper, we gave an $n^{\mathcal{O}(n)}$ -time algorithm for the DELAUNAY REALIZATION problem. We have thus obtained the first exact exponential-time algorithm for this problem. Still, the existence of a practical (faster) exact algorithm for DELAUNAY REALIZATION is left for further research. In this context, it is not even clear whether a significantly faster algorithm, say a polynomial-time algorithm, exists. Perhaps one of the first questions to ask in this regard is whether there exist instances of graphs that are realizable but for which the integers in any integral solution need to be exponential in the input size? If yes, does even the representation of these integers need to be exponential in the input size?

References

- 1 Karim A. Adiprasito, Arnau Padrol, and Louis Theran. Universality theorems for inscribed polytopes and delaunay triangulations. *Discrete & Computational Geometry*, 54(2):412–431, 2015.
- 2 Akanksha Agrawal, Saket Saurabh, and Meirav Zehavi. A finite algorithm for the realizability of a delaunay triangulation. *arXiv*, 2022. doi:10.48550/ARXIV.2210.03932.
- 3 Md. Ashraful Alam, Igor Rivin, and Ileana Streinu. Outerplanar graphs and Delaunay triangulations. In *Proceedings of the 23rd Annual Canadian Conference on Computational (CCCG)*, 2011.
- 4 Sanjeev Arora, Rong Ge, Ravindran Kannan, and Ankur Moitra. Computing a nonnegative matrix factorization – provably. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing*, STOC, pages 145–162, 2012.
- 5 M. Artin. *Algebra*. Pearson Prentice Hall, 2011.
- 6 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

- 7 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, 3rd ed. edition, 2008.
- 8 Norishige Chiba, Kazunori Onoguchi, and Takao Nishizeki. Drawing plane graphs nicely. *Acta Inf.*, 22(2):187–201, 1985.
- 9 K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Computational Geometry*, 4:387–421, 1989.
- 10 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.
- 11 Hubert de Fraysseix, János Pach, and Richard Pollack. Small sets supporting fary embeddings of planar graphs. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 426–433, 1988.
- 12 Giuseppe Di Battista and Luca Vismara. Angles of planar triangular graphs. *SIAM Journal on Discrete Mathematics*, 9(3):349–359, 1996.
- 13 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 14 M. Dillencourt. Toughness and Delaunay triangulations. In *Proceedings of the Third Annual Symposium on Computational Geometry*, SoCG, pages 186–194, 1987.
- 15 Michael B. Dillencourt. Realizability of Delaunay triangulations. *Information Processing Letters*, 33:283–287, 1990.
- 16 Michael B. Dillencourt and Warren D. Smith. Graph-theoretical conditions for inscribability and Delaunay realizability. *Discrete Mathematics*, 161(1-3):63–77, 1996.
- 17 D. Yu. Grigor’ev and N. N. Vorobjov, Jr. Solving systems of polynomial inequalities in subexponential time. *Journal of Symbolic Computation*, 5:37–64, 1988.
- 18 Leonidas J. Guibas, Donald E. Knuth, and Micha Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7(1):381–413, 1992.
- 19 Tetsuya Hiroshima, Yuichiro Miyamoto, and Kokichi Sugihara. Another proof of polynomial-time recognizability of Delaunay graphs. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 83:627–638, 2000.
- 20 Craig D Hodgson, Igor Rivin, and Warren D Smith. A characterization of convex hyperbolic polyhedra and of convex polyhedra inscribed in the sphere. *Bulletin of the American Mathematical Society*, 27:246–251, 1992.
- 21 Jan Kratochvíl and Jivri Matoušek. Intersection graphs of segments. *J. Comb. Theory, Ser. B*, 62(2):289–315, 1994.
- 22 Timothy Lambert. An optimal algorithm for realizing a Delaunay triangulation. *Information Processing Letters*, 62(5):245–250, 1997.
- 23 Colin McDiarmid and Tobias Müller. Integer realizations of disk and segment graphs. *Journal of Combinatorial Theory, Series B*, 103(1):114–143, 2013.
- 24 Tobias Müller, Erik Jan van Leeuwen, and Jan van Leeuwen. Integer representations of convex polygon intersection graphs. *SIAM J. Discrete Math.*, 27(1):205–231, 2013.
- 25 Takao Nishizeki, Kazuyuki Miura, and Md. Saidur Rahman. Algorithms for drawing plane graphs. *IEICE Transactions*, 87-D(2):281–289, 2004.
- 26 Yasuaki Oishi and Kokichi Sugihara. Topology-oriented divide-and-conquer algorithm for Voronoi diagrams. *Graphical Models and Image Processing*, 57:303–314, 1995.
- 27 Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Inc., 1992.
- 28 János Pach and Pankaj K. Agarwal. *Combinatorial Geometry*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, New York, 1995.
- 29 James Renegar. On the computational complexity and geometry of the first-order theory of the reals. *Journal of Symbolic Computation*, 13:255–352, 1992.
- 30 Igor Rivin. Euclidean structures on simplicial surfaces and hyperbolic volume. *Annals of Mathematics*, 139:553–580, 1994.

1:16 A Finite Algorithm for the Realizability of a Delaunay Triangulation

- 31 Kokichi Sugihara. Simpler proof of a realizability theorem on Delaunay triangulations. *Information Processing Letters*, 50:173–176, 1994.
- 32 Kokichi Sugihara and Masao Iri. Construction of the Voronoi diagram for one million generators in single-precision arithmetic. *Proceedings of the IEEE*, 80:1471–1484, 1992.
- 33 Kokichi Sugihara and Masao Iri. A robust topology-oriented incremental algorithm for Voronoi diagrams. *International Journal of Computational Geometry & Applications*, 4(02):179–228, 1994.
- 34 William Thomas Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 3(1):743–767, 1963.
- 35 Hassler Whitney. *Congruent Graphs and the Connectivity of Graphs*, pages 61–79. Birkhäuser Boston, Boston, MA, 1992.

Parameterized Complexity of Perfectly Matched Sets

Akanksha Agrawal ✉ 

Indian Institute of Technology Madras, Chennai, India

Sutanay Bhattacharjee ✉

Indian Institute of Technology Madras, Chennai, India

Satyabrata Jana ✉ 

The Institute of Mathematical Sciences, HBNI, Chennai, India

Abhishek Sahu ✉

Indian Institute of Technology Madras, Chennai, India

Abstract

For an undirected graph G , a pair of vertex disjoint subsets (A, B) is a *pair of perfectly matched sets* if each vertex in A (resp. B) has exactly one neighbor in B (resp. A). In the above, the size of the pair is $|A|$ ($= |B|$). Given a graph G and a positive integer k , the PERFECTLY MATCHED SETS problem asks whether there exists a pair of perfectly matched sets of size at least k in G . This problem is known to be NP-hard on planar graphs and W[1]-hard on general graphs, when parameterized by k . However, little is known about the parameterized complexity of the problem in restricted graph classes. In this work, we study the problem parameterized by k , and design FPT algorithms for: i) apex-minor-free graphs running in time $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$, and ii) $K_{b,b}$ -free graphs. We obtain a linear kernel for planar graphs and $k^{\mathcal{O}(d)}$ -sized kernel for d -degenerate graphs. It is known that the problem is W[1]-hard on chordal graphs, in fact on split graphs, parameterized by k . We complement this hardness result by designing a polynomial-time algorithm for interval graphs.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Perfectly Matched Sets, Parameterized Complexity, Apex-minor-free graphs, d -degenerate graphs, Planar graphs, Interval Graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.2

Funding Akanksha Agrawal: Supported by New Faculty Initiation Grant no. NFIG008972.

1 Introduction

MATCHING is one of the very classical polynomial-time solvable problems in Computer Science with varied applications. Finding a matching with additional structure, such as an induced matching has been well studied both in classical complexity as well as parameterized complexity, see, for instance, [4, 9, 18, 20, 24, 24, 27, 28] (list is only illustrative, and not comprehensive). In this article, we are interested in a matching that is slightly weaker than the structure of an induced matching but still more structured than a matching.

For a graph G , a pair of vertex disjoint subsets, (A, B) is a *pair of perfectly matched sets* in G if each vertex in A has exactly one neighbor in B and each vertex in B has exactly one neighbor in A ; the *size* of the pair is $|A|$ ($= |B|$). Note that there can be edges between vertices of A (resp. B), which is forbidden in the case of induced matching. We study the problem called PERFECTLY MATCHED SETS, which is defined below.

PERFECTLY MATCHED SETS

Parameter: k

Input: An undirected graph G and an integer k .

Question: Does there exist a pair of perfectly matched sets of size at least k in G ?



© Akanksha Agrawal, Sutanay Bhattacharjee, Satyabrata Jana, and Abhishek Sahu; licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 2; pp. 2:1–2:13

Leibniz International Proceedings in Informatics



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

This problem was first introduced in [27] where it was named as *Maximum TR-matching problem* (Transmitter- Receiver problem). The paper showed that this problem is NP-complete when restricted to graphs having degree 3. Evan, Goldreich, and Tong in [13] showed that TR-matching is NP-complete on bipartite graphs. This problem was revisited by Aravind and Saxena in 2021, [1] where they called the problem as PERFECTLY MATCHED SETS. They designed FPT algorithms for this problem parameterized by the structural parameters such as distance to cluster, distance to co-cluster, and treewidth. They also prove that the problem is NP-hard on planar graphs and W[1]-hard parameterized by the solution size k , when restricted to bipartite graphs and split graphs.

The PERFECTLY MATCHED SETS problem is also closely related to the problem PERFECT MATCHING CUT where we want edge cuts of size k , such that the vertices participating in these edges induce a matching and a perfect matching, respectively. We remark that in PERFECTLY MATCHED SETS, we do not insist that the edges between the pair of perfectly matched sets (A, B) is a cut in the graph. The MATCHING CUT and PERFECT MATCHING CUT problems have been investigated in the literature even when restricted to well-studied graph classes, see, for instance, [2, 6, 7, 21, 22, 25].

Our Results. In this paper, we investigate the parameterized complexity of the PERFECTLY MATCHED SETS problem when the input graph is from a structured graph family, for several choices of well-studied graph families. The starting point of our work is the result by Aravind and Saxena [1]. The paper showed that the problem is W[1]-hard even on split graphs, which is an important subclass of chordal graphs. Inspired by this negative result, we turn to interval graphs, which is arguably the most well-studied subclass of chordal graphs. We obtain the following result by using a dynamic programming based algorithm.

► **Theorem 1.** PERFECTLY MATCHED SETS on interval graphs admits an algorithm running in time $\mathcal{O}(n^5)$.

Aravind and Saxena [1] showed that PERFECTLY MATCHED SETS is NP-complete even when the input graph is planar. Inspired by this we design an FPT algorithm for a strictly more general class of apex-minor-free graphs. A graph H is an *apex* graph if there is $v \in V(H)$, such that $H - \{v\}$ is planar. Consider any finite set \mathcal{H} of graphs that contains at least one apex graph, and let $\mathcal{F}_{\mathcal{H}}$ be the family of graphs that do not contain any graph from \mathcal{H} as a minor. The \mathcal{H} -MINOR FREE PMS problem is the PERFECTLY MATCHED SETS problem with an additional guarantee that the input graph belongs to $\mathcal{F}_{\mathcal{H}}$. Note that for $\mathcal{H} = \{K_5, K_{3,3}\}$, $\mathcal{F}_{\mathcal{H}}$ is the family of planar graphs. We obtain the following result:

► **Theorem 2.** For any (fixed) finite set \mathcal{H} of graphs that contains at least one apex graph, \mathcal{H} -MINOR FREE PMS has an FPT algorithm running in time $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$.

We remark that the same approach used in obtaining the above result can be used to obtain an FPT algorithm on bounded genus graphs, due to bidimensionality [10]. We remark that having a pair of perfectly matched sets of size at least k is expressible in MSO (actually, even in FO). So, there is an FPT algorithm on the much more general nowhere dense classes (admittedly with a worse running time)[16].

For $b \in \mathbb{N}$, a graph is $K_{b,b}$ -free if it does not contain a bi-clique with b vertices on each side as a subgraph. We obtain the following result by using an approach similar to random separation [3], in combination with a result of Dabrowski et al. [9].

► **Theorem 3.** For any fixed $b \in \mathbb{N}$, PERFECTLY MATCHED SETS on $K_{b,b}$ -free graphs admits an FPT algorithm, when parameterized by k .

Kanj et al. [18] and Erman et al. [20] independently designed $\mathcal{O}(k^c)$ kernels for the INDUCED MATCHING problem for graphs of arboricity bounded by c . The authors [18] also showed that any twinless graph of average degree d and bounded chromatic number contains an induced matching of size $\Omega(n^{1/d})$. The core of their proof is the *system of strong representatives* of a set family. This combinatorial tool also forms the backbone of our following result.

► **Theorem 4.** PERFECTLY MATCHED SETS admits a $k^{\mathcal{O}(d)}$ -sized kernel on d -degenerate graphs.

As planar graphs are 5-degenerate, the theorem above directly gives us a polynomial kernel for PERFECTLY MATCHED SETS on these graphs. Following an approach by Kanj et al. [18] for obtaining a linear kernel for INDUCED MATCHING on planar graphs, we obtain a linear kernel (improving upon the already obtained polynomial kernel) for PERFECTLY MATCHED SETS on this graph class.

2 Preliminaries

Sets and graph notations. We use $\mathbb{N} = \{1, 2, \dots\}$ to denote the set of natural numbers. We use $[k]$ as a shorthand for $\{1, 2, \dots, k\}$ and use $[k]_0$ for $[k] \cup \{0\}$, where $k \in \mathbb{N}$. In this article, we only consider simple undirected graphs. Given a graph G , we denote the vertex set and edge set of G by $V(G)$ and $E(G)$ respectively. Unless specified, n and m denote the number of vertices and edges of the graph G . Two vertices u, v are said to be *adjacent* if there is an edge (denoted by $\{u, v\}$) between u and v in G . For $X \subseteq V(G)$, $G[X]$ denotes the induced subgraph of G with vertex set X and edge set $\{\{u, v\} \mid u, v \in X \text{ and } \{u, v\} \in E(G)\}$, $G - X$ denotes the subgraph $G[V(G) \setminus X]$. For an edge set $E' \subseteq E$, $V(E')$ denotes the set of all the vertices of G having at least one edge in E' incident on it. $E(A, B)$ denotes the set of edges with one endpoint in A and the other in B . The open neighborhood of a vertex v , denoted by $N_G(v)$, is the set of vertices adjacent to v . The *closed neighborhood* of v is defined as $N_G[v] = N_G(v) \cup \{v\}$. The subscript in the notation for neighborhood is omitted if the graph under consideration is clear. For $X \subseteq V(G)$, $N[X]$ denotes the set of vertices $\bigcup_{v \in X} N[v]$. Two distinct vertices u, v is said to be a pair of *false twins* if $N_G(u) = N_G(v)$ and *true twins* if $N_G[u] = N_G[v]$. A *clique* in graph G is a set of vertices such that there is an edge between every pair of vertices in the set. An *independent set* in the graph G is a set of vertices such that there is no edge between any pair of vertices in the set. $K_{n,m}$ is the complete bipartite graph, also known as a biclique, with partitions of size n and m . A k -biclique is a $2k$ -vertex complete bipartite graph. A subset $D \subseteq V(G)$ is said to be a dominating set of G if $N[D] = V(G)$. A *vertex cover* of a graph is a set of vertices that includes at least one endpoint of every edge of the graph. The cardinality of the smallest size dominating set is called as *domination number* of G . D is said to be a *2-dominating set* if $N[N[D]] = V(G)$. $G \setminus e$ denotes the graph obtained by contracting the edge e in G . The contraction of an edge $\{u, v\}$ in the graph involves the deletion of vertices u and v from G and the addition of a new vertex w , which is adjacent to all the vertices of $N(u) \cup N(v)$. For two graphs G_1 and G_2 , we denote $G_1 \subseteq G_2$ if G_1 is an induced subgraph of G_2 .

Graph classes. A graph is *planar* if it can be drawn in the plane without edge intersections except at the endpoint). A graph G is a d -degenerate graph if every induced subgraph of G contains a vertex of degree at most d . A $K_{b,b}$ -free graph is a graph that does not contain biclique $K_{b,b}$ as a subgraph (not necessarily induced). An *apex graph* is a graph that can

be made planar by removing one of its vertices. Apex-minor-free graphs are basically those graphs that exclude a fixed apex graph as a minor. More precisely, \mathcal{C} is apex-minor-free graph class if there exists some apex graph H such that no graph from \mathcal{C} admits H as a minor. An *interval graph* is an undirected graph formed from a set of intervals on the real line, with a vertex for each interval and an edge between vertices whose intervals intersect. It is the intersection graph of the intervals. [5]. For standard graph definition and notations, we refer to the graph theory book by R. Diestel [11]. For parameterized complexity terminology, we refer to the parameterized algorithms book by Cygan et al. [8].

Treewidth. A *tree decomposition* of a graph $G = (V, E)$ is a pair (T, X) where T is a tree on vertex set $V(T)$. The vertices of $V(T)$ are called *nodes*. Also, $X = (\{X_i \mid i \in V(T)\})$ is a collection of subsets of V such that -

1. Every vertex of G is contained in at least one bag. $\cup_{i \in V(T)} X_i = V$,
2. For every edge $\{u, v\} \in E$, there exists a node $i \in V(T)$ such that bag X_i contains both u and v .
3. For each $u \in V$, the set of nodes whose bags contain u , $T_u = \{i \in V(T) : i \in X_i\}$ forms a connected subtree of T .

The *width* of a tree decomposition $(T, (\{X_i \mid i \in V(T)\})$ is equal to the maximum size of its bag minus 1, $\max_{i \in V(T)} \{|X_i| - 1\}$. The *treewidth* of a graph G , $\text{tw}(G)$ is the minimum width of a tree decomposition over all tree decompositions of G .

Perfectly matched sets. A *matching* in a graph G is a set of edges M such that no two edges in M share the same endpoint. A matching M is *maximal* if $G - V(M)$ is edge less. A matching M is said to be an *induced matching* if the subgraph induced by the vertices in M contains only the edges of M . If M is maximal then $V(M)$ is a vertex cover of G , and it is easy to verify that $\text{tw}(G) \leq |V(M)|$. For a pair (A, B) of disjoint subsets of vertices of $V(G)$, we say (A, B) is a *pair of perfectly matched sets* if every vertex in A (resp. B) has exactly one neighbor in B (resp. A). The size of the pair is $|A| = |B|$.

Parameterized problems and kernels. A parameterized problem Π is a subset of $\Gamma^* \times \mathbb{N}$ for some finite alphabet Γ . An instance of a parameterized problem consists of (X, k) , where k is called the parameter. The notion of kernelization is formally defined as follows. A kernelization algorithm, or in short, a kernelization, for a parameterized problem $\Pi \subseteq \Gamma^* \times \mathbb{N}$ is an algorithm that, given $(X, k) \in \Gamma^* \times \mathbb{N}$, outputs in time polynomial in $|X| + k$ a pair $(X', k') \in \Gamma^* \times \mathbb{N}$ such that (a) $(X, k) \in \Pi$ if and only if $(X', k') \in \Pi$ and (b) $|X'|, |k'| \leq g(k)$, where g is some computable function depending only on k . The output of kernelization (X', k') is referred to as the kernel and the function g is referred to as the size of the kernel. If $g(k) \in k^{\mathcal{O}(1)}$, then we say that Π admits a polynomial kernel. We refer to the monographs [12, 14, 26] for a detailed study of the area of kernelization.

3 Polynomial-time Algorithm for Interval Graphs

Recall that PERFECTLY MATCHED SETS is W[1]-hard when parameterized by the solution size k even when restricted to split graphs (and thus, chordal graphs). Interval graphs belong to the class of chordal graphs. In this section, we present a polynomial-time dynamic programming algorithm that computes a maximum-sized pair of perfectly matched sets for any given interval graph.

Let G be an interval graph with vertex set $V(G) = \{v_1, v_2, \dots, v_n\}$. Since G is an interval graph, there exists a corresponding geometric intersection representation of G , where each vertex $v_i \in V(G)$ is associated with an interval $I_i = [\ell(I_i), r(I_i)]$ in the real line, where $\ell(I_i)$ and $r(I_i)$ denote left and right endpoints, respectively in I_i . Two vertices v_i and v_j are adjacent in G if and only if their corresponding intervals I_i and I_j intersect with each other. We can also assume that along with the graph, we are also given the corresponding underlying intervals on the real line, as there are well-known linear-time algorithms that compute such a representation [19]. We use \mathcal{I} to denote the set $\{I_i : v_i \in V\}$ of intervals and P to denote the set of all endpoints of these intervals, i.e., $P = \cup_{I \in \mathcal{I}} \{\ell(I), r(I)\}$. In the remaining section, we will use v_i and I_i interchangeably. Note that we can assume that the endpoints of all the intervals in the interval representation are distinct – otherwise, we can slightly perturb the endpoints of the intervals to obtain a new interval representation of the graph in which this is true.

► **Proposition 5.** *Let G be a connected interval graph. There exists an ordering, $<$, of $V(G)$ such that for $u, v, w \in V(G)$ if $u < v < w$ and $\{u, w\} \in E(G)$ then $\{v, w\} \in E(G)$.*

We remark that such an ordering in Proposition 5 can be obtained based on the right endpoints of intervals, more specifically the set $\{r(I_i)\}$ and the ordering is as follows: for any two vertices v_i and v_j , we have $v_i < v_j$ if and only if $r(I_i) < r(I_j)$. We call such an ordering, the *right-end ordering* of $V(G)$.

► **Lemma 6.** *Let G be an interval graph with a right-end ordering, $<$, of $V(G)$. Consider any distinct pair of edges $\{u, v\}$ and $\{u', v'\}$ in a pair of perfectly matched sets (A, B) where $u < v$ and $u' < v'$. If $u < u'$, then $v < v'$.*

Proof. Towards a contradiction suppose there are edges $\{u, v\}, \{u', v'\}$ in the pair of perfectly matched sets (A, B) , where $u < v, u' < v', u < u'$ and $u' < v$. Then, either $u < u' < v' < v$, or $u < u' < v < v'$. In either of these cases, by Proposition 5, v is adjacent to both u' and v' which is a contradiction to the fact that (A, B) is perfectly matched sets in G . ◀

Lemma 6 directly implies the following remark.

► **Remark 7.** Let $\{\{u_i, v_i\} : 1 \leq i \leq k\}$ be a set of k edges in a pair (A, B) of perfectly matched sets in G with $u_1 < u_2 < \dots < u_k$ and $u_i < v_i$, for each $i \in [k]$. Then, $u_1 < v_1 < u_2 < v_2 < \dots < u_k < v_k$.

Algorithm and its Correctness. We define a table for our dynamic-programming algorithm. Let $v_1 < v_2 < \dots < v_n$ be the right-end ordering of the vertex set of G . For every tuple (v_i, v_j, t) , where $\{v_i, v_j\} \in E(G)$, $i, j \in [n]$, $i < j$ and $t \in [\lfloor n/2 \rfloor]$, we define two Boolean values: (i) $\text{PM}[(v_i, A), (v_j, B); t]$ and (ii) $\text{PM}[(v_i, B), (v_j, A); t]$.¹ The entry $\text{PM}[(v_i, A), (v_j, B); t]$ is **true** if there exists a pair (A, B) of perfectly matched sets of size t such that $v_i \in A$, $v_j \in B$ and for all the vertices $v \in (A \cup B) \setminus \{v_i, v_j\}$, we have $v < v_i$. Similarly the entry $\text{PM}[(v_i, B), (v_j, A); t]$ is **true** if there exists a pair (A, B) of perfectly matched sets of size t such that $v_i \in B$, $v_j \in A$ and for all the vertices $u \in (A \cup B) \setminus \{v_i, v_j\}$, we have $u < v_i$.

In the base case, both $\text{PM}[(v_i, A), (v_j, B); 1]$ and $\text{PM}[(v_i, B), (v_j, A); 1]$ are **true** for every possible pair v_i and v_j (note because of the way the entry is defined, $\{v_i, v_j\}$ must be an edge in G). We will use the convention that *empty* OR is 0. In the lemma below, we give a recursive formula for computing the values $\text{PM}[(v_i, A), (v_j, B); t]$ for $t > 1$.

¹ A and B in these entries are just symbols, added for extra clarity.

► **Lemma 8.** *For every integer $t \in \llbracket n/2 \rrbracket \setminus \{1\}$, and every pair of adjacent vertices v_i, v_j in G where $i < j$, the following recurrence holds:*

$$\text{PM}[(v_i, A), (v_j, B); t] = \bigvee_{\substack{\{x, y\} \in E(G) \\ x < y < v_i}} \left(\left(\text{PM}[(x, A), (y, B); t-1] \wedge [\{x, v_j\} \notin E(G)] \wedge [\{y, v_i\} \notin E(G)] \right) \vee \left(\text{PM}[(x, B), (y, A); t-1] \wedge [\{x, v_i\} \notin E(G)] \wedge [\{y, v_j\} \notin E(G)] \right) \right)$$

Proof. In the forward direction let us assume that $\text{PM}[(v_i, A), (v_j, B); t] = \mathbf{true}$. So according to the definition of our dynamic-programming table, $\{v_i, v_j\} \in E(G)$ and there exists a pair (A, B) of perfectly matched sets of size t such that $v_i \in A$, $v_j \in B$ and for all the vertices $v \in (A \cup B) \setminus \{v_i, v_j\}$, we have $v < v_i$. Now consider the pair $(A' = A \setminus \{v_i\}, B' = B \setminus \{v_j\})$. It is easy to see that this pair is a perfectly matched sets of size $t-1$ and all the vertices v in the pair having the property that $v < v_i$. Consider the last vertex in the right-end ordering of $V(G)$ which occurs in the vertex set $A' \cup B'$. Let this vertex be y and x be its (only) neighbour in B' . Note that $x < y$ and for any vertex $v \in (A' \cup B') \setminus \{x, y\}$, it must hold that $v < x$ (see Remark 7). If $y \in B'$, then clearly, $\{x, v_j\} \notin E(G)$, $\{y, v_i\} \notin E(G)$, and $\text{PM}[(x, A), (y, B); (t-1)] = \mathbf{true}$. Otherwise, $y \in A'$, and then $\{x, v_i\} \notin E(G)$, $\{y, v_j\} \notin E(G)$, and $\text{PM}[(x, B), (y, A); (t-1)] = \mathbf{true}$.

In the reverse direction, assume that there exists a pair of vertices $x < y$, $\{x, y\} \in E(G)$ such that $\text{PM}[(x, A), (y, B); (t-1)] = \mathbf{true}$ and $\{x, v_j\} \notin E(G)$, $\{y, v_i\} \notin E(G)$. (The case when $\text{PM}[(x, B), (y, A); (t-1)] = \mathbf{true}$ and $\{x, v_i\} \notin E(G)$, $\{y, v_j\} \notin E(G)$ can be argued symmetrically.) The above means that there is a pair of perfectly matched sets (A', B') with $t-1$ edges such that: $\{x, y\} \in E(G)$, $x \in A'$, $y \in B'$, and for each $v \in (A' \cup B') \setminus \{x, y\}$, we have $v < x$. Let $A = A' \cup \{v_i\}$ and $B = B' \cup \{v_j\}$. Note that we have $x < y < v_i < v_j$, and thus, for each $v \in A' \cup B'$, we have $v < v_i < v_j$. For a contradiction suppose that we have $v \in B'$, such that $\{v, v_i\} \in E(G)$. Note that $v < x < y < v_i$, as $\{y, v_j\} \notin E(G)$ (see Remark 7). But then from Lemma 6, we can obtain that $\{v, x\} \in E(G)$, which contradicts that (A', B') is a pair of perfectly matched sets. Similarly, towards a contradiction suppose that we have $v \in A'$, such that $\{v, v_j\} \in E(G)$. Then, $v < x < y < v_j$, and thus, $\{y, v\} \in E(G)$, which is a contradiction. From the above discussions, we can conclude that $\text{PM}[(v_i, A), (v_j, B); t] = \mathbf{true}$. ◀

Similarly, we have a recursive formula for computing the values $\text{PM}[(v_i, B), (v_j, A); t]$ for $t > 1$. The correctness proof is similar to that of Lemma 8.

► **Lemma 9.** *For every integer $t \in \llbracket n/2 \rrbracket \setminus \{1\}$, and every pair of adjacent vertices v_i, v_j in G where $i < j$, the following recurrence holds:*

$$\text{PM}[(v_i, B), (v_j, A); t] = \bigvee_{\substack{\{x, y\} \in E(G) \\ x < y < v_i}} \left(\left(\text{PM}[(x, A), (y, B); t-1] \wedge [\{y, v_j\} \notin E(G)] \wedge [\{x, v_i\} \notin E(G)] \right) \vee \left(\text{PM}[(x, B), (y, A); t-1] \wedge [\{x, v_j\} \notin E(G)] \wedge [\{y, v_i\} \notin E(G)] \right) \right)$$

We can compute all the entries of our dynamic programming table using the recurrence relations given by Lemma 8 and Lemma 9.

Time Complexity. For a pair of adjacent vertices v_i, v_j , where $i < j$, the time required to compute $\text{PM}[(v_i, A), (v_j, B); t]$ and $\text{PM}[(v_i, B), (v_j, A); t]$, once we have computed the entries till the values at most $t-1$, is bounded by $\mathcal{O}(n^2)$. As $t < n$, the number of entries we have to compute is bounded by $\mathcal{O}(n^3)$, thus bounding the total running time of our algorithm by $\mathcal{O}(n^5)$. This proves Theorem 1.

4 FPT Algorithm for Apex-Minor-Free Graphs

Consider any (fixed) finite set \mathcal{H} of graphs that contains at least one apex graph; we will work with this fixed family throughout this section. Recall that $\mathcal{F}_{\mathcal{H}}$ is the family of graphs that do not contain any graph from \mathcal{H} as a minor, and the \mathcal{H} -MINOR FREE PMS problem is the same as the PERFECTLY MATCHED SETS problem with an additional guarantee that the input graph belongs to $\mathcal{F}_{\mathcal{H}}$. In this section, we prove Theorem 2 by designing a simple FPT algorithm with the desired running time. Let (G, k) be an instance of \mathcal{H} -MINOR FREE PMS. Our algorithm will begin by greedily trying to construct a solution, if we succeed then the algorithm halts. Otherwise, we will be able to bound the size of a 2-dominating in G by $\mathcal{O}(k)$. This together with a result of Fomin [15]) will imply that the treewidth of G is bounded by $\mathcal{O}(\sqrt{k})$. Now we can use the algorithm of Aravind and Saxena [1] for PERFECTLY MATCHED SETS parameterized by treewidth to obtain the proof of the theorem. We begin by stating the two useful results.

► **Proposition 10** (Lemma 2, [15]). *For an \mathcal{H} -minor free graph G , if ℓ is the size of a minimum 2-dominating set of G , then the treewidth of G is bounded by $c_{\mathcal{H}} \cdot \sqrt{\ell}$, where $c_{\mathcal{H}}$ is a constant depending on \mathcal{H} .*

► **Proposition 11** (Theorem 7, [1]). *There exists an algorithm that calculate maximum perfectly matched sets for an n vertex graph with treewidth at most w in time $\mathcal{O}(12^w \cdot \text{poly}(n))$.*

The next lemma gives the procedure that either resolves the instance or obtains a small 2-dominating set in G .

► **Lemma 12.** *There is a polynomial time algorithm that either correctly concludes that (G, k) is a yes-instance of \mathcal{H} -MINOR FREE PMS, or outputs a 2-dominating set Q of G where $|Q| \leq 2 \cdot (k - 1)$.*

Proof. Let (G, k) be an instance of the problem. If G has an isolated vertex, then such a vertex is not part of any perfectly matched set, and thus we remove it. We will next create a sequence of perfectly matched sets $S_0 \subset S_1 \subset \dots \subset S_q$ and graphs $G_0 \supseteq G_1 \supseteq \dots \supseteq G_q$, which, intuitively speaking, will be constructed by greedily adding an edge (one at a time) to form a perfectly matched set.

Initialize $S_0 = \emptyset$ and $G_0 = G$. Iteratively do the following: if there is an edge $e_i = \{u_i, v_i\} \in E(G_i)$, then set $S_{i+1} = S_i \cup \{e\}$ and $G_{i+1} = G_i - (N_G[u] \cup N_G[v])$. The q be an integer where the above procedure stops, which is the case when G_q has no edges. Notice that for any $i \in [q]_0$, each $S \in \{S_j \setminus S_i \mid j \in \{i+1, i+2, \dots, q\}\}$ is a pair of perfectly matched sets in G_i . The above in particular implies that S_q is a pair of perfectly matched sets in $G = G_0$. Also, for each $i \in [q]_0$, $|S_i| = i$. If $q \geq k$, then we have obtained a pair of perfectly matched sets in G of size at least k , and thus we can conclude that the instance is a yes-instance. Otherwise $q \leq k - 1$, and we let $Q = \{u_i, v_i \mid i \in [q]\}$. Consider any vertex $u \in V(G) \setminus N_G[Q]$. Since G has no isolated vertices, u must have a neighbor v in G . Note that $v \notin Q$, as $u \in V(G) \setminus N_G[Q]$. Also, if $v \notin N_G(Q)$, then $\{u, v\}$ is an edge in G_q , which contradicts that G_q has no edges. The above discussions imply that Q is a 2-dominating set in G of size at most $|Q| \leq 2 \cdot (k - 1)$. ◀

We are now ready to prove Theorem 2.

Proof of Theorem 2. Consider an instance (G, k) of \mathcal{H} -MINOR FREE PMS. If Lemma 12 returns that the instance is a yes-instance, then we are done. Otherwise, it returns a 2-dominating set in G of size at most $2 \cdot (k - 1)$. From Proposition 10, the treewidth of G is

bounded by $c_{\mathcal{H}} \cdot \sqrt{2 \cdot (k-1)}$, where $c_{\mathcal{H}}$ is a constant depending on the family \mathcal{H} . Now using Lemma 7.4 of [8], we compute a nice tree decomposition of width at most $c_{\mathcal{H}} \cdot \sqrt{2 \cdot (k-1)}$ in time bounded by $\mathcal{O}(nk)$. Now we can use Proposition 11 to resolve the instance. ◀

5 FPT Algorithm for $K_{b,b}$ -free Graphs

The goal of this section is to prove Theorem 3. Consider any fixed number $b \in \mathbb{N}$. Recall that a graph is $K_{b,b}$ -free if it does not contain a *subgraph* isomorphic to $K_{b,b}$. We obtain an FPT algorithm for PERFECTLY MATCHED SETS on $K_{b,b}$ -free graphs by using an approach similar to random separation [3], in combination with the below-stated result of Dabrowski et al. [9].

► **Proposition 13** (Lemma 2, [9]). *For any natural numbers s, t and p , there is a number $N'(s, t, p)$ such that every graph with a matching of size at least $N'(s, t, p)$ contains either a clique K_s , an induced bi-clique $K_{t,t}$ or an induced matching of size p . Here, $N'(s, t, p) = R(s, R(s, N(t, p)))$ where $R(s, t)$ is the non-symmetric Ramsey number.*

Let (G, k) be an instance of PERFECTLY MATCHED SETS, where G is a $K_{b,b}$ -free graph with n vertices. We color the vertices of $V(G)$ independently and randomly using two colors, *red* and *blue* (with equal probability). This forms a random partition $V_R \uplus V_B$ of the vertices of G , where V_R and V_B are the set of vertices colored with red and blue color, respectively. We call these two partitions as *color classes*. Next, we obtain the graph G' from G by removing all the edges between the vertices of the same color class. Thus, the edges in G' have endpoints of differing colors, and thus it is bipartite. We compute (in polynomial time) a maximum sized matching M in G' [23]. We will next argue that either M has at most $N'(3, b, k)$ edges, or we can conclude that (G, k) is a yes-instance.

Case 1. Firstly suppose that M has at least $N'(3, b, k)$ edges. Recall that G is bipartite, so it does not have any K_3 . Moreover, as G is $K_{b,b}$ -free, we can obtain that G' has no induced $K_{b,b}$. As the size of a maximum matching in G' is at least $N'(3, b, k)$, using Proposition 13 we can obtain that G' has an induced matching M_I of size at least k . Now using the next observation we can conclude that (G, k) is a yes-instance of the problem.

► **Observation 14.** $(V_R \cap V(M_I), V_B \cap V(M_I))$ is a pair of perfectly matched sets in G of size at least k .

Proof. Consider $x \in V_R \cap V(M_I)$, where x has a neighbor $y \in V_B \cap V(M_I)$ and $\{x, y\}$ is an edge in M_I . Let $z \neq y$ be another neighbor of x in $V_B \cap V(M_I)$. Then since x is colored with red and z is colored with blue, the edge $(x, z) \in E(G')$. But this is a contradiction to the fact that M_I is an induced matching. From the above discussions, we can obtain that each vertex in $V_R \cap V(M_I)$ has exactly one neighbor in $V_B \cap V(M_I)$ and vice-versa. ◀

Case 2. Now suppose that in G' the matching M has less than $N'(3, b, k)$ edges, and thus, $\text{tw}(G') \leq 2 \cdot N'(3, b, k)$. Now in G' , we look for a pair of perfectly matched sets (X, Y) where $X \subseteq V_R$ and $Y \subseteq V_B$. Let us denote this version of PERFECTLY MATCHED SETS as the *colored*-PERFECTLY MATCHED SETS problem. Aravind et al. [1] designed an FPT algorithm for PERFECTLY MATCHED SETS parameterized by the treewidth of the given graph. They use a *nice* tree decomposition of the graph, where in each bag $\beta(t)$, $X \cap \beta(t)$ and $Y \cap \beta(t)$ play a crucial role in the construction of their algorithm. To adapt their algorithm for *colored*-PERFECTLY MATCHED SETS, we only need to enforce that $X \cap \beta(t)$ and $Y \cap \beta(t)$ are selected from V_R and V_B , respectively. Precisely in Section 5.3 of their draft [1], $A \cap \beta(t) = A_t$

and $B \cap \beta(t) = B_t$ can be replaced by $A \cap (\beta(t) \cap V_R) = A_t$ and $B \cap (\beta(t) \cap V_B) = B_t$, respectively, to obtain an algorithm for the colored version. Notice that they denote the desired perfectly matched sets by (A, B) while we do it by (X, Y) . Hence we have an FPT algorithm running in time $2^{\mathcal{O}(\text{tw}(G'))} \cdot n^{\mathcal{O}(1)}$ to obtain a pair of perfectly matched sets (X, Y) of G' of size k where $X \subseteq V_R, Y \subseteq V_B$. We remark that the algorithm given by [1] can actually compute such a set by the standard backtracking technique, and thus even for our colored case, we can compute a pair of perfectly matched sets in G' . Now we claim the following.

► **Observation 15.** (X, Y) is also a pair of perfectly matched sets of G .

Proof. Suppose (X, Y) is not a pair of perfectly matched sets of G . Notice that $E(G') \subseteq E(G)$ and hence there is a vertex v in X with more than one neighbor in Y or there is a vertex u in Y with more than one neighbor in X . Without loss of generality let such a vertex v be in X . Let two of its neighbors in Y be y_1 and y_2 . But the edges $\{v, y_1\}$ and $\{v, y_2\}$ are also in G' as they have endpoints with differing colors. But this contradicts the fact that (X, Y) is a pair of perfectly matched sets of G' . ◀

In the construction of G' from G , we delete edges with endpoints in the same color classes. Hence a pair of perfectly matched sets of G may not remain a pair of perfectly matched sets of G' . But in the claim below, we show that for a fixed size of perfectly matched sets, the chances of such an event happening stays low.

► **Observation 16.** Any k -sized perfectly matched sets (X, Y) of G is also a perfectly matched sets of G' with probability at least 2^{-2k} .

Proof. The probability that all vertices of X are colored red and all vertices of Y are colored blue is at least 2^{-2k} . Thus we can obtain that with probability at least 2^{-2k} (X, Y) is also a perfectly matched sets of G' . ◀

The proof of the following lemma follows from Observations 14, 15 and 16 with the standard trick of making independent runs of the discussed algorithm.

► **Lemma 17.** There exists a randomized FPT algorithm running in time $2^{\mathcal{O}(N'(3,b,k)+k)} \cdot n^{\mathcal{O}(1)}$ that, given a PERFECTLY MATCHED SETS instance (G, k) on $K_{b,b}$ -free graphs, either reports a failure or finds a pair of perfectly matched sets in G of size at least k . Moreover, if the algorithm is given a yes-instance, it returns a solution with constant probability.

We now explain the derandomization procedure for the above algorithm. It involves deterministically constructing a family \mathcal{F} of coloring functions $f : [n] \rightarrow [2]$ rather than selecting a random coloring $\chi : [n] \rightarrow [2]$ such that it is assured that one of the functions from \mathcal{F} colors one set from a pair of perfectly matched sets of size k (when (G, k) is a yes-instance) with color 1 and the other set with color 2. To this end, we will use the following.

► **Definition 18** (Definition 5.19, [8]). An (n, k) -universal set is a family \mathcal{U} of subsets of $[n]$ such that for each $S \subseteq [n]$ of size k , the family $\{A \cap S : A \in \mathcal{U}\}$ contains all 2^k subsets of S .

► **Proposition 19** (Theorem 5.20, [8]). For any $n, k \geq 1$, we can construct an (n, k) -universal set of size $2^k k^{\mathcal{O}(\log k)} \log n$ in time $2^k k^{\mathcal{O}(\log k)} n \log n$.

We assume that $V(G) = [n]$ (otherwise we can relabel the vertices). We first construct an $(n, 2k)$ -universal set, \mathcal{U} , using the above proposition. Now we construct a family of function \mathcal{F} from $[n]$ to $\{1, 2\}$ as follows, where \mathcal{F} is initialized to \emptyset . For each $U \in \mathcal{U}$, add the function

$f_U : [n] \rightarrow [2]$, where $f^{-1}(1) = U$. Note that if G has a pair of perfectly matched sets (A, B) of size k , then there is $U \in \mathcal{U}$, such that $(A \cup B) \cap U = A$. Thus at least one function in \mathcal{F} is the correct coloring for us. We can iterate over each of the colorings given by \mathcal{F} , and this leads us to the following result.

► **Theorem 20.** PERFECTLY MATCHED SETS on $K_{b,b}$ -free graphs admits a deterministic FPT algorithm running in time $k^{\mathcal{O}(\log k)} \cdot 2^{\mathcal{O}(N'(3,b,k)+k)} \cdot n^{\mathcal{O}(1)}$.

6 Kernelization for PERFECTLY MATCHED SETS on d -degenerate graphs

In this section, we design a polynomial kernel for d -degenerate graphs, and thus prove Theorem 4. We design our kernel using the *strong systems of distinct representatives* [17] (to be defined shortly). Recall that a graph G is d -degenerate if every induced subgraph of it contains a vertex of degree at most d . We start by stating the definition of strong systems of distinct representatives and a useful result regarding it.

► **Definition 21** (Strong systems of distinct representatives, [18]). A k -tuple (x_1, x_2, \dots, x_k) is a *system of distinct representatives* for sets S_1, S_2, \dots, S_k , if for each $i \in [k]$, $x_i \in S_i$. Moreover, it is *strong* if additionally, for each $i \in [k]$ and $j \in [k] \setminus \{i\}$, $x_i \notin S_j$.

► **Proposition 22** (Theorem 8.12 [17]). Consider any family \mathcal{F} with more than $\binom{r+k}{k}$ distinct sets of sizes at most r . Then, at least $k+2$ sets in this family have a strong system of distinct representatives.

The following property of a d -degenerate graph follows directly from the definition.

► **Proposition 23.** A d -degenerate graph on n vertices has at most dn edges.

Next, we give a lower bound on the number of low-degree vertices in a d -degenerate graph.

► **Lemma 24.** Let G be d -degenerate graph with $n \geq 6$ vertices. Then G has strictly more than $5n/6$ vertices of degree at most $12d$.

Proof. Let G be d -degenerate graph with n vertices. By Proposition 23, the number of edges in G is at most dn . So the sum of the degrees of the vertices in G is bounded by $2dn$. Assume that, there are at most $5n/6$ vertices of degree at most $12d$ in G . Then we have a set $U \subseteq V(G)$ of at least $n/6 \geq 1$ vertices of degree strictly more than $12d$. Now the sum of the degrees of the vertices in U is strictly more than $(n/6) \cdot 12d = 2dn$, a contradiction. Hence there are strictly more than $5n/6$ vertices of degree at most $12d$ in G . ◀

► **Observation 25.** In a pair of perfectly matched sets (A, B) of a graph G , there are at most two non-adjacent vertices $x, y \in A \cup B$ such that $N(x) = N(y)$.

Proof. Let $x, y, z \in A \cup B$ be three pairwise non-adjacent vertices such that $N(x) = N(y) = N(z)$. At least two of these vertices are either in A or B . Without loss of generality let $x, y \in A$. But then x and y , both have the exactly same neighbors in B , which contradicts that $A \cup B$ is a pair of perfectly matched sets of G . ◀

With Observation 25, we obtain the following reduction rule.

► **Reduction Rule 1.** Let u, v, w be three distinct vertices in $V(G)$ such that $N(u) = N(v) = N(w)$, then reduce (G, k) to $(G - w, k)$.

► **Lemma 26.** *Reduction Rule 1 is safe.*

Proof. Consider an application of Reduction Rule 1 in which a vertex, say $w \in V(G)$ was deleted because there are two distinct vertices u and v other than w such that $N(u) = N(v) = N(w)$. We will prove that (G, k) is a yes-instance of PERFECTLY MATCHED SETS if and only if $(G - w, k)$ is a yes-instance of PERFECTLY MATCHED SETS.

If $(G - w, k)$ is a yes-instance, any pair of perfectly matched sets in $G - w$ is also a pair of perfectly matched sets in G , thus (G, k) must also be a yes-instance. For the other direction suppose that (G, k) is a yes-instance of the problem, and we have two disjoint sets $A, B \subseteq V(G)$ such that every vertex in A has exactly one neighbor in B and vice-versa. If $w \notin A \cup B$, then (A, B) is a pair of perfectly matched sets in $G - w$ of size k , and we are done. Else, exactly one of A and B must contain w . Without loss of generality we assume that $w \in A$. From Observation 25, we know that $|(A \cup B) \cap \{u, v, w\}| \leq 2$. Now neither v nor u belongs to A . If $B \cap \{u, v\} = \emptyset$, then $(A \setminus \{w\} \cup \{u\}, B)$ is a pair of perfectly matched sets in $G - w$ of size k . Else, exactly one of v or u belongs to B , say $u \in B$ (the other case is symmetric). Then, $(A \setminus \{w\} \cup \{v\}, B)$ is a pair of perfectly matched sets in $G - w$ of size k . ◀

We are now ready to prove Theorem 4.

Proof of Theorem 4. Let (G, k) be an instance of PERFECTLY MATCHED SETS where G is a d -degenerate graph. If Reduction Rule 1 on (G, k) is applicable, then we apply it in polynomial time and reduced the number of vertices. When the reduction rule is no longer applicable, we do the following. Let X be the set of vertices of with degree at most $12d$, and let $t = |X|$. Consider the family $\mathcal{F} = \{N(u) \mid u \in X\}$ (with repetitions removed). By the non-applicability of Reduction Rule 1 and Lemma 24, we can obtain that $|\mathcal{F}| \geq t/2 \geq (5n/6)/2 = 5n/12$. Also note that each set in \mathcal{F} has size at most $12d$.

If $|\mathcal{F}| \leq \binom{12d+k}{k}$, then $5n/12 < \mathcal{F} \leq \binom{12d+k}{k}$. Therefore n , i.e., the number of vertices in G is bounded by $k^{O(d)}$. Otherwise, $|\mathcal{F}| > \binom{12d+k}{k}$, and we argue that (G, k) is a yes-instance. From Proposition 22, at least $k + 2$ of these sets form \mathcal{F} have a strong system of distinct representatives, say these sets are $N(v_1), N(v_2), \dots, N(v_{k+2})$ and $(u_1, u_2, \dots, u_{k+2})$ is its strong system of distinct representatives. Let $A = \{v_1, v_2, \dots, v_{k+2}\}$ and $B = \{u_1, u_2, \dots, u_{k+2}\}$. Note that for each $i \in [k + 2]$, we have $\{v_i, u_i\} \in E(G)$. For any $i \in [k + 2]$ and $j \in [k] \setminus \{i\}$, $\{v_i, u_j\} \notin E(G)$, as $u_j \notin N(v_i)$ by the definition of a strong system of distinct representatives. Thus, (A, B) is a pair of perfectly matched sets of size at least $(k + 2)$ in G . ◀

As planar graphs are 5-degenerate, the above result directly gives us a polynomial kernel (which is not linear!) for planar graphs. We next obtain a linear kernel for planar graphs.

Linear Kernel on Planar Graphs. We describe a procedure to obtain a linear-sized vertex kernel for planar graphs. To this end, we state the following useful result.

► **Proposition 27** (Theorem 4.11, [18]). *A twinless planar graph with $n \geq 2$ vertices contains an induced matching of size at least $n/40$.*

From Proposition 27, we have the following observation.

► **Observation 28.** *Let G be a planar graph on $n \geq 4$ vertices such that there are no three vertices that are pairwise false twins. Then G contains a pair of perfectly matched sets of size at least $n/80$.*

Proof. From G , we can construct a twinless planar graph G' by keeping exactly one of the false twins i.e. for any two false twins u and v , we delete exactly one of them. Hence G' is a twinless planar graph with size at least $n/2 \geq 2$ vertices. From Proposition 27, G' has an induced matching of size at least $n/80$, which is also an induced matching in G . But such an induced matching gives us a pair of perfectly matched sets of size $n/80$. ◀

► **Theorem 29.** PERFECTLY MATCHED SETS on planar graphs admits an $\mathcal{O}(k)$ -sized kernel.

Proof. Consider an instance (G, k) of the problem, where G is a planar graph with n vertices. Apply Reduction Rule 1 as long as it is applicable. If $|V(G)| < 2$, then we are done. Otherwise, from Observation 28, G has a pair of perfectly matched sets with size at least $n/80$. If $k \leq n/80$, then the given instance is a yes-instance, and otherwise $|V(G)| < 80k$. ◀

References

- 1 N. R. Aravind and Roopam Saxena. Perfectly Matched Sets in Graphs: Hardness, Kernelization Lower Bound, and FPT and Exact Algorithms. *CoRR*, abs/2107.08584, 2021. [arXiv:2107.08584](#).
- 2 Paul S. Bonsma. The complexity of the matching-cut problem for planar graphs and other graph classes. *J. Graph Theory*, 62(2):109–126, 2009.
- 3 Leizhen Cai, Siu Man Chan, and Siu On Chan. Random Separation: A New Method for Solving Fixed-Cardinality Optimization Problems. In Hans L. Bodlaender and Michael A. Langston, editors, *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, volume 4169 of *Lecture Notes in Computer Science*, pages 239–250. Springer, 2006.
- 4 Kathie Cameron. Induced matchings in intersection graphs. *Discret. Math.*, 278(1-3):1–9, 2004.
- 5 Yixin Cao and Dániel Marx. Interval Deletion Is Fixed-Parameter Tractable. *ACM Trans. Algorithms*, 11(3):21:1–21:35, 2015.
- 6 Chi-Yeh Chen, Sun-Yuan Hsieh, Hoàng-Oanh Le, Van Bang Le, and Sheng-Lung Peng. Matching Cut in Graphs with Large Minimum Degree. *Algorithmica*, 83(5):1238–1255, 2021.
- 7 Vasek Chvátal. Recognizing decomposable graphs. *J. Graph Theory*, 8(1):51–53, 1984.
- 8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 9 Konrad K. Dabrowski, Marc Demange, and Vadim V. Lozin. New results on maximum induced matchings in bipartite graphs and beyond. *Theor. Comput. Sci.*, 478:33–40, 2013.
- 10 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Dimitrios M. Thilikos. The Bidimensional Theory of Bounded-Genus Graphs. *SIAM J. Discret. Math.*, 20(2):357–371, 2006.
- 11 Reinhard Diestel. Graph theory. *Graduate texts in mathematics*, 173, 2017.
- 12 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.
- 13 Shimon Even, Oded Goldreich, Shlomo Moran, and Po Tong. On the NP-completeness of certain network testing problems. *Networks*, 14(1):1–24, 1984.
- 14 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.
- 15 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Subexponential algorithms for partial cover problems. *Inf. Process. Lett.*, 111(16):814–818, 2011.
- 16 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *CoRR*, abs/1311.3899, 2013. [arXiv:1311.3899](#).
- 17 Stasys Jukna. *Extremal Combinatorics – With Applications in Computer Science*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2011.
- 18 Iyad A. Kanj, Michael J. Pelsmayer, Marcus Schaefer, and Ge Xia. On the induced matching problem. *J. Comput. Syst. Sci.*, 77(6):1058–1070, 2011.

- 19 Johannes Köbler, Sebastian Kuhnert, and Osamu Watanabe. Interval graph representation with given interval and intersection lengths. *Journal of Discrete Algorithms*, 34:108–117, 2015.
- 20 Łukasz Kowalik, Matjaž Krnc, Tomasz Waleń, et al. Improved induced matchings in sparse graphs. *Discrete Applied Mathematics*, 158(18):1994–2003, 2010.
- 21 Hoàng-Oanh Le and Van Bang Le. On the Complexity of Matching Cut in Graphs of Fixed Diameter. In *27th International Symposium on Algorithms and Computation, ISAAC 2016, December 12-14, 2016, Sydney, Australia*, volume 64 of *LIPICs*, pages 50:1–50:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.
- 22 Van Bang Le and Jan Arne Telle. The Perfect Matching Cut Problem Revisited. In *Graph-Theoretic Concepts in Computer Science – 47th International Workshop, WG 2021, Warsaw, Poland, June 23-25, 2021, Revised Selected Papers*, volume 12911 of *Lecture Notes in Computer Science*, pages 182–194. Springer, 2021.
- 23 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|v|} |E|)$ Algorithm for Finding Maximum Matching in General Graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27. IEEE Computer Society, 1980.
- 24 Hannes Moser and Somnath Sikdar. The parameterized complexity of the induced matching problem. *Discret. Appl. Math.*, 157(4):715–727, 2009.
- 25 Augustine M. Moshi. Matching cutsets in graphs. *J. Graph Theory*, 13(5):527–536, 1989.
- 26 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
- 27 Larry J. Stockmeyer and Vijay V. Vazirani. NP-Completeness of Some Generalizations of the Maximum Matching Problem. *Inf. Process. Lett.*, 15(1):14–19, 1982.
- 28 Michele Zito. Induced matchings in regular graphs and trees. In *Graph-Theoretic Concepts in Computer Science*, pages 89–101, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

On the Hardness of Generalized Domination Problems Parameterized by Mim-Width

Brage I. K. Bakkane ✉

University of Bergen, Norway

Lars Jaffke ✉

University of Bergen, Norway

Abstract

For nonempty $\sigma, \rho \subseteq \mathbb{N}$, a vertex set S in a graph G is a (σ, ρ) -dominating set if for all $v \in S$, $|N(v) \cap S| \in \sigma$, and for all $v \in V(G) \setminus S$, $|N(v) \cap S| \in \rho$. The MIN/MAX (σ, ρ) -DOMINATING SET problems ask, given a graph G and an integer k , whether G contains a (σ, ρ) -dominating set of size at most k and at least k , respectively. This framework captures many well-studied graph problems related to independence and domination. Bui-Xuan, Telle, and Vatshelle [TCS 2013] showed that for finite or co-finite σ and ρ , the MIN/MAX (σ, ρ) -DOMINATING SET problems are solvable in XP time parameterized by the *mim-width* of a given branch decomposition of the input graph. In this work we consider the parameterized complexity of these problems and obtain the following: For minimization problems, we complete several scattered W[1]-hardness results in the literature to a full dichotomy into polynomial-time solvable and W[1]-hard cases, and for maximization problems we obtain the same result under the additional restriction that σ and ρ are finite sets. All W[1]-hard cases hold assuming that a *linear* branch decomposition of bounded mim-width is given, and with the solution size being an additional part of the parameter. Furthermore, for all W[1]-hard cases we also rule out $f(w)n^{o(w/\log w)}$ -time algorithms assuming the Exponential Time Hypothesis, where f is any computable function, n is the number of vertices and w the mim-width of the given linear branch decomposition of the input graph.

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

Keywords and phrases generalized domination, linear mim-width, W[1]-hardness, Exponential Time Hypothesis

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.3

Related Version *First author's Master's thesis*: <https://bora.uib.no/bora-xmlui/handle/11250/3001140> [2]

Funding *Lars Jaffke*: Supported by the Norwegian Research Council (project number 274526).

1 Introduction

Maximum induced matching width [35], or mim-width for short, is a width measure of graphs based on branch decompositions over the vertex set. On the one hand, mim-width has high expressive power, while on the other hand, it allows for efficient algorithms for many fundamental NP-hard problems when the input graph is given together with a decomposition of small width. Mim-width strictly generalizes tree-width and clique-width, in the sense that a bound on each of the latter measures implies a bound on the mim-width, while there are n -vertex graphs that have clique-width $\Omega(\sqrt{n})$ and mim-width 1 [3, 24]. Mim-width and twin-width [9] are incomparable. Moreover, the mim-width remains bounded by a constant on several deeply studied graph classes such as interval graphs, permutation graphs, and some of their generalizations, see e.g. [3, 10, 27, 35], as well as several graph classes excluding



© Brage I. K. Bakkane and Lars Jaffke;

licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 3; pp. 3:1–3:19

Leibniz International Proceedings in Informatics



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

■ **Table 1** Some examples of MIN/MAX (σ, ρ) -DOMINATING SET problems and their complexity when parameterized by the mim-width of a given (linear) branch decomposition of the input graph plus solution size. In all occurrences, the value of d is a fixed constant.

Standard name	σ	ρ	MIN	MAX
INDEPENDENT SET	$\{0\}$	\mathbb{N}	P	W[1]-h [19]
DOMINATING SET	\mathbb{N}	$\mathbb{N} \setminus \{0\}$	W[1]-h [19]	P
INDEPENDENT DOMINATING SET	$\{0\}$	$\mathbb{N} \setminus \{0\}$	W[1]-h [19]	W[1]-h [19]
TOTAL DOMINATING SET	$\mathbb{N} \setminus \{0\}$	$\mathbb{N} \setminus \{0\}$	W[1]-h [28]	P
STRONG STABLE SET/2-PACKING	$\{0\}$	$\{0, 1\}$	P	W[1]-h [This]
PERFECT CODE	$\{0\}$	$\{1\}$	W[1]-h [This]	W[1]-h [This]
TOTAL NEARLY PERFECT SET	$\{0, 1\}$	$\{0, 1\}$	P	W[1]-h [This]
WEAKLY PERFECT DOMINATING SET	$\{0, 1\}$	$\{1\}$	W[1]-h [This]	W[1]-h [This]
TOTAL PERFECT DOMINATING SET	$\{1\}$	$\{1\}$	W[1]-h [This]	W[1]-h [This]
INDUCED MATCHING	$\{1\}$	\mathbb{N}	P	W[1]-h [28]
DOMINATING INDUCED MATCHING	$\{1\}$	$\mathbb{N} \setminus \{0\}$	W[1]-h [28]	W[1]-h [28]
PERFECT DOMINATING SET	\mathbb{N}	$\{1\}$	W[1]-h [This]	?
d -DOMINATING SET	\mathbb{N}	$\{d, d + 1, \dots\}$	W[1]-h [28]	?
INDUCED d -REGULAR SUBGRAPH	$\{d\}$	\mathbb{N}	P	W[1]-h [28]
SUBGRAPH OF MIN DEGREE $\geq d$	$\{d, d + 1, \dots\}$	\mathbb{N}	P	?
INDUCED SUBG. OF MAX DEGREE $\leq d$	$\{0, 1, \dots, d\}$	\mathbb{N}	P	W[1]-h [28]

small graphs as induced subgraphs [11]. This implies that algorithms for graphs of bounded mim-width often unify and extend several algorithmic results on graph classes from the literature.

In recent years, an increasing number of problems has been shown to admit such algorithms [4, 5, 6, 13, 20, 25, 28, 29, 30]. However, all of these algorithms run in XP time when parameterized by the mim-width of the given branch decomposition of the input graph, and the parameterized complexity of these problem is much less understood. In this work, we contribute to the systematic study of the parameterized complexity of problems parameterized by the mim-width of a given (linear) branch decomposition of the input graph, by showing dichotomies into polynomial-time solvable and W[1]-hard cases for locally checkable minimization and maximization problems.

The *locally checkable vertex subset problems*, or (σ, ρ) -*domination problems* [34], capture many problems related to independence and domination in graphs in a unified framework. Here, a problem is formulated by prescribing for its solutions, which are vertex sets, for each vertex v in the graph, how many neighbors it has to have in the set, depending on whether v is in the set or not. Concretely, for two nonempty sets $\sigma, \rho \subseteq \mathbb{N}$, a (σ, ρ) -*dominating set* in a graph G is a set of vertices S such that for each vertex in S , the number of neighbors it has in S is an element of σ , and for each vertex outside of S , the number of neighbors it has in S is an element of ρ . The MIN/MAX (σ, ρ) -DOMINATING SET problems ask, given a graph G and an integer k , whether G contains a (σ, ρ) -dominating set of size at most k and at least k , respectively. Observe for instance that the MIN $(\mathbb{N}, \mathbb{N} \setminus \{0\})$ -DOMINATING SET problem is the MINIMUM DOMINATING SET problem, and that the MAX $(\{0\}, \mathbb{N})$ -DOMINATING SET problem is the MAXIMUM INDEPENDENT SET problem. Many more problems can be expressed in this way, see Table 1 for examples. While such problems are often NP-complete, several (σ, ρ) -domination problems are trivial to solve – for instance all MIN (σ, ρ_0) -DOMINATING SET problems, where $0 \in \rho_0$. This is simply because the empty set is a solution to any such MIN (σ, ρ_0) -DOMINATING SET problem.

The (σ, ρ) -domination problems play a central role in the algorithmic study of mim-width. They are among the first problems that have been shown to be solvable in XP time parameterized by the mim-width of a given branch decomposition of the input graph by Bui-Xuan et al. [13] (whenever σ and ρ are finite or co-finite), and contain the first problems for which W[1]-hardness in this parameterization was shown. Fomin et al. [19] proved that MAXIMUM INDEPENDENT SET and MINIMUM DOMINATING SET are W[1]-hard parameterized by the mim-width of a given linear branch decomposition of the input graph. The W[1]-hardness of several other (σ, ρ) -domination problems was shown by Jaffke et al. [28]. However, these results are far from complete dichotomies. For minimization problems, we achieve such a dichotomy in this work, and for maximization problems, whenever σ and ρ are finite. In both cases, hardness already holds for the more restrictive parameterization by *linear* mim-width.

► **Theorem 1.** *Let $\sigma, \rho \subseteq \mathbb{N}$ be nonempty. If $0 \in \rho$, then MIN (σ, ρ) -DOMINATING SET is polynomial-time solvable, otherwise it is W[1]-hard parameterized by the mim-width of a given linear branch decomposition of the input graph plus solution size.*

► **Theorem 2.** *Let $\sigma, \rho \subseteq \mathbb{N}$ be nonempty and finite. If $\rho = \{0\}$, then MAX (σ, ρ) -DOMINATING SET is polynomial-time solvable, otherwise it is W[1]-hard parameterized by the mim-width of a given linear branch decomposition of the input graph plus solution size.*

Note that since the solution size can be a part of the parameter in the previous theorems, they extend several hardness results for MIN/MAX (σ, ρ) -DOMINATING SET problems parameterized by solution size due to Golovach et al. [23]. They obtained a dichotomy into polynomial-time solvable and W[1]-complete for MIN (σ, ρ) -DOMINATING SET when σ and ρ are finite.

Mim-width and the Exponential Time Hypothesis. All known XP-algorithms for problems parameterized by mim-width, except for the ones in [6], run in $n^{O(w)}$ time, where n is the number of vertices of the input graph, and w the mim-width of the given branch decomposition. A natural follow-up question to Theorems 1 and 2 is whether the dependence on w can be improved, in particular if one of these problems admits an $n^{o(w)}$ time algorithm. Several of the reductions given in [19, 28] start from the MULTICOLORED CLIQUE problem parameterized by the number of color classes k , and the mim-width of the instance constructed in the reduction is quadratic in k . This is due to the fact that the gadgeteering depends on the number of edges in the (complete) quotient graph associated with the color partition of the input graph. Therefore these reductions only rule out $f(w)n^{o(\sqrt{w})}$ time algorithms under the Exponential Time Hypothesis (ETH). We can observe that the same reduction works if we start from the PARTITIONED SUBGRAPH ISOMORPHISM problem parameterized by the number of edges h in the pattern graph, and the mim-width of the reduced instance remains $O(h)$;¹ this gives a strengthened lower bound of $f(w)n^{o(w/\log w)}$ time by a theorem due to Marx [31]. All reductions presented in this work start from the PARTITIONED SUBGRAPH ISOMORPHISM problem and give the improved lower bounds under the ETH. It remains an open problem to close the gap between the $f(w)n^{o(w/\log w)}$ time lower bounds and the $n^{O(w)}$ time algorithms.

¹ For a worked out example, see [2].

► **Corollary 3.** *Let $\sigma, \rho \subseteq \mathbb{N}$ be nonempty. If $0 \notin \rho$, then MIN (σ, ρ) -DOMINATING SET does not admit $f(w)n^{o(w/\log w)}$ time algorithms, for any computable function f , on n -vertex graphs given with a linear branch decomposition of mim-width w , unless the ETH is false. If σ and ρ are finite and $\rho \neq \{0\}$, then the same holds for MAX (σ, ρ) -DOMINATING SET.*

Related work. Bui-Xuan et al. [13] showed that the MIN/MAX (σ, ρ) -DOMINATING SET problems are XP-time solvable parameterized by the mim-width of a given branch decomposition of the input graph, whenever σ and ρ are either finite or co-finite. The first W[1]-hardness proofs for several MIN/MAX (σ, ρ) -DOMINATING SET problems were given in [19, 28]. However, several other problems have been shown to be even harder on graphs of bounded mim-width, which often follows from the NP-completeness of problems on graph classes that have constant mim-width [3]. For instance, the following problems are para-NP-hard parameterized by the mim-width of a given linear branch decomposition of the input graph: CLIQUE and CO-DOMINATING SET [22, 35], GRAPH COLORING [21], MAXIMUM CUT [1], and HAMILTONIAN PATH [29]. The NP-completeness of MIN/MAX (σ, ρ) -DOMINATING SET problems has been systematically studied by Telle [33], and Golovach et al. [23] considered their complexity parameterized by solution size.

Methods. As mentioned above, all reductions we give start from the PARTITIONED SUBGRAPH ISOMORPHISM (PSI) problem. Here, we are given two graphs G and K , and a partition of $V(G)$ where each part is associated with a vertex from K , and the question is whether G contains K as a subgraph witnessed by an isomorphism that respects the partition of $V(G)$. This problem is known to be W[1]-hard parameterized by $h = |E(K)|$ and not to have $f(h)n^{o(h/\log h)}$ -time algorithms, where $n = |V(G)|$, unless the Exponential Time Hypothesis fails [18, 31, 32].

We give a high level outline of how we reduce the PSI problem to any (σ, ρ) -DOMINATING SET problem. As the overall strategy is the same for all choices of σ and ρ , and whether we are concerned with minimization or maximization, we do not specify which case we are in for now. Let (G, K) be an instance of PSI and for ease of reference, suppose that $V(K) = \{1, \dots, k\}$, and let V_i be the part of the partition of $V(G)$ corresponding to vertex i . The graph H of the (σ, ρ) -DOMINATING SET instance contains, for each V_i , a set \hat{S}_i of *selection vertices* that encodes which vertex of V_i is chosen in a potential solution to (G, K) . For each edge $ij \in E(K)$, we add a subgraph to H that preserves information about adjacencies between the vertices in V_i and V_j , but induces cuts that have no induced matchings of size larger than two. This construction is adapted from the work of Fomin et al. [19]. It ensures that once a (σ, ρ) -dominating set D contains precisely one vertex from each \hat{S}_i , then the remainder of the vertices in D witness the existence of a K -subgraph in G .

To ensure that each solution to the (σ, ρ) -DOMINATING SET instance picks precisely one vertex from each \hat{S}_i , we add gadgets to H that depend on the choice of σ and ρ and whether we are concerned with a minimization or a maximization problem. These gadgets are constructed carefully enough so that the linear mim-width of H does not increase prohibitively. In the end, we have a partition of H such that each subgraph induced by a part has linear mim-width that only depends on some fixed constants contained in σ and ρ , and such that the cuts between the parts do not contain large induced matchings either. By adapting a lemma of Brettell et al. [12] to linear mim-width, and with a slightly more careful analysis, we conclude that we can construct in polynomial time a linear branch decomposition of H that has mim-width $O(h)$.

The high degree of generality in our reductions is achieved by the following: the construction combined with the budget are tight enough so that, roughly speaking, in minimization problems, each vertex can only be *minimally dominated* and in maximization problems, each vertex has to be *maximally dominated*. This means that in either case, σ and ρ only contain one relevant value for feasible solutions: for minimization that is $\varsigma = \min \sigma$ and $\varrho = \min \rho$, and for maximization ς becomes $\max \sigma$ and ϱ becomes $\max \rho$. The linear mim-width of H depends on ς and ϱ , and in the case of minimization problems, these are always constants. In the case of maximization, however, ς and ϱ are only constant when σ and ρ are finite.

Throughout the paper, proofs of statements marked with “♣” and full proofs of sketches are deferred to the full version.

2 Preliminaries

For basic background in graph theory, we refer to [15], and for basics in parameterized complexity, we refer the reader to [14, 16]. We use the following notation: $\mathbb{N} = \{0, 1, 2, 3, \dots\}$, $[n] = \{1, 2, \dots, n\}$, $[n]_0 = \{0, 1, 2, \dots, n\}$. For a graph G , we denote by $V(G)$ its vertex set and by $E(G)$ its edge set. For $A, B \subseteq V(G)$ with $A \cap B = \emptyset$, we let $G[A, B]$ be the bipartite graph with vertex set $A \cup B$ and edge set $\{ab \mid ab \in E(G), a \in A, b \in B\}$. A *matching* in a graph G is a set $M \subseteq E(G)$ of pairwise disjoint edges. We say that M is *induced* if there are no additional edges between the endpoints of the edges in M ; that is, if u is an endpoint of some edge in M and v is some endpoint of some edge in M , then either $uv \in M$ or $uv \notin E(G)$. For a graph G , we denote by $\text{cc}(G)$ the set of its connected components. For additional clarification of basic graph theoretic concepts and notation we refer to the full version.

Mim-Width. For a graph G and $A, B \subseteq V(G)$ with $A \cap B = \emptyset$ we define $\text{cutmim}_G(A, B)$ to be the largest size of any induced matching in $G[A, B]$. For a set $A \subseteq V(G)$, we let $\text{mim}_G(A) = \text{cutmim}_G(A, V(G) \setminus A)$.

A *branch decomposition* of a graph G is a pair (T, \mathcal{L}) , where T is a tree where all of whose vertices have degree at most 3, and \mathcal{L} a bijection mapping the vertices of the graph $V(G)$ to the leaves of the tree T . For a subtree T' of T , we denote by $V_{T'}$ the vertices of G that are mapped to leaves of T' . The *mim-width* of (T, \mathcal{L}) is $\text{mimw}_G(T, \mathcal{L}) = \max_{e \in E(G), T' \in \text{cc}(T-e)} \text{mim}_G(V_{T'})$. The *mim-width* of G , denoted by $\text{mimw}(G)$, is the minimum mim-width over all its branch decompositions.

A branch decomposition (T, \mathcal{L}) is called *linear* if T is a caterpillar graph, i.e., a tree containing an induced path P such that each vertex in $V(T) \setminus V(P)$ has precisely one neighbor on P . The *linear mim-width* of a graph G , denoted by $\text{linmimw}(G)$, is the minimum mim-width over all its linear branch decompositions. Linear branch decompositions can be equated with linear orderings of the vertex set of a graph. For a linear order Λ of the vertices of G , we will therefore write $\text{mimw}_G(\Lambda)$ for the mim-width of the linear branch decomposition corresponding to Λ . In all definitions given in these last paragraphs, we may drop G as a subscript if it is clear from the context.

Exponential-Time Hypothesis. The Exponential-Time Hypothesis (ETH) is a conjecture about the complexity of the 3-SAT problem, which given a boolean formula in conjunctive normal form and clauses of size at most three, asks whether it has a satisfying assignment.

► **Conjecture 4** (ETH [26], informal). *The 3-SAT problem cannot be solved in $2^{o(n)}$ time, where n is the number of variables of the input formula.*

2.1 Generalized dominating set problems

Let $\sigma, \rho \subseteq \mathbb{N}$, and let G be a graph. A vertex set $S \subseteq V(G)$ is a (σ, ρ) -dominating set, if for all $v \in V(G)$: If $v \in S$, then $|N(v) \cap S| \in \sigma$, and if $v \notin S$, then $|N(v) \cap S| \in \rho$. The computational problems associated with (σ, ρ) -dominating sets we consider in this work are:

MIN/MAX (σ, ρ) -DOMINATING SET

Input: Graph G , integer k

Question: Does G contain a (σ, ρ) -dominating set of size at most/at least k ?

Many maximization and minimization problems formulated in this manner are computationally hard, in the sense that they are NP-hard and W[1]-hard with solution size as a parameter. We now discuss the exceptions that are relevant for this work, i.e. some cases when the MIN/MAX (σ, ρ) -DOMINATING SET problems are polynomial-time solvable.

Trivial minimization problems. Whenever $0 \in \rho$, the empty set is a solution of the MIN (σ, ρ) -DOMINATING SET problem. This case is then trivial as any algorithm can always return the empty set as a valid optimal solution. These are the only trivial cases for minimization.

Trivial maximization problems. We focus here on trivial cases where σ and ρ are finite, since these are the cases for which we show hardness in this work. Note however that there are more trivial cases when σ and ρ need not be finite, for instance when $\sigma = \mathbb{N}$: in this case, the entire vertex set of the input graph is a valid optimal solution.

If $\rho = \{0\}$, then any solution has to consist of connected components of the input graph. Suppose S is a $(\sigma, \{0\})$ -dominating set of a graph G and let $C_1, C_2 \in \text{cc}(G)$. Then, whether or not $C_1 \subseteq S$ is independent of whether or not $C_2 \subseteq S$. Furthermore, for any connected component $C \in \text{cc}(G)$, we can verify in polynomial time whether or not C can be contained in a $(\sigma, \{0\})$ -dominating set: we only have to check for all $v \in C$ that $\deg(v) \in \sigma$. Therefore we can use a greedy algorithm to solve the problem, by first identifying all connected components of the input graph followed by greedily including each connected component C in the solution if it passes the aforementioned check.

► **Observation 5.** Let $\sigma, \rho \subseteq \mathbb{N}$. If $0 \in \rho$, then MIN (σ, ρ) -DOMINATING SET is polynomial-time solvable, and if $\rho = \{0\}$, then MAX (σ, ρ) -DOMINATING SET is polynomial-time solvable.

2.2 Problem Definitions

We collect here the definitions of the problems that are relevant to this work. The following parameterized variant of the MIN/MAX (σ, ρ) -DOMINATING SET problems is the main object of study.

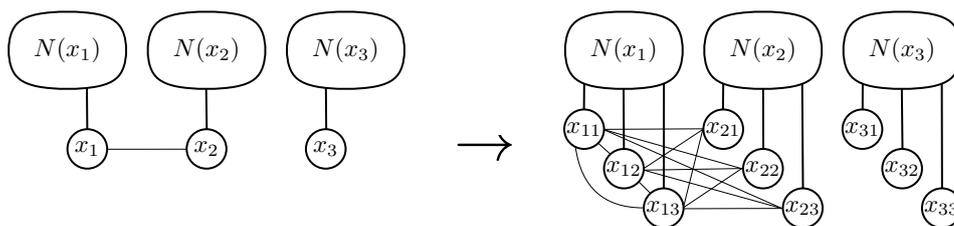
MIN/MAX (σ, ρ) -DOMINATING SET[LMIM + SOL]

Input: Graph G , integer k , linear order Λ of $V(G)$.

Parameter: $\text{mimw}(\Lambda) + k$.

Question: Does G contain a (σ, ρ) -dominating set of size at most/at least k ?

The starting point of our reductions will be the PARTITIONED SUBGRAPH ISOMORPHISM problem, which is known to be W[1]-hard and not to have $f(h)n^{o(h/\log h)}$ -time algorithms, unless the ETH is false [31].



■ **Figure 1** 2-Blowups of the vertices x_1 , x_2 , and x_3 . In vertex x_1 , a clique blowup was performed and in x_2 and x_3 and independent blowup.

PARTITIONED SUBGRAPH ISOMORPHISM

Input: (G, K, ϕ) , where G and K are graphs, and $\phi: V(G) \rightarrow V(K)$.
Parameter: $h = |E(K)|$.
Question: Is there an injective function $f: V(K) \rightarrow V(G)$ such that $ab \in E(K) \Rightarrow f(a)f(b) \in E(G)$ for all $a, b \in V(K)$, and $\phi(f(a)) = a$ for all $a \in V(K)$?

We introduce some notation that will be useful when talking about instances of PARTITIONED SUBGRAPH ISOMORPHISM. We say a function $f: V(K) \rightarrow V(G)$ *preserves neighbors* if $ab \in E(K) \Rightarrow f(a)f(b) \in E(G)$ for all $a, b \in V(K)$, and f *preserves colors* (relative to $\phi: V(G) \rightarrow V(K)$) if $\phi(f(a)) = a$ for all $a \in V(K)$. As these above mentioned hardness result from [31] also holds when the pattern graph K is connected, we will commonly make this assumption throughout the paper.

3 Graph operations and bounds on the linear mim-width

The following lemma can be seen as an analogue of a lemma due to Brettel et al. [12] for linear mim-width.

► **Lemma 6** (♣, Cf. Lemma 7 in [12]). *Let G be a graph, let $X = (X_1, \dots, X_p)$ be a partition of $V(G)$ such that $\text{cutmim}_G(X_i, X_j) \leq c$ for all distinct $i, j \in [p]$, and let G/X be the quotient graph of X . Then,*

$$\text{linmimw}(G) \leq |E(G/X)| \cdot c + \max_{i \in [p]} \text{linmimw}(G[X_i]).$$

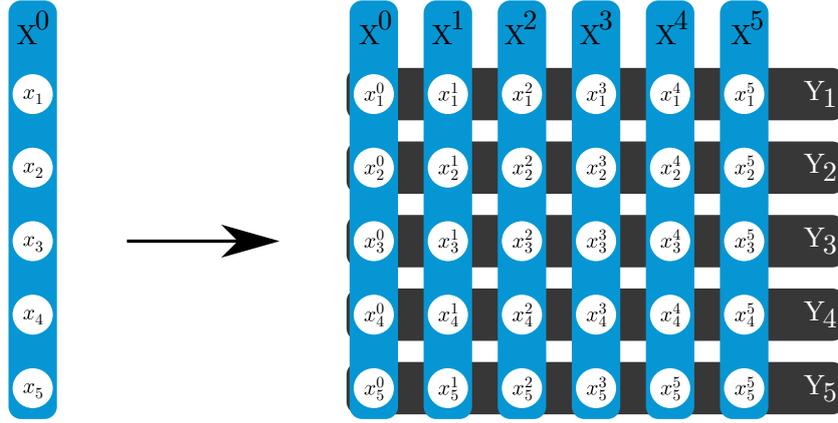
Moreover, if for all $i \in [p]$, Λ_i is a linear order of X_i , then one can in polynomial time construct a linear order Λ of G with

$$\text{mimw}(\Lambda) \leq |E(G/X)| \cdot c + \max_{i \in [p]} \text{mimw}(\Lambda_i).$$

The following operation is illustrated in Figure 1.

► **Definition 7** (Blowup). *Let G be a graph, $v \in V(G)$, and $k \in \mathbb{N}$. A clique/independent k -blowup of v is the operation of adding k twins of v which form a clique/independent set. We call an operation simply a blowup if it is either a clique k -blowup or an independent k -blowup for some $k \in \mathbb{N}$.*

We show that performing blowups cannot increase the mim-width by more than 1. Note in the following lemma that we consider a series of blowups performed at once instead of a single blowup.



■ **Figure 2** A depth-5 grid of cliques implant at $\{x_1, \dots, x_5\}$. Shaded regions indicate cliques.

► **Lemma 8** (♣). *Let G be a graph, and let Λ be a linear ordering of G . Let G' be obtained from G by a series of blowups. Then, there is a linear order Λ' of $V(G')$ computable in polynomial time from Λ such that $\text{mimw}(\Lambda') \leq \text{mimw}(\Lambda) + 1$.*

We define another operation that will find a similar use in the later sections.

► **Definition 9** (Depth- ℓ grid of cliques implant). *Let G be a graph, let $X = \{x_1, \dots, x_k\} \subseteq V(G)$ be a clique in G , and let $\ell \in \mathbb{N}$. For all $i \in [k]$, let $x_i = x_i^0$. The operation of*

- *adding, for all $i \in [\ell]$, vertices x_1^i, \dots, x_k^i ,*
- *for each $i \in [\ell]$, making $\{x_1^i, \dots, x_k^i\} = X^i$ a clique (called the i -th column), and*
- *for each $j \in [k]$, making $\{x_j^0, \dots, x_j^\ell\} = Y_j$ a clique (called the j -th row),*

is called a depth- ℓ grid of cliques implant (at X in G).

For an illustration of the previous operation see Figure 2.

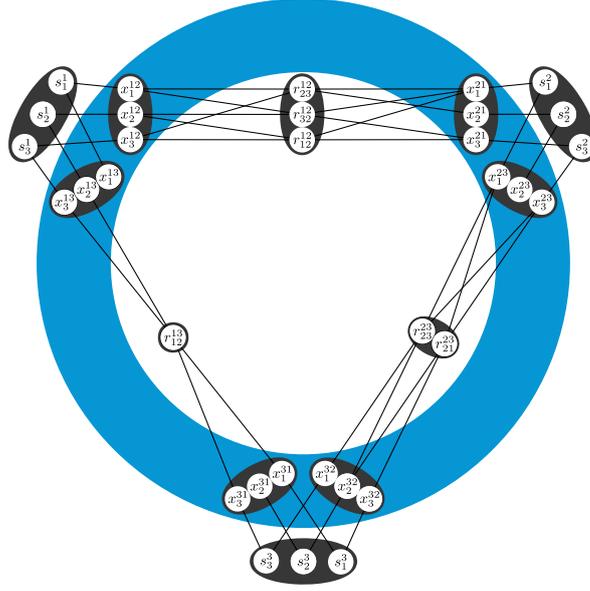
► **Lemma 10** (♣). *Let G be a graph, let Λ be a linear ordering of G . Let G' be obtained from G by a depth- ℓ grid of cliques implant. There exists a linear ordering Λ' of G' computable in polynomial time from Λ , such that $\text{mimw}(\Lambda') \leq \text{mimw}(\Lambda) + \ell$.*

4 Hardness of (σ, ρ) -Dominating Set problems

In this section we discuss the main results of this work, which are the hardness results for non-trivial MIN/MAX (σ, ρ) -DOMINATING SET problems. Note that the cases that are not covered by the following theorem ($0 \in \rho$ for minimization and $\rho = \{0\}$ for maximization) have been observed to be trivial in Observation 5.

► **Theorem 11.** *Let $\sigma, \rho \subseteq \mathbb{N}$ be nonempty where $0 \notin \rho$. Then, the MIN (σ, ρ) -DOMINATING SET[LMIM + SOL] problem is W[1]-hard. Moreover, unless the ETH is false, it cannot be solved in $f(w)n^{o(w/\log w)}$ time, where f is any computable function, on n -vertex graphs given with a linear ordering of mim-width w .*

Furthermore, if σ and ρ are nonempty, finite, and $\rho \neq \{0\}$, then the MAX (σ, ρ) -DOMINATING SET[LMIM + SOL] problem is W[1]-hard, and cannot be solved in $f(w)n^{o(w/\log w)}$ time, where f , n , and w are as above, unless the ETH is false.



■ **Figure 3** Example of \mathcal{H} for $p = 3$, and $k = 3$, and K is the complete graph with three vertices. Colored regions indicate cliques.

The proof is by a reduction from the $W[1]$ -hard problem PARTITIONED SUBGRAPH ISOMORPHISM, where first a core graph \mathcal{H} is constructed. Afterwards the graph is modified to obtain either H_0 , H_1 , H_2 , or H_3 depending on σ and ρ in such a manner that all of the above mentioned cases are captured. These modifications use, among other things, the two operations described in Section 3.

4.1 The core graph \mathcal{H}

Let (K, G, ϕ) be an instance of the PARTITIONED SUBGRAPH ISOMORPHISM problem. Recall that for the sake of our reduction, we can assume that K is connected. Throughout, we assume that $V(K) = \{1, \dots, k\}$, and that (V_1, \dots, V_k) is the partition of $V(G)$ according to ϕ , that is, for all $i \in [k]$, $V_i = \{v \in V(G) \mid \phi(v) = i\}$.

We describe how to construct from it the above mentioned core graph \mathcal{H} . We can assume that $|V_i| = p$, for all $i \in [k]$, where $p = \max\{|V_i| \mid i \in [k]\}$. If this is not the case then we can simply add isolated vertices to the sets whose cardinality is less than p . Isolated vertices clearly do not affect the PARTITIONED SUBGRAPH ISOMORPHISM instance, as K has no isolated vertices as we assumed it was connected. For all $i \in [k]$, we let $V_i = \{v_1^i, \dots, v_p^i\}$. The core graph \mathcal{H} is constructed as follows:

1. For all $i, j \in [k]$ such that $ij \in E(K)$, and for all $a \in [p]$, we add the vertex x_a^{ij} to $V(\mathcal{H})$. We let $X^{ij} = \{x_a^{ij} \mid a \in [p]\}$.
2. For all $i, j \in [k]$ such that $ij \in E(K)$ and for all $a, b \in [p]$ such that $v_a^i v_b^j \in E(G)$, we add the vertex $r_{ab}^{ij} = r_{ba}^{ji}$ to $V(\mathcal{H})$. We connect r_{ab}^{ij} to all the vertices in $\{x_{a'}^{ij} \mid a' \neq a, a' \in [p]\}$, and all the vertices in $\{x_{b'}^{ji} \mid b' \neq b, b' \in [p]\}$. We let $R^{ij} = \{r_{ab}^{ij} \mid v_a^i v_b^j \in E(G)\} = R^{ji}$.
3. For all $i \in [k]$ and for all $a \in [p]$ we add the vertex s_a^i to $V(\mathcal{H})$. Furthermore for all $j \in [k]$ such that $ij \in E(K)$, and all $a \in [p]$ we connect s_a^i to x_a^{ij} . We let $S^i = \{s_a^i \mid a \in [p]\}$.
4. For all $i \in [k]$, we make S^i a clique. For all $ij \in E(K)$, we make R^{ij} a clique. We let $X = \bigcup_{ij \in E(K)} X^{ij}$ and make X a clique.

See Figure 3 for an illustration. Notice that $R^{ij} = R^{ji}$ but $X^{ij} \neq X^{ji}$ for all $ij \in E(K)$.

The notation \mathcal{I} , \mathcal{J} , Z^α , and z_β^α . As S^i and R^{ij} have many similar properties in our reduction, we use the following slight abuse of notation. We let $\mathcal{I} = [k] \cup \{ij \mid ij \in E(K)\}$, and for $\alpha \in \mathcal{I}$, we let Z^α be S^i if $\alpha = i$ for some $i \in [k]$, and we let Z^α be R^{ij} if $\alpha = ij$ for some $ij \in E(K)$. We let $\mathcal{J} = [p] \cup [p] \times [p]$. For a pair $\alpha \in \mathcal{I}$, $\beta \in \mathcal{J}$, the vertex z_β^α is s_a^i if $\alpha \in [k]$ and $\beta \in [p]$ and r_{ab}^{ij} if $\alpha \in E(K)$ and $\beta \in [p] \times [p]$. Note that for the case $\alpha \in [k]$ and $\beta \in [p] \times [p]$ (or $\alpha \in E(K)$ and $\beta \in [p]$), the vertex z_β^α is not defined.

As outlined above, it is essential that the core graph has bounded linear mim-width. We sketch how to obtain a linear order whose mim-width is linear in the number of edges in K .

▷ **Claim 12.** There is a linear order Λ of $V(\mathcal{H})$ computable in polynomial time such that $\text{mimw}(\Lambda) \leq 4|E(K)| + 4$.

Proof (sketch). Consider the following partition of $V(\mathcal{H})$: Let $\Gamma = \{\Gamma_\alpha \mid \alpha \in \mathcal{I}\}$, where

- for all $i \in [k]$, $\Gamma_i = S^i \cup \bigcup_{ij \in E(K)} X^{ij}$ and
- for all $ij \in E(K)$, $\Gamma_{ij} = R^{ij}$.

We give the vertices in Γ_i the ordering

$$\Lambda_i : s_1^i < x_1^{i1} < x_1^{i2} < \dots < x_1^{ik} < s_2^i < x_2^{i1} < \dots < x_2^{ik} < \dots < s_p^i < x_p^{i1} < \dots < x_p^{ik},$$

and we give the vertices in Γ_{ij} any linear ordering Λ_{ij} . As $\Gamma_{ij} = R^{ij}$ is a clique, any linear ordering of Γ_{ij} has mim-width 1. Furthermore, one can show that the mim-width of Λ_i , for each $i \in [k]$, is at most 3. Considering the subgraph \mathcal{H}' of \mathcal{H} obtained by removing all edges between X^{ij} and $X^{i'j'}$ for distinct $ij, i'j' \in E(K)$, we can prove that $\text{cutmim}_{\mathcal{H}'}[\Gamma_\alpha, \Gamma_{\alpha'}] \leq 2$ for any distinct $\alpha, \alpha' \in \mathcal{I}$. The number of edges in \mathcal{H}'/Γ is equal to $2|E(K)|$, so by Lemma 6 we can obtain a linear order of the vertices of \mathcal{H}' whose mim-width is at most $4|E(K)| + 3$. (Take any linear order of $V(\mathcal{H}')$ that respects each Λ_α , $\alpha \in \mathcal{I}$.) To get the bound for \mathcal{H} , note that turning a set of vertices into a clique can increase the mim-width of any cut by at most 1. ◁

4.2 Minimization problems

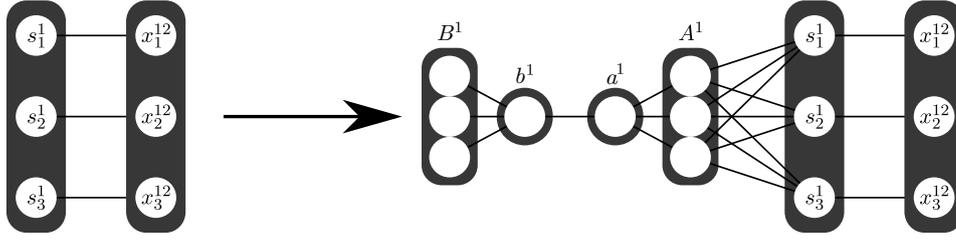
We now turn to the case when we want to show hardness for a $\text{MIN}(\sigma, \rho)$ -DOMINATING SET problem, and describe how the core graph \mathcal{H} will be enhanced/transformed to give the graph of the resulting instance. This construction crucially depends on the minimum values of σ and ρ . Therefore, we let $\varsigma = \min(\sigma)$ and $\varrho = \min(\rho)$. Note that $\varsigma + \varrho = O(1)$. In each of the cases, we show how the graph of the resulting instance is constructed, and give the budget. We state three claims, one regarding the linear mim-width of the constructed graph, and two claims that assert the correctness of the reduction. We exemplify these proofs in Section 4.2.2, where all arguments for the case treated there are given. The remaining proofs are deferred to the full version.

4.2.1 When $\varrho = \varsigma + 1$ and $\varsigma \geq 1$

We transform \mathcal{H} into the graph solution size pair (H_0, k_0) , where

$$k_0 = (2\varsigma + 2)(k + |E(K)|) + (\varsigma + 1)$$

and H_0 is constructed as follows. Recall that Z^α is either S^i when $\alpha = i \in [k]$, or R^{ij} when $\alpha = ij \in E(K)$.



■ **Figure 4** Example of how $S^1 \cup X^{12}$ is transformed for minimization problems when $p = 3$, $\varsigma = 3$, $\varrho = 4$. Circles indicate vertices, and grey colored regions indicate cliques.

1. For all $\alpha \in \mathcal{I}$, we create two cliques A^α and B^α which both have size ς , where A^α is adjacent to all of Z^α .
2. We add two adjacent vertices a^α and b^α . The vertex a^α is adjacent to all vertices in A^α , and b^α is adjacent to all vertices in B^α .
3. We add a clique \mathcal{X} of size $\varsigma + 1$ which is partitioned in two parts: \mathcal{X}_1 and \mathcal{X}_2 , where $|\mathcal{X}_2| = 1$. Every vertex in \mathcal{X}_1 is adjacent to all the vertices in X , and every vertex in \mathcal{X}_2 is only adjacent to the vertices in \mathcal{X} .

For an illustration see Figure 4.

▷ **Claim 13 (♣)**. There is a linear order Λ_0 of $V(H_0)$ computable in polynomial time such that $\text{mimw}(\Lambda_0) = O(|E(K)|)$.

▷ **Claim 14 (♣)**. If (K, G, ϕ) is a YES-instance of the PARTITIONED SUBGRAPH ISOMORPHISM problem, then there exists a $(\{\varsigma\}, \{\varrho\})$ -dominating set of size k_0 in H_0 .

▷ **Claim 15 (♣)**. If there exists a (σ, ρ) -dominating set of size at most k_0 in H_0 , then (K, G, ϕ) is a YES-instance of the PARTITIONED SUBGRAPH ISOMORPHISM problem.

4.2.2 When $\varrho > \varsigma + 1$ and $\varsigma \geq 1$

Let $\varrho' = \varrho - \varsigma$. In this case, we create the graph solution size pair (H_1, k_1) , where

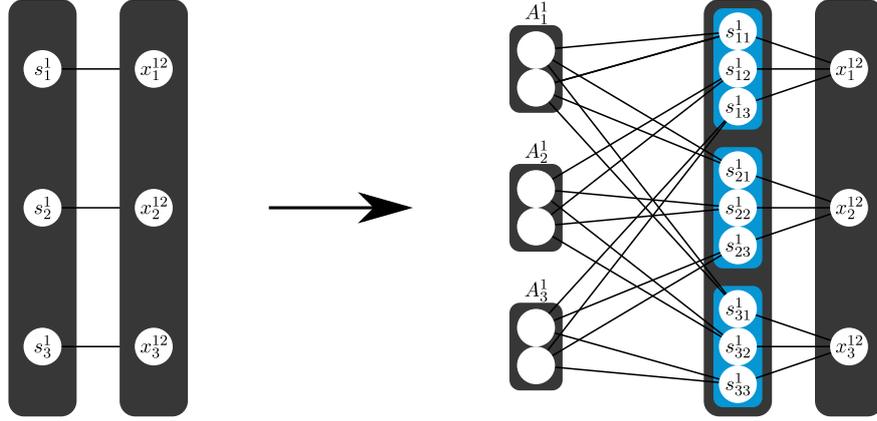
$$k_1 = (\varrho'\varsigma + \varrho')(k + |E(K)|) + (\varsigma + 1),$$

and H_1 is obtained from \mathcal{H} as follows. Recall the operation of a blowup, Definition 7, and see Figure 5 for an illustration of the following.

1. For each $\alpha \in \mathcal{I}$ and $\beta \in \mathcal{J}$ such that $z_\beta^\alpha \in V(\mathcal{H})$, we perform an independent $(\varrho' - 1)$ -blowup of z_β^α . We call the twins of z_β^α : $z_{\beta 2}^\alpha, \dots, z_{\beta \varrho'}^\alpha$, and we let $z_\beta^\alpha = z_{\beta 1}^\alpha$.
2. For each $\alpha \in \mathcal{I}$ and $\ell \in [\varrho']$, we add a clique A_ℓ^α of size ς , where every vertex in A_ℓ^α is adjacent to every vertex in $Z_{*\ell}^\alpha = \{z_{\beta \ell}^\alpha \mid \beta \in \mathcal{J} \text{ s.t. } z_\beta^\alpha \in V(\mathcal{H})\}$. We let $A^\alpha = \bigcup_{\ell \in [\varrho']} A_\ell^\alpha$.
3. We add a clique \mathcal{X} of size $\varsigma + 1$ to H_1 ; this clique is partitioned into two parts \mathcal{X}_1 and \mathcal{X}_2 , where $|\mathcal{X}_2| = 1$. Every vertex in \mathcal{X}_1 is adjacent to all vertices in X , and the vertex in \mathcal{X}_2 is only adjacent to \mathcal{X} .

We use the following notation. We call the set containing z_β^α with its $\varrho' - 1$ twins $Z_{\beta*}^\alpha = \{z_{\beta \ell}^\alpha \mid \ell \in [\varrho']\}$, and we let $Z_{**}^\alpha = \bigcup_{\ell \in [\varrho']} Z_{*\ell}^\alpha$. Note that the vertices in $Z_{\beta*}^\alpha$ are not adjacent to any other vertex in $Z_{\beta*}^\alpha$, however they are all adjacent to $Z_{\beta'*}^\alpha$ for all $\beta' \neq \beta$.

▷ **Claim 16**. There is a linear order Λ_1 of $V(H_1)$ computable in polynomial time such that $\text{mimw}(\Lambda_1) = O(|E(K)|)$.



■ **Figure 5** Example of the modification of $S^1 \cup X^{12}$ for $p = 3$, $\zeta = 2$, $\varrho = 5$. Circles indicate vertices, and grey colored regions indicate cliques. The blue regions indicate independent sets.

Proof. Let Λ be a linear order of \mathcal{H} of mim-width $O(|E(K)|)$ obtained from Claim 12 in polynomial time. H_1 is constructed from \mathcal{H} by blowing up all vertices $z_\beta^\alpha \in V(\mathcal{H})$, where $\alpha \in \mathcal{I}$ and $\beta \in \mathcal{J}$, and adding $|E(K)| + k + 1 = O(|E(K)|)$ vertex sets of constant size. We can place the latter vertices anywhere in the ordering Λ without increasing the mim-width by more than $O(|E(K)|)$, call the resulting ordering Λ' . From Λ' we can obtain a linear order of H_1 in polynomial time whose mim-width is at most one larger using Lemma 8. \triangleleft

We now show the correctness of the reduction in the following two claims.

▷ **Claim 17.** If (K, G, ϕ) is a YES-instance of the PARTITIONED SUBGRAPH ISOMORPHISM problem, then there exists a $(\{\zeta\}, \{\varrho\})$ -dominating set of size k_1 in H_1 .

Proof. Let $f : V(K) \rightarrow V(G)$ be the injective function preserving neighbors and colors. Let $f(i) = v_{c_i}^i$ for all $i \in [k]$ and for some $c_1, \dots, c_k \in [p]$. Note that $ij \in E(K)$ implies that $v_{c_i}^i v_{c_j}^j \in E(G)$ further implying that $r_{c_i c_j}^{ij} \in V(\mathcal{H}) \subseteq V(H_1)$. We argue that

$$D = \mathcal{X} \cup \bigcup_{i \in [k]} S_{c_i^*}^i \cup \bigcup_{ij \in E(K)} R_{c_i c_j^*}^{ij} \cup \bigcup_{\alpha \in \mathcal{I}} A^\alpha$$

is a $(\{\zeta\}, \{\varrho\})$ -dominating set of size k_1 in H_1 . First, we observe that

$$|D| = \zeta + 1 + k \cdot \varrho' + |E(K)| \cdot \varrho' + (k + |E(K)|)\zeta\varrho' = k_1.$$

The sets \mathcal{X} , $\bigcup_{\alpha \in \mathcal{I}} Z_{**}^\alpha$, $\bigcup_{\alpha \in \mathcal{I}} A^\alpha$, $\bigcup_{ij \in E(K)} X^{ij}$ form a partition of $V(H_1)$. First consider any vertex x in $\mathcal{X} \subseteq D$, and recall that \mathcal{X} is a clique of size $\zeta + 1$. If x is the unique vertex in \mathcal{X}_2 , then $N(x) = \mathcal{X}_1 \subseteq D$, so x has ζ neighbors in D . If $x \in \mathcal{X}_1$, then $N(x) = X \cup \mathcal{X} \setminus \{x\}$, and since $X \cap D = \emptyset$, we have that x has ζ neighbors in D as well.

Next, consider a vertex z in $\bigcup_{\alpha \in \mathcal{I}} Z_{**}^\alpha$. There are two cases. In the first case, $\alpha = i \in [k]$, and $z = s_{j\ell}^i$ for some $j \in [p]$ and $\ell \in [\varrho']$. If $j = c_i$, then $z = s_{c_i\ell}^i \in D$, and $N(s_{c_i\ell}^i) \cap D = A_\ell^i$, and therefore $|N(z) \cap D| = \zeta$. (Note that $s_{c_i\ell}^i$ is not adjacent to any vertex in $S_{c_i^*}^i \subseteq D$.) If $j \neq c_i$, then $N(z) \cap D = A_\ell^i \cup S_{c_i}^i$, so $|N(z) \cap D| = \zeta + \varrho' = \varrho$. The second case, when $\alpha = ij \in E(K)$, can be argued in the same way.

Now let a be a vertex in $A^\alpha \subseteq D$, for some $\alpha \in \mathcal{I}$, and assume that $\alpha = i \in [k]$. Then we have that $a \in A_\ell^i$ for some $\ell \in [\varrho']$, and the intersection of $N(a)$ with D consists of $A_\ell^i \setminus \{a\}$, and the vertex $s_{c_i\ell}^i$. We conclude that $|N(a) \cap D| = \zeta$; the case when $\alpha = ij \in E(K)$ is the same.

Finally, consider some vertex in X^{ij} , where $ij \in E(K)$, in particular such a vertex is x_a^{ij} for some $a \in [p]$. Since $D \cap X = \emptyset$, we have that $x_a^{ij} \notin D$. Furthermore, $N(x_a^{ij})$ contains $\mathcal{X}_1 \subseteq D$. Now, suppose that $a = c_i$. Then, $S_{c_i}^i$ is also in $N(x_a^{ij}) \cap D$. The only other neighbors of vertices in X^{ij} that are not in $X \cup S_{**}^i \cup \mathcal{X}$ are in R_{**}^{ij} . However, the only vertices in $D \cap R_{**}^{ij}$ are in $R_{c_i c_j}^{ij}$, and by construction, $x_{c_i}^{ij}$ is not adjacent to any of them. Therefore, $x_{c_i}^{ij}$ has $\varsigma + \varrho' = \varrho$ neighbors in D . If $a \neq c_i$, the argument is similar, but with the roles of S_a^i and $R_{c_i c_j}^{ij}$ exchanged. \triangleleft

\triangleright **Claim 18.** If there exists a (σ, ρ) -dominating set of size at most k_1 in H_1 , then (K, G, ϕ) is a YES-instance of the PARTITIONED SUBGRAPH ISOMORPHISM problem.

Proof. Let $D \subseteq V(H_1)$ be the (σ, ρ) -dominating set of size at most k_1 in H_1 . We show that for all $\alpha \in \mathcal{I}$ there is some $\beta \in \mathcal{J}$ such that $Z_{\beta}^\alpha \subseteq D$. From these pairs (α, β) , we will then derive a solution to (K, G, ϕ) . Recall that $|Z_{\beta}^\alpha| = \varrho'$ for all such α, β , so as a first step we show that

$$\text{for all } \alpha \in \mathcal{I}, |Z_{**}^\alpha \cap D| = \varrho'. \quad (1)$$

Let $\alpha \in \mathcal{I}$. We first show that $|(Z_{**}^\alpha \cup A^\alpha) \cap D| = \varrho'(\varsigma + 1)$, and narrow down to prove (1) afterwards. Towards this, we argue that $|(Z_{**}^\alpha \cup A^\alpha) \cap D| \geq \varrho'(\varsigma + 1)$. Observe that

$$\text{for all } \ell \in [\varrho'], A_\ell^\alpha \subseteq D \text{ or } |N[A_\ell^\alpha] \cap D| \geq \varrho. \quad (2)$$

Indeed, for all $v \in A_\ell^\alpha$, either $v \in D$, or $|N(v) \cap D| \geq \varrho$. This in turn means that either $A^\alpha \subseteq D$ or that there is some $\ell \in [\varrho']$, such that $|N[A_\ell^\alpha] \cap D| \geq \varrho$. Let a be the number of $\ell \in [\varrho']$ such that $A_\ell^\alpha \not\subseteq D$. Since $|A_\ell^\alpha| = \varsigma$ and for distinct $\ell, \ell' \in [\varrho']$, $N[A_\ell^\alpha] \cap N[A_{\ell'}^\alpha] = \emptyset$, this implies together with (2) that

$$|(Z_{**}^\alpha \cup A^\alpha) \cap D| \geq (\varrho' - a)\varsigma + a\varrho = \varrho'(\varsigma + a). \quad (3)$$

Now, if $a \geq 1$, then we can conclude immediately that $|(Z_{**}^\alpha \cup A^\alpha) \cap D| \geq \varrho'(\varsigma + 1)$, so suppose $a = 0$. Then, by (2), $A_\ell^\alpha \subseteq D$ for all $\ell \in [\varrho']$. However, $|A_\ell^\alpha| = \varsigma$, so there has to be at least one more vertex in $N(A_\ell^\alpha) \cap D$. Since $N(A_\ell^\alpha) \subseteq Z_{**}^\alpha$, and for distinct $\ell, \ell' \in [\varrho']$ $N(A_\ell^\alpha) \cap N(A_{\ell'}^\alpha) = \emptyset$, it follows that D has to contain at least another ϱ' vertices from Z_{**}^α ; therefore, also when $a = 0$, we have that $|(Z_{**}^\alpha \cup A^\alpha) \cap D| \geq \varrho'(\varsigma + 1)$.

We show that by the choice of k_1 , the inequality we just argued is an equality. To do so, consider \mathcal{X} . Since there is a vertex in \mathcal{X} whose degree is ς , and since $\varsigma < \varrho$, we conclude that $\mathcal{X} \subseteq D$. We have argued that

$$|D| \geq \varsigma + 1 + |\mathcal{I}| \cdot \varrho' \cdot (\varsigma + 1) = \varsigma + 1 + (k + |E(K)|)(\varrho'\varsigma + \varrho') = k_1,$$

and since $|D| \leq k_1$ by assumption, we have that $|D| = k_1$ and

$$\text{for all } \alpha \in \mathcal{I}, |(Z_{**}^\alpha \cup A^\alpha) \cap D| = \varrho'(\varsigma + 1). \quad (4)$$

Note that since the vertices considered so far already use up all the budget, we also have $X \cap D = \emptyset$.

As a last step, we argue that $A^\alpha \subseteq D$, which together with (4) implies (1). (Recall that $|A^\alpha| = \varrho'\varsigma$.) In other words, we want to show that $a = 0$. If $a > 1$, then by (3) we get a contradiction with (4). So suppose that $a = 1$, and let $\ell \in [\varrho']$ be such that $A_\ell^\alpha \not\subseteq D$. For each $\ell' \in [\varrho'] \setminus \{\ell\}$, $A_{\ell'}^\alpha \subseteq D$; and since $|A_{\ell'}^\alpha| = \varsigma$, there is at least one more vertex in

3:14 Hardness of Domination Problems Parameterized by Mim-Width

$N(A_{\ell'}^{\alpha}) \cap D$. Similar to above, this allows us to conclude that $|D'| \geq (\varrho' - 1)(\varsigma + 1)$, where $D' = D \cap \bigcup_{\ell' \in [\varrho'] \setminus \{\ell\}} N[A_{\ell'}^{\alpha}]$. By (2), we have that $|N[A_{\ell}^{\alpha}] \cap D| \geq \varrho$, and by construction $N[A_{\ell}^{\alpha}] \cap D' = \emptyset$. Together with (4) this means that

$$\varrho'(\varsigma + 1) = |(Z_{**}^{\alpha} \cup A^{\alpha}) \cap D| \geq (\varrho' - 1)(\varsigma + 1) + \varrho = \varrho'(\varsigma + 1) + \varrho' - 1,$$

which only holds if $\varrho' \leq 1$. However, $\varrho > \varsigma + 1$, so $\varrho' = \varrho - \varsigma > 1$, a contradiction. We have argued that $a = 0$, and therefore $A^{\alpha} \subseteq D$, proving (1) due to (4).

Now that we know that $|Z_{**}^{\alpha} \cap D| = \varrho'$, it remains to show that there is some $\beta \in \mathcal{J}$ such that $Z_{\beta*}^{\alpha} \subseteq D$. Suppose not, then there exists some $\gamma \in \mathcal{J}$ such that $1 \leq |Z_{\gamma*}^{\alpha} \cap D| < \varrho'$. Let $z_{\gamma\ell}^{\alpha} \notin D$ where $\ell \in [\varrho']$. The neighborhood of $z_{\gamma\ell}^{\alpha}$ is contained in A_{ℓ}^{α} , Z_{**}^{α} , and X . So, $z_{\gamma\ell}^{\alpha}$ has ς neighbors in $D \cap A_{\ell}^{\alpha}$, no neighbors in $D \cap X$ (recall that $X \cap D = \emptyset$), and at most $\varrho' - 1$ neighbors in $D \cap Z_{**}^{\alpha}$. The latter is due to the fact that $Z_{\gamma*}^{\alpha}$ contains at least one vertex from D , and the fact that $Z_{\gamma*}^{\alpha}$ is an independent set. So $|N(z_{\gamma\ell}^{\alpha}) \cap D| \leq \varsigma + \varrho' - 1 = \varrho - 1$, a contradiction with D being a (σ, ρ) -dominating set.

Then for all $i \in [k]$ there exists a $c_i \in [p]$ such that $S_{c_i*}^i \subseteq D$, and for all $ij \in E(K)$ there exists $d_i, d_j \in [p]$ such that $R_{d_i d_j*}^{ij} \subseteq D$. Suppose that $c_i \neq d_i$ then notice the vertex $x_{d_i}^{ij}$ is only being dominated by the $\varsigma < \varrho$ vertices in $\mathcal{X} \cap D$, but $\varsigma \notin \rho$. Therefore $c_i = d_i$, and by a similar argument $c_j = d_j$. We can conclude that the edges $\{v_{c_i}^i v_{c_j}^j \mid i, j \in [k]\}$ exist in G . Then the function $f : V(K) \rightarrow V(G)$ where $f(i) = v_{c_i}^i$, is a function preserving neighbors and colors. \triangleleft

4.2.3 When $\varrho < \varsigma + 1$

Let $\varsigma' = \varsigma - \varrho + 1$. In this case, we construct the graph solution size pair: (H_2, k_2) , where

$$k_2 = (\varsigma + 1) \cdot (|E(K)| + k) + \varsigma + 1.$$

The graph H_2 is obtained from \mathcal{H} by the modifications given below. Recall the operation of a depth- ℓ grid of cliques implant, see Definition 9; and for convenience, for all $\alpha \in \mathcal{I}$ and $\beta \in \mathcal{J}$ such that $z_{\beta}^{\alpha} \in V(\mathcal{H})$, let $z_{\beta}^{\alpha} = z_{\beta 0}^{\alpha}$.

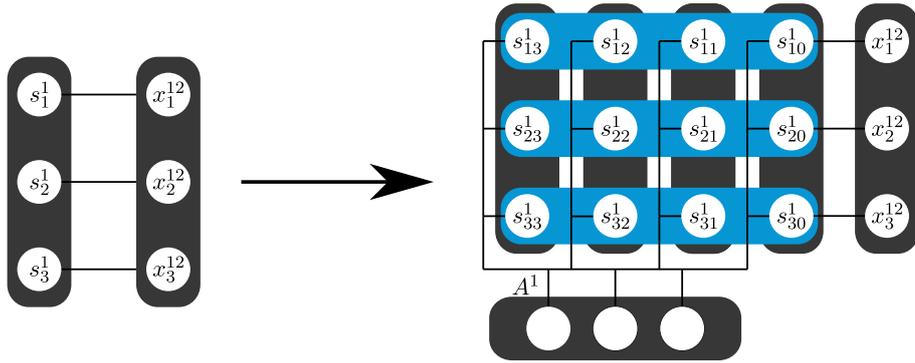
1. For each $\alpha \in \mathcal{I}$, we perform a depth- ς' grid of cliques implant at Z^{α} . We call the ℓ -th column $Z_{*\ell}^{\alpha}$, for all $\ell \in [\varsigma']_0$, and the β -th row $Z_{\beta*}^{\alpha}$, for all β such that $z_{\beta}^{\alpha} \in Z^{\alpha}$. Let $Z_{**}^{\alpha} = \bigcup_{\ell \in [\varsigma']_0} Z_{*\ell}^{\alpha}$.
2. For each $\alpha \in \mathcal{I}$: If $\varrho > 1$, then we add a clique A^{α} of size $\varrho - 1$, where the vertices in A^{α} are adjacent to all vertices in Z_{**}^{α} . If $\varrho = 1$ then $A^{\alpha} = \emptyset$.
3. We add a clique \mathcal{X} of size $\varsigma + 1$ to H_2 . This clique is partitioned into two parts \mathcal{X}_1 and \mathcal{X}_2 , where \mathcal{X}_1 has size $\varrho - 1$ and all its vertices are adjacent to all vertices in X . The vertices in \mathcal{X}_2 are only adjacent to all all vertices in \mathcal{X} and \mathcal{X}_2 has size $\varsigma' + 1$.²

▷ **Claim 19 (♣).** There is a linear order Λ_2 of $V(H_2)$ computable in polynomial time such that $\text{mimw}(\Lambda_2) = O(|E(K)|)$.

▷ **Claim 20 (♣).** If (K, G, ϕ) is a YES-instance of the PARTITIONED SUBGRAPH ISOMORPHISM problem, then there exists a $(\{\varsigma\}, \{\varrho\})$ -dominating set of size k_2 in H_2 .

▷ **Claim 21 (♣).** If there exists a (σ, ρ) -dominating set of size at most k_2 in H_2 , then (K, G, ϕ) is a YES-instance of the PARTITIONED SUBGRAPH ISOMORPHISM problem.

² The set \mathcal{X} is not needed for correctness when $\varsigma = 0$, but for simplicity we include it anyway.



■ **Figure 6** Example of the modifications to $S^1 \cup X^{12}$ for $p = 3, \varsigma = 6, \varrho = 4$.

4.2.4 When $\varrho \geq 1$ and $\varsigma = 0$

In this case, we construct the graph solution size pair (H_3, k_3) , where $k_3 = \varrho(k + |E(K)|)$, and H_3 is constructed from \mathcal{H} follows.

1. For each $\alpha \in \mathcal{I}, \beta \in \mathcal{J}$ such that $z_\beta^\alpha \in V(\mathcal{H})$, we perform an independent $(\varrho - 1)$ -blowup of z_β^α .³ We call the twins of z_β^α : $z_{\beta 2}^\alpha, \dots, z_{\beta \varrho}^\alpha$, and we let $z_\beta^\alpha = z_{\beta 1}^\alpha$. We let $Z_{\beta*}^\alpha = \{z_{\beta \ell}^\alpha \mid \ell \in [\varrho]\}, Z_{*\ell}^\alpha = \{z_{\beta \ell}^\alpha \mid \beta \in \mathcal{J} \text{ s.t. } z_\beta^\alpha \in V(\mathcal{H})\}$, and $Z_{**}^\alpha = Z^\alpha \cup \bigcup_{\ell \in [\varrho]} Z_{*\ell}^\alpha$.
2. For all $\alpha \in \mathcal{I}$, we add a clique A^α of size ϱ , and we connect all of its vertices to to all the vertices in Z_{**}^α .

▷ **Claim 22 (♣).** There is a linear order Λ_3 of $V(H_3)$ computable in polynomial time such that $\text{mimw}(\Lambda_3) = O(|E(K)|)$.

▷ **Claim 23 (♣).** If (K, G, ϕ) is a YES-instance of the PARTITIONED SUBGRAPH ISOMORPHISM problem, then there exists a $(\{\varsigma\}, \{\varrho\})$ -dominating set of size k_3 in H_3 .

▷ **Claim 24 (♣).** If there exists a (σ, ρ) -dominating set of size at most k_3 in H_3 , then (K, G, ϕ) is a YES-instance of the PARTITIONED SUBGRAPH ISOMORPHISM problem.

4.3 Maximization problems

For maximization problems, we can reuse the constructions as in the previous section; however we let $\varsigma = \max(\sigma)$ and $\varrho = \max(\rho)$ (instead of taking the minima of σ and ρ). This is why we require σ and ρ to be finite.

When $\varsigma < \varrho$, we construct (H_1, k_1) as in Section 4.2.2. Therefore the mim-width bound follows by the same arguments, and one direction of the correctness proof is already shown in Claim 17. A bit of attention is necessary in case $\varrho' = 0$, but the arguments still work after some minor tweaks. In case $\varsigma \geq \varrho$, we construct (H_2, k_2) as in Section 4.2.3. Again, the mim-width bound and one direction of the correctness proof (Claim 20) are already taken care of. The remaining proofs and other details are given in the full version.

³ If $\varrho = 1$ then this step is skipped.

5 Conclusion

In this work, we proved that each $\text{MIN } (\sigma, \rho)\text{-DOMINATING SET}$ problem is either polynomial-time solvable or $\text{W}[1]$ -hard parameterized by the mim-width of a given linear branch decomposition of the input graph plus solution size, and that the same holds for $\text{MAX } (\sigma, \rho)\text{-DOMINATING SET}$ problems whenever σ and ρ are finite. An immediate open question is whether we can complete the dichotomy for maximization problems to the cases when σ and/or ρ are infinite.

► **Open Problem 1.** *Is it true that for all $\sigma, \rho \subseteq \mathbb{N}$, including infinite sets, $\text{MAX } (\sigma, \rho)\text{-DOMINATING SET}$ is either polynomial-time solvable or $\text{W}[1]$ -hard when parameterized by the mim-width of a given linear branch decomposition of the input graph?*

For all the $\text{W}[1]$ -hard cases, our reductions also ruled out $f(w)n^{o(w/\log w)}$ -time algorithms under the ETH, for any computable f , where n is the number of vertices of the input graph and w the mim-width of the given linear branch decomposition. Since the algorithms for finite and co-finite $\text{MIN/MAX } (\sigma, \rho)\text{-DOMINATING SET}$ problems run in $n^{O(w)}$ time [13], it is a natural question to close this gap.

► **Open Problem 2.** *Are there finite or co-finite sets $\sigma, \rho \subseteq \mathbb{N}$ such that an algorithm for the $\text{MIN/MAX } (\sigma, \rho)\text{-DOMINATING SET}$ problem that is $\text{W}[1]$ -hard parameterized by the mim-width w of a given (linear) branch decomposition of the input n -vertex graph, running in $n^{o(w)}$ time, would refute the ETH?*

In this work, we only considered minimization and maximization variants of $(\sigma, \rho)\text{-DOMINATING SET}$ problems. A third variant, say the $\text{EXACT } (\sigma, \rho)\text{-DOMINATING SET}$ problem, asks for a (σ, ρ) -dominating set of size *exactly* k . While all hardness proofs given in this work also work for $\text{EXACT } (\sigma, \rho)\text{-DOMINATING SET}$ problems, these problems are *not* trivial to solve when $0 \in \rho$, as the empty set is not a solution in this case (unless, of course, $k = 0$). We therefore ask the following question, and remark that the analogous question parameterized by solution size was asked by Golovach et al. [23].

► **Open Problem 3.** *Are there some (finite or co-finite) $\sigma, \rho \subseteq \mathbb{N}$ with $0 \in \rho$ such that $\text{EXACT } (\sigma, \rho)\text{-DOMINATING SET}$ parameterized by the mim-width of a given (linear) branch decomposition is $\text{W}[1]$ -hard?*

In a recent work [17], Eiben et al. introduced a framework of width measures based on branch decompositions over the vertex set. There, given a family \mathcal{F} of bipartite graphs, the value of a cut is determined as the largest graph in \mathcal{F} that appears as a semi-induced subgraph across the cut. Mim-width is an instantiation of this framework where \mathcal{F} is the family of matchings. Our hardness proofs greatly rely on the fact that mim-width is not closed under taking the complement of the graph. It would be interesting to see what happens to the complexity of the problems in this work when one considers the width measure obtained by letting \mathcal{F} be the union of the family of matchings *and anti-matchings* as the parameter, which results in a parameter related to mim-width that is closed under the complement.

Lastly, we want to point out that we cannot expect to prove $\text{W}[1]$ -completeness for the $\text{W}[1]$ -hard cases of $\text{MIN/MAX } (\sigma, \rho)\text{-DOMINATING SET}$ parameterized by linear mim-width considered in this work. In a recent work, Bodlaender et al. [7] showed that the $\text{MINIMUM DOMINATING SET}$ and $\text{MAXIMUM INDEPENDENT SET}$ problems parameterized by the mim-width of a given linear branch decomposition of the input graph are XNLP -complete [8]. This in turn implies that these problems are $\text{W}[t]$ -hard for all t , which makes containment in

W[1] unlikely. Furthermore, we believe that the ideas used in our work and those from [7] can be combined to show that all W[1]-hard cases from our work are indeed XNLP-hard. Membership in XNLP can be derived for all finite or co-finite σ and ρ , in a similar way as it is done for MAXIMUM INDEPENDENT SET and MINIMUM DOMINATING SET in [7].

References

- 1 Ranendu Adhikary, Kaustav Bose, Satwik Mukherjee, and Bodhayan Roy. Complexity of maximum cut on interval graphs. In Kevin Buchin and Éric Colin de Verdière, editors, *Proceedings of the 37th International Symposium on Computational Geometry (SoCG 2021)*, volume 189 of *LIPIcs*, pages 7:1–7:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.SocG.2021.7.
- 2 Brage I. K. Bakkane. Hardness of non-trivial generalized domination problems parameterized by linear mim-width. Master’s thesis, University of Bergen, Norway, 2022.
- 3 Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoretical Computer Science*, 511:54–65, 2013. doi:10.1016/j.tcs.2013.01.011.
- 4 Benjamin Bergougnoux, Jan Dreier, and Lars Jaffke. A logic-based algorithmic meta-theorem for mim-width. *CoRR*, abs/2022.13335, 2022.
- 5 Benjamin Bergougnoux and Mamadou Moustapha Kanté. More applications of the d -neighbourhood equivalence: Acyclicity and connectivity constraints. *SIAM Journal on Discrete Mathematics*, 35(3):1881–1926, 2021. doi:10.1137/20M1350571.
- 6 Benjamin Bergougnoux, Charis Papadopoulos, and Jan Arne Telle. Node multiway cut and subset feedback vertex set on graphs of bounded mim-width. *Algorithmica*, 84(5):1385–1417, 2022. doi:10.1007/s00453-022-00936-w.
- 7 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Lars Jaffke, and Paloma T. Lima. XNLP-completeness for parameterized problems on graphs with a linear structure. *CoRR*, abs/2201.13119, 2022. To appear in the proceedings of IPEC 2022. arXiv:2201.13119.
- 8 Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. In *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science, (FOCS 2021)*, pages 193–204. IEEE, 2021. doi:10.1109/FOCS52979.2021.00027.
- 9 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: Tractable FO model checking. *Journal of the ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.
- 10 Flavia Bonomo-Braberman, Nick Brettell, Andrea Munaro, and Daniël Paulusma. Solving problems on generalized convex graphs via mim-width. In Anna Lubiw and Mohammad R. Salavatipour, editors, *Proceedings of the 17th International Symposium on Algorithms and Data Structures (WADS 2021)*, volume 12808 of *Lecture Notes in Computer Science*, pages 200–214. Springer, 2021. doi:10.1007/978-3-030-83508-8_15.
- 11 Nick Brettell, Jake Horsfield, Andrea Munaro, Giacomo Paesani, and Daniël Paulusma. Bounding the mim-width of hereditary graph classes. *Journal of Graph Theory*, 99:117–151, 2022.
- 12 Nick Brettell, Jake Horsfield, Andrea Munaro, and Daniël Paulusma. List k -colouring P_t -free graphs: A mim-width perspective. *Information Processing Letters*, 173:106168, 2022. doi:10.1016/j.ipl.2021.106168.
- 13 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoretical Computer Science*, 511:66–76, 2013.
- 14 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.

- 15 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 5th edition, 2016.
- 16 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.
- 17 Eduard Eiben, Robert Ganian, Thekla Hamm, Lars Jaffke, and O-joung Kwon. A unifying framework for characterizing and computing width measures. In Mark Braverman, editor, *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 63:1–63:23, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2022.63.
- 18 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009. doi:10.1016/j.tcs.2008.09.065.
- 19 Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the tractability of optimization problems on H -graphs. *Algorithmica*, 82(9):2432–2473, 2020. doi:10.1007/s00453-020-00692-9.
- 20 Esther Galby, Andrea Munaro, and Bernard Ries. Semitotal domination: New hardness results and a polynomial-time algorithm for graphs of bounded mim-width. *Theoretical Computer Science*, 814:28–48, 2020.
- 21 M. R. Garey, David S. Johnson, G. L. Miller, and Christos H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic and Discrete Methods*, 1(2):216–227, 1980. doi:10.1137/0601025.
- 22 M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.
- 23 Petr A. Golovach, Jan Kratochvíl, and Ondrej Suchý. Parameterized complexity of generalized domination problems. *Discrete Applied Mathematics*, 160(6):780–792, 2012. doi:10.1016/j.dam.2010.11.012.
- 24 Martin Charles Golumbic and Udi Rotics. On the clique-width of some perfect graph classes. *International Journal of Foundations of Computer Science*, 11(3):423–443, 2000. doi:10.1142/S0129054100000260.
- 25 Carolina Lucía Gonzalez and Felix Mann. On d -stable locally checkable problems on bounded mim-width graphs. *CoRR*, abs/2203.15724, 2022. doi:10.48550/arXiv.2203.15724.
- 26 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.
- 27 Lars Jaffke. *Bounded Width Graph Classes in Parameterized Algorithms*. PhD thesis, University of Bergen, Norway, 2020.
- 28 Lars Jaffke, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. Mim-width III. Graph powers and generalized distance domination problems. *Theoretical Computer Science*, 796:216–236, 2019. doi:10.1016/j.tcs.2019.09.012.
- 29 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width I. Induced path problems. *Discrete Applied Mathematics*, 278:153–168, 2020. doi:10.1016/j.dam.2019.06.026.
- 30 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width II. The feedback vertex set problem. *Algorithmica*, 82:118–145, 2020. doi:10.1007/s00453-019-00607-3.
- 31 Dániel Marx. Can you beat treewidth? *Theory of Computing*, 6(1):85–112, 2010. doi:10.4086/toc.2010.v006a005.
- 32 Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(4):757–771, 2003. doi:10.1016/S0022-0000(03)00078-3.
- 33 Jan Arne Telle. Complexity of domination-type problems in graphs. *Nordic Journal on Computing*, 1(1):157–171, 1994.

- 34 Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k -trees. *SIAM Journal on Discrete Mathematics*, 10(4):529–550, 1997. doi: 10.1137/S0895480194275825.
- 35 Martin Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, Norway, 2012.

FPT Approximation for Fair Minimum-Load Clustering

Sayan Bandyapadhyay ✉

Department of Informatics, University of Bergen, Norway

Fedor V. Fomin ✉ 

Department of Informatics, University of Bergen, Norway

Petr A. Golovach ✉ 

Department of Informatics, University of Bergen, Norway

Nidhi Purohit ✉

Department of Informatics, University of Bergen, Norway

Kirill Simonov ✉

Algorithms and Complexity Group, TU Wien, Austria

Abstract

In this paper, we consider the Minimum-Load k -Clustering/Facility Location (MLkC) problem where we are given a set P of n points in a metric space that we have to cluster and an integer $k > 0$ that denotes the number of clusters. Additionally, we are given a set F of cluster centers in the same metric space. The goal is to select a set $C \subseteq F$ of k centers and assign each point in P to a center in C , such that the maximum *load* over all centers is minimized. Here the load of a center is the sum of the distances between it and the points assigned to it.

Although clustering/facility location problems have rich literature, the minimum-load objective has not been studied substantially, and hence MLkC has remained a poorly understood problem. More interestingly, the problem is notoriously hard even in some special cases including the one in line metrics as shown by Ahmadian et al. [APPROX 2014, ACM Trans. Algorithms 2018]. They also show APX-hardness of the problem in the plane. On the other hand, the best-known approximation factor for MLkC is $O(k)$, even in the plane.

In this work, we study a fair version of MLkC inspired by the work of Chierichetti et al. [NeurIPS, 2017]. Here the input points are partitioned into ℓ protected groups, and only clusters that proportionally represent each group are allowed. MLkC is the special case with $\ell = 1$. For the fair version, we are able to obtain a randomized 3-approximation algorithm in $f(k, \ell) \cdot n^{O(1)}$ time. Also, our scheme leads to an improved $(1 + \epsilon)$ -approximation in the case of Euclidean norm with the same running time (depending also linearly on the dimension d). Our results imply the same approximations for MLkC with running time $f(k) \cdot n^{O(1)}$, achieving the first constant-factor FPT approximations for this problem in general and Euclidean metric spaces.

2012 ACM Subject Classification Theory of computation \rightarrow Facility location and clustering; Theory of computation \rightarrow Parameterized complexity and exact algorithms

Keywords and phrases fair clustering, load balancing, parameterized approximation

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.4

Funding The research leading to these results have been supported by the Research Council of Norway via the project BWCA (grant no. 314528), European Research Council (ERC) via grant LOPPRE, reference 819416, and Austrian Science Fund (FWF) via project P31336 (New Frontiers for Parameterized Complexity).



© Sayan Bandyapadhyay, Fedor V. Fomin, Petr A. Golovach, Nidhi Purohit, and Kirill Simonov; licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 4; pp. 4:1–4:14

Leibniz International Proceedings in Informatics



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

1 Introduction

Clustering is the task of partitioning a set of data items into a number of groups (or clusters) such that each group contains similar items. Typically, the similarity in the clusters is modeled by a proxy objective function, which one needs to optimize. Being a fundamental computational problem in nature, clustering has a host of diverse applications in computer science and other disciplines. Consequently, the problem has been studied with several different and possibly independent objectives. Some of these became notably popular, for example, k -means, k -median, and k -center [22, 19, 5, 26]. In this paper, we consider an objective which has not been studied substantially in the literature. In particular, we consider minimum-load clustering. Here we are given a set P of points in a metric space that we have to cluster and an integer $k > 0$ that denotes the number of clusters. Additionally, we are given a set F of cluster centers in the same metric space. The goal is to select a set $C \subseteq F$ of k centers and assign each point in P to a center in C , such that the maximum *load* over all centers is minimized. Here the load of a center is the sum of the distances between it and the points assigned to it. That is, if P' is the set of points assigned to a center c , then its load is $\sum_{p \in P'} d(c, p)$, where d is the given metric. We formally refer to this problem as Minimum-Load k -Clustering (MLkC). MLkC can be used to model applications where the cost of serving the clients (or points) assigned to a facility (or center) is incurred by the facility, e.g., assigning jobs to the k best servers from a pool of servers balancing their loads.

Surprisingly, MLkC is NP-hard even if the solution set of centers C is given, via a reduction from makespan-minimization [2]. In fact, this assignment version of the problem can be shown to be NP-hard even in line metrics and for $k = 2$, via a simple reduction from the Partition problem [29]. (In Partition, given a set of integers, the goal is to partition it into two subsets such that the difference between the sums of the integers in two subsets is minimized.) Moreover, [2] proved that the problem is strongly NP-hard in line metrics (points on a line) and APX-hard in the plane. On the positive side, an $O(k)$ -approximation follows for this problem from any existing $O(1)$ -approximation for k -median [12, 5, 24, 31, 11]. This is true, as k -median minimizes the sum of the loads of the centers. Also, constant-approximations are known for MLkC in some special cases, e.g., in star metrics and line metrics. Beyond these special cases, obtaining better than $O(k)$ -approximation in polynomial time remained a notoriously hard question, even in the plane. Indeed, as explicitly pointed out by [2], MLkC is resilient to attack by the standard approximation techniques including LP rounding and local search, which has been fairly successful in obtaining good approximation algorithms for other clustering problems. Given these difficulties, we investigate whether it is possible to obtain $O(1)$ -approximation for MLkC if we allow time $f(k) \cdot n^{O(1)}$ instead of only $n^{O(1)}$, for some function $f(\cdot)$ independent of the input size n . Indeed, we study a much more general *fair* version of the problem.

Fair clustering was introduced by [14] with the goal of removing inherent biases from the regular clustering models. In this setting, we also have a sensitive or protected feature of the data points, e.g., gender or race. The goal is to obtain a clustering where the fraction of points from a traditionally underrepresented group (w.r.t. the protected feature) in every cluster is approximately equal to the fraction of points from this group in the whole dataset. For simplicity, they assumed that the protected feature can take only two values and designed fair k -center and k -median clustering algorithms in this setting. In particular, here one is given two sets of points R and B of color red and blue, respectively, and a balance parameter $t \in [0, 1]$. The objective is to find a clustering such that in every cluster O , the ratio between the number of red points and the number of blue points is at least t and at

most $1/t$, i.e., $t \leq \frac{|O \cap R|}{|O \cap B|} \leq 1/t$. Subsequently, [33] considered a general model where the protected feature can take any number of values and designed fair clustering algorithms for the k -center objective. Later, [9] and [8] independently considered a fair clustering model that generalizes the models in both [14] and [33]. In this model, we are given a partition $\{P_1, P_2, \dots, P_\ell\}$ of the input point set P and balance parameters $0 \leq \beta_i \leq \alpha_i \leq 1$ for each group $1 \leq i \leq \ell$. Then a clustering is called (α, β) -fair if the fraction of points from each group i in every cluster is at least β_i and at most α_i . In this paper, we study the (α, β) -Fair Minimum-Load k -Clustering (FMLkC) problem, where the goal is to compute an (α, β) -fair clustering that minimizes the maximum load. (For a formal definition, please see Section 2.) We note that the only clustering objectives considered in all the above mentioned works on fair clustering are k -means, k -median and k -center. To the best of our knowledge, fair clustering was not studied with the minimum-load objective before our work.

1.1 Our Results and Techniques

Considering the FMLkC problem in general and Euclidean metric spaces we obtain the following results.

► **Theorem 1 (Informal).** *There is a 3-approximation algorithm for (α, β) -Fair Minimum-Load k -Clustering in general metric spaces that runs in time $2^{\tilde{O}(k\ell^2)} n^{O(1)}$. For d -dimensional Euclidean spaces, there is a $(1 + \epsilon)$ -approximation algorithm for (α, β) -Fair Minimum-Load k -Clustering with running time $2^{\tilde{O}(k\ell^2/\epsilon^{O(1)})} n^{O(1)} d$.*

In the above theorem, the $\tilde{O}(\cdot)$ notation hides logarithmic factors. Note that all the running times are *fixed-parameter tractable* (FPT) [16] in k , ℓ and ϵ . Moreover, our results imply the same approximations for Minimum-Load k -Clustering with running times FPT in only k and ϵ , achieving the first constant-factor FPT approximations for this problem in general and Euclidean metric spaces. Note that in the Euclidean case, the running time depends only polynomially on the dimension d . Recall that no better than $O(k)$ -approximation was known before even in the plane, and this version is known to be APX-hard. Also, the reduction mentioned before from Partition eliminates the existence of an exact algorithm in time $f(k) \cdot n^{O(1)}$, unless $P \neq NP$, as MLkC in line metrics is already NP-hard when $k = 2$. In this sense, our FPT $(1 + \epsilon)$ -approximation for Euclidean spaces is tight and the best possible.

Our results are motivated by the recent FPT approximation results for constrained clustering with popular k -median and k -means objectives [15, 7]. However, these results are based on coresets construction. A coreset is a summary of the original dataset from which it is possible to retrieve a near-optimal clustering. Their main contribution is to show that it is possible to obtain coresets of size polynomial in k , $\log n$ and d . Alternatively, the input can be compressed to an almost equivalent instance of size $\text{poly}(kd \log n)$. Then one can enumerate all possible k -tuples of centers in FPT time using the coreset and output the k -tuple having the minimum clustering cost. This yields a $(1 + \epsilon)$ -approximation for Euclidean spaces and a slightly larger 3-approximation for general metric spaces due to some technical reasons. However, such a small-sized coreset is not known for our problems. Instead, we adapt approaches from [17, 23, 10] used for directly obtaining FPT approximations for constrained k -median and k -means clustering. We note that these schemes were known only in the special Euclidean case until recently [23]. All these schemes produce in FPT (in k) time a list of k -tuples of centers, such that at least one such k -tuple is a near-optimal set of centers. Using the similarity of the k -median and the minimum-load objectives, we show these approaches can be adapted for our problems as well. However, given such a k -tuple of

centers, assigning the points to the best centers or finding the optimal clustering, in our case is still NP-hard. Nevertheless, we give a Mixed-Integer Linear Programming (MILP) based $(1 + \epsilon)$ -approximation for this assignment problem that runs in time FPT in k and ℓ (in k only for MLkC). Our MILP is partly motivated by the fair k -median MILP [7]. However, our MILP and its rounding are more involved compared to that for fair k -median, especially due to the difference in the objectives. For example, if we forget about the fairness constraints, in that case the assignment algorithm for k -median is trivial: assign each point to its closest center. However, even in this case the assignment problem for MLkC is NP-hard. Also, no near-optimal assignment scheme was known in the literature (a 2-approximate assignment scheme follows from the generalized assignment problem (GAP) [34]). Thus, in this case we give a novel $(1 + \epsilon)$ -approximate assignment scheme. In this case, we do not need MILP – rounding of an LP is sufficient to obtain the desired assignment. All these schemes applied together help us achieve the desired FPT approximations.

1.2 Related Work

[18] and [4] studied the MLkC problem under the name *min-max star cover*, where $F = P$. In this setting, MLkC can be viewed as a weighted covering problem where the task is to cover the nodes of a graph by stars. Both works obtain *bicriteria* approximation for this problem where the solution returned has near-optimal load, but uses more than k centers. [2] studied several special cases of the MLkC problem (under the name Minimum-Load k -Facility Location¹). They fully resolved the status of the MLkC problem in line metrics. On the one hand, they designed a PTAS based on dynamic programming. On the other hand, they proved that this version is strongly NP-hard. They also designed a quasi-PTAS in tree metrics. Moreover, they studied a variant of the problem with client demands in star metrics.

The notion of fair clustering introduced by [14] has been studied extensively in the literature. For k -center objective, there are several polynomial-time true $O(1)$ -approximations [33, 9]. For k -median and k -means objectives, polynomial-time $O(1)$ -approximations are designed by violating the fairness constraints by an additive factor [9, 8] and true $O(d \log n)$ -approximation is known in \mathbb{R}^d for two groups [6]. On the other hand, it is possible to obtain true $O(1)$ -approximations for these two objectives if one is allowed to use $f(k, \ell) \cdot n^{O(1)}$ time [7]. Clustering problems have been studied under several other notions of fairness, e.g., see [3, 13, 27, 28, 21, 32, 1].

Organization. We define some useful notations and our problem formally in Section 2. Then we describe the assignment algorithm for the FMLkC problem in Section 3. Finally, in Section 4, we describe the full algorithms for FMLkC in details.

2 Preliminaries

We are given a set P of points in a metric space $(\mathcal{X}, d(\cdot, \cdot))$, that we have to cluster. We are also given a set F of cluster centers in the same metric space. We note that P and F are not-necessarily disjoint, and in fact, P may be equal to F . In the Euclidean version of a clustering problem, $P \subseteq \mathbb{R}^d$, $F = \mathbb{R}^d$ and $d(\cdot, \cdot)$ is the Euclidean distance.² In the metric

¹ MLkC can also be viewed as a facility location problem with zero facility opening costs where we can still open only k facilities.

² Due to the lack of better notations, we denote the dimension by d and distance function by $d(\cdot, \cdot)$.

version, we assume that F is finite. Thus, strictly speaking, the Euclidean version is not a special case of the metric version. In the metric version, we denote $|P \cup F|$ by n and in the Euclidean version, $|P|$ by n . For any integer $t \geq 1$, we denote the set $\{1, 2, \dots, t\}$ by $[t]$.

For a partition $\mathbb{O} = \{O_1, \dots, O_k\}$ of P and a set of k cluster centers $C = \{c_1, \dots, c_k\} \subset F$, we say that \mathbb{O} is a *clustering* of P with the centers c_1, \dots, c_k . We say that the *load cost* of this clustering (also, simply *cost of clustering*) is $\max_{i \in [k]} \text{cost}_{c_i}(O_i)$. Here $\text{cost}_{c_i}(O_i)$ denotes the sum-of-distances cost of the cluster O_i with the center c_i , which is $\text{cost}_{c_i}(O_i) = \sum_{x \in O_i} d(x, c_i)$. We use the following notation to denote the cost of clustering w.r.t. the set of centers C up to a permutation of the clusters,

$$\text{cost}_C(\mathbb{O}) = \min_{i_1, \dots, i_k} \max_{j \in [k]} \text{cost}_{c_j}(O_{i_j}),$$

where i_1, \dots, i_k is a permutation of $[k]$. We also denote by $\text{cost}(O_i)$ the cost of a cluster O_i with the optimal choice of a center, that is, $\text{cost}(O_i) = \min_{c \in F} \text{cost}_c(O_i)$, and by $\text{cost}(\mathbb{O})$ the optimal cost of clustering \mathbb{O} ,

$$\text{cost}(\mathbb{O}) = \min_{\substack{C \subset F \\ |C|=k}} \text{cost}_C(\mathbb{O}).$$

Alternatively, a clustering with centers in $C \subset F$ can be defined as an assignment $\varphi : P \rightarrow C$. The assignment φ then corresponds to a clustering $\{\varphi^{-1}(c)\}_{c \in C}$, and we say that the cost of the assignment φ is $\text{cost}(\varphi) = \max_{c \in C} \sum_{x \in P: \varphi(x)=c} d(x, c)$.

Now we formally define the main problem of our interest, where the goal is to find the minimum-cost clustering that satisfies the fairness constraints.

► **Definition 2.** *In the (α, β) -Fair Minimum-Load k -Clustering (FMLkC) problem, we are given a partition $\{P_1, P_2, \dots, P_\ell\}$ of P . We are also given an integer $k > 0$ and two fairness vectors $\alpha, \beta \in [0, 1]^\ell$, $\alpha = (\alpha_1, \dots, \alpha_\ell)$, $\beta = (\beta_1, \dots, \beta_\ell)$. The objective is to select a set of at most k centers $C \subset F$ and an assignment $\varphi : P \rightarrow C$ such that φ satisfies the following fairness constraints:*

$$\begin{aligned} |\{x \in P_i : \varphi(x) = c\}| &\leq \alpha_i \cdot |\{x \in P : \varphi(x) = c\}|, & \forall c \in C, \forall i \in [\ell], \\ |\{x \in P_i : \varphi(x) = c\}| &\geq \beta_i \cdot |\{x \in P : \varphi(x) = c\}|, & \forall c \in C, \forall i \in [\ell], \end{aligned}$$

and $\text{cost}(\varphi)$ is minimized among all such assignments.

Minimum-Load k -Clustering (MLkC) is a restricted case of FMLkC with $\ell = 1$, and hence there is no fairness constraints involved in this case. The (α, β) -Fair k -median problem is defined identically except there the cost is $\text{cost}(\varphi) = \sum_{c \in C} \sum_{x \in P: \varphi(x)=c} d(x, c)$.

3 Assignment Problem for FMLkC

In the (α, β) -fair assignment problem, we are additionally given a set of centers $C \subset F$ and the goal is to find an assignment $\varphi : P \rightarrow C$ such that φ satisfies the fairness constraints and $\text{cost}(\varphi) = \max_{c \in C} \sum_{x \in P: \varphi(x)=c} d(x, c)$ is minimized.

We refer to an assignment as a fair assignment if it satisfies the fairness constraints. Also, we denote the optimal cost of an (α, β) -fair assignment by OPT. In this section, for any $\epsilon > 0$, we give a $(1 + \epsilon)$ -approximation for this problem in $f(k, \ell, \epsilon) \cdot n^{O(1)}$ time for some computable function f . In particular, we solve a budgeted version of the problem where we are also given a budget B and the goal is to decide whether there is a fair assignment of cost at most B .

► **Lemma 3.** *Suppose there is an algorithm \mathcal{A} that given an instance of budgeted (α, β) -fair assignment and any $\epsilon > 0$, in $T(n, k, \ell, \epsilon)$ time, either returns a feasible assignment of cost at most $(1 + \epsilon)B$, or correctly detects that there is no feasible assignment with budget B . Then for any $\epsilon > 0$, one can obtain a $(1 + \epsilon)$ -approximation for (α, β) -fair assignment in $(k\ell)^{O(k\ell)}n^{O(1)} + O_\epsilon(\log k) \cdot T(n, k, \ell, \epsilon/3)$ time³.*

Proof. The idea is to first find a range where OPT belongs and then apply \mathcal{A} with budget within this range to find a feasible assignment. Given an instance I of (α, β) -fair assignment, first we use an algorithm (Theorem 8.2, [7]) to compute a fair assignment of the points to the centers of C that minimizes the (α, β) -fair k -median cost. This algorithm runs in time $(k\ell)^{O(k\ell)}n^{O(1)}$. Let D be the computed (α, β) -fair k -median cost returned by the algorithm. Then $D \leq k \cdot \text{OPT}$, as the optimal cost of (α, β) -fair assignment is at least $1/k$ fraction of the optimal (α, β) -fair k -median cost. Also, $\text{OPT} \leq D$, as optimal (α, β) -fair assignment cost is at most the optimal (α, β) -fair k -median cost. Hence $D/k \leq \text{OPT} \leq D$.

Let $\epsilon' = \epsilon/3$ and m be the maximum t such that $(1 + \epsilon')^t \leq D/k$. Also, let M be the minimum t such that $D \leq (1 + \epsilon')^t$. Thus $(1 + \epsilon')^m \leq \text{OPT} \leq (1 + \epsilon')^M$. We run the algorithm \mathcal{A} setting ϵ to be ϵ' for budget $B = (1 + \epsilon')^i$ where $m \leq i \leq M$, and use the binary search to find minimum i such that it returns a feasible assignment for some budget B . Let B' be the budget for which this algorithm returns a feasible assignment. Then $B' \leq (1 + \epsilon') \text{OPT}$, as any instance with budget $B \geq \text{OPT}$ is a yes-instance, and for such a B , \mathcal{A} returns a feasible assignment of cost at most $(1 + \epsilon)B$. Hence, the cost of the assignment returned by \mathcal{A} with budget B' is at most $(1 + \epsilon')^2 \text{OPT} \leq (1 + \epsilon) \text{OPT}$. As the algorithm \mathcal{A} can be used at most $O_\epsilon(\log(M - m + 1)) = O_\epsilon(\log(D/(D/k))) = O_\epsilon(\log k)$ times, the whole algorithm runs in time $(k\ell)^{O(k\ell)}n^{O(1)} + O_\epsilon(\log k) \cdot T(n, k, \ell, \epsilon/3)$. ◀

In the following, we design an LP rounding based algorithm for budgeted (α, β) -fair assignment with the properties required in the above lemma. Moreover, this algorithm runs in time $k^{O(k\ell)}\ell^{O((k\ell^2/\epsilon)\log(\ell/\epsilon))}n^{O(1)}$. Hence, we obtain the following theorem.

► **Theorem 4.** *For any $\epsilon > 0$, a $(1 + \epsilon)$ -approximation for (α, β) -fair assignment can be obtained in time $k^{O(k\ell)}\ell^{O((k\ell^2/\epsilon)\log(\ell/\epsilon))}n^{O(1)}$.*

Next, we design the algorithm for the budgeted version of (α, β) -fair assignment. Recall that in the budgeted version, we are given an instance I containing ℓ disjoint groups $\{P_i\}$ of $P = \{p_1, \dots, p_n\}$, a set of k centers $C = \{c_1, \dots, c_k\}$ and the budget B . Our algorithm first rounds each distance to a power of $(1 + \epsilon)$. Fix any center $c_i \in C$. We partition the points in P into a number of classes based on their distances from c_i . For all $p \in P$, let $\hat{d}(p, c_i) = (1 + \epsilon)^t \epsilon^2 B$ where $t = \lceil \log_{1+\epsilon}(d(p, c_i)/(\epsilon^2 B)) \rceil$. For each distinct t , let $d_t = (1 + \epsilon)^t \epsilon^2 B$. We refer to the points p with distance $\hat{d}(p, c_i) = d_t$ from c_i as the *distance class* t with respect to (w.r.t.) c_i , which is denoted by S_{it} .

► **Observation 5.** *For all $p \in P, c \in C, d(p, c) \leq \hat{d}(p, c) \leq (1 + \epsilon) \cdot d(p, c)$.*

Let I' be the new instance of budgeted (α, β) -fair assignment with the modified distance \hat{d} . Note that \hat{d} does not necessarily satisfy the triangle inequality. As \hat{d} is obtained by scaling d by at most a factor of $(1 + \epsilon)$, we have the following observation.

► **Observation 6.** *If there is a feasible assignment for I with budget B , then there is a feasible assignment for I' with budget $(1 + \epsilon)B$. Also, if there is a feasible assignment for I' with budget $(1 + \epsilon)B$, then there is a feasible assignment for I with budget $(1 + \epsilon)B$.*

³ $O_\epsilon(\cdot)$ notation hides a factor of $O(1/\epsilon)$.

By the above observation, it is sufficient to consider \hat{d} instead of d for the purpose of computing an assignment of cost at most $(1 + \epsilon)B$. Henceforth, by distance we mean \hat{d} .

Denote by φ^* a feasible assignment for I' of cost at most $(1 + \epsilon)B$ (if any). We define a point $p \in P$ to be *costly* w.r.t. a center $c \in C$ if $\hat{d}(p, c) \geq \epsilon^2 B$. Otherwise, we define the point to be *cheap* w.r.t. c . Note that the number of costly points that can be assigned to each center in φ^* is at most $(1 + \epsilon)/\epsilon^2 \leq 2/\epsilon^2$. The next observation follows from the fact that $d_t = (1 + \epsilon)^t \epsilon^2 B$.

► **Observation 7.** *For any point p and center $c \in C$ with $\hat{d}(p, c) = d_t$, $t < 0$ if p is cheap w.r.t. c , and $t \geq 0$ if p is costly w.r.t. c .*

Now, as we are shooting for an assignment of cost at most $(1 + \epsilon)B$, we can discard all the distances $\hat{d}(p, c)$ larger than $(1 + \epsilon)B$, i.e., we can assume that such a p will never be assigned to c . Without loss of generality, we assume that all the distances we have are bounded by $(1 + \epsilon)B$. Let Δ be the maximum t such that there are $p \in P$ and $c \in C$ with $\hat{d}(p, c) = d_t$ for a costly point p w.r.t. c . By our previous assumption, $\hat{d}(p, c) \leq (1 + \epsilon)B$. Thus $\Delta \leq \lceil \log_{1+\epsilon}((1 + \epsilon)B/(\epsilon^2 B)) \rceil = O((1/\epsilon) \log(1/\epsilon))$. For $1 \leq i \leq k$, $0 \leq t \leq \Delta$ and $1 \leq g \leq \ell$, let $z_{i,t,g}$ be the number of costly points $p \in P_g$ assigned to c_i in φ^* such that $\hat{d}(p, c_i) = d_t$. Note that each $z_{i,t,g} \leq 2/\epsilon^2$, as the total number of costly points assigned to a center is at most $2/\epsilon^2$. Thus the total number of distinct choices for these variables is $(2/\epsilon^2)^{k\ell\Delta} = (1/\epsilon)^{O((k\ell/\epsilon) \log(1/\epsilon))}$.

► **Observation 8.** *There are $(1/\epsilon)^{O((k\ell/\epsilon) \log(1/\epsilon))}$ distinct choices for the variables $\{z_{i,t,g} : 1 \leq i \leq k, 0 \leq t \leq \Delta, 1 \leq g \leq \ell\}$.*

As we can probe all such possible choices, we assume that we know the exact values of these variables in φ^* . Next, we describe a Mixed-Integer Linear Program (MILP) for the budgeted version of the problem, which is partly motivated by the (α, β) -fair k -median MILP [7]. For every point p_j and center c_i , we have a fractional variable x_{ij} denoting the extent up to which p_j is assigned to c_i . For every center c_i and group $g \in \{1, \dots, \ell\}$, we have an integral variable y_{gi} denoting the “weight” of the points assigned to c_i from group g . The constraints of the MILP are described as follows. Constraint 1 ensures that each point is assigned to the centers up to an extent of 1. Constraint 2 ensures that the weight assigned from each group g to each center c_i is exactly y_{gi} . Constraints 3 and 4 are fairness constraints. Constraint 5 ensures that the weight of costly points from each class t and group g assigned to each c_i is exactly the same as the guessed value $z_{i,t,g}$. Constraint 6 ensures that the total load assigned to each c_i is bounded by $(1 + \epsilon)B$. The first and the second expressions on the left hand side of this constraint are corresponding to costly and cheap points, respectively.

$$\sum_{1 \leq i \leq k} x_{ij} = 1 \quad \forall j \in [n] \quad (1)$$

$$\sum_{j \in [n]: p_j \in P_g} x_{ij} = y_{gi} \quad \forall i \in [k], \forall g \in [\ell], \quad (2)$$

$$y_{gi} \geq \beta_g \sum_{j \in [n]} x_{ij} \quad \forall i \in [k], \forall g \in [\ell] \quad (3)$$

$$y_{gi} \leq \alpha_g \sum_{j \in [n]} x_{ij} \quad \forall i \in [k], \forall g \in [\ell] \quad (4)$$

$$\sum_{p_j \in P_g \cap S_{it}} x_{ij} = z_{i,t,g} \quad \forall i \in [k], \forall t \in \{0, \dots, \Delta\}, \forall g \in [\ell] \quad (5)$$

4:8 FPT Approximation for Fair Minimum-Load Clustering

$$\sum_{g=1}^{\ell} \sum_{0 \leq t \leq \Delta} d_t z_{i,t,g} + \sum_{t < 0} d_t \sum_{p_j \in S_{it}} x_{ij} \leq (1 + \epsilon)B \quad \forall i \in [k] \quad (6)$$

$$x_{ij} \geq 0 \quad \forall j \in [n], \forall i \in [k], \quad (7)$$

$$y_{gi} \in \mathbb{Z}_{\geq 0} \quad \forall i \in [k], \forall g \in [\ell]. \quad (8)$$

Let us denote the above MILP by Fair-LP. A solution to Fair-LP is denoted by (x, y) . We note that the assignment φ^* induces a feasible solution to Fair-LP. We use the following popular and celebrated result to solve this MILP.

► **Proposition 9** ([30, 25, 20]). *An MILP with K integral variables and encoding size L , can be solved in time $K^{O(K)} L^{O(1)}$.*

As Fair-LP has $k\ell$ integral variables $\{y_{gi}\}$ and polynomial encoding size, it can be solved in $(k\ell)^{O(k\ell)} n^{O(1)}$ time. If this algorithm outputs that there is no feasible solution to Fair-LP, we conclude that I is a no-instance. Otherwise, let (x^*, y^*) denote the feasible solution returned by this algorithm. Note that although the y^* values are integral, x^* values can very well be fractional. Next, we show how to round these variables to obtain an integral solution to Fair-LP such that the load of every center is increased by an additive amount of $O(\epsilon\ell B)$ compared to its load in (x^*, y^*) .

Fix any group g . First, we show how to round the variables corresponding to the points of P_g . For this purpose, we construct a network $G_N = (V_N, E_N)$ with source S and sink T (see Figure 1). For each point $p_j \in P_g$, there is a node v_j in V_N . For each distance class t of every center c_i , there is a node w_{it} . Also, for each center c_i , there is a node u_i . For each $v_j \in V_N$, there is an arc (S, v_j) of capacity 1. For each $p_j \in P_g$ and center c_i , there is an arc (v_j, w_{it}) of capacity 1 where t is the index such that $p_j \in S_{it}$, i.e. $\hat{d}(p_j, c_i) = d_t$. For center c_i and distance class t , let $\lambda_{it}^g = \sum_{p_j \in P_g \cap S_{it}} x_{ij}^*$, i.e. the weight assigned from $P_g \cap S_{it}$ to c_i . For each node w_{it} , there is an arc (w_{it}, u_i) of capacity $\lceil \lambda_{it}^g \rceil$. Lastly, for each center c_i , there is an arc (u_i, T) of capacity y_{gi}^* .

Note that for each center c_i , the number of non-empty distance classes is at most the number of points n . Hence, the size of V_N is a polynomial in n . Also, note that the solution (x^*, y^*) projected on the points of P_g induces a feasible fractional solution for the problem of computing a flow of value $|P_g|$ in G_N .

► **Observation 10.** *The network G_N has a fractional flow of value $|P_g|$.*

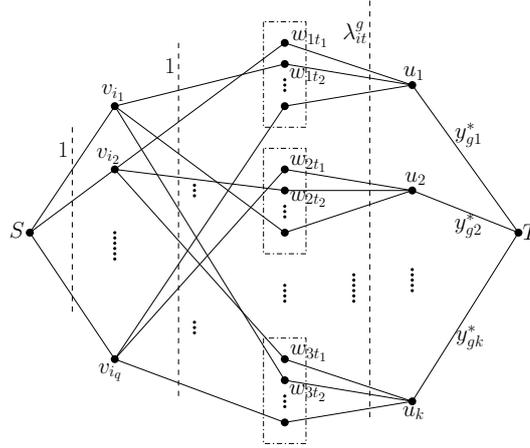
As all the capacities of the arcs are integral, by integrality of flow, there exists an integral feasible flow in G_N of value $|P_g|$. We compute such a flow f by using any polynomial time flow computation algorithm. This flow solution f naturally gives us an integral assignment φ_f of the points in P_g to the centers in C .

► **Observation 11.** *The number of points assigned to each center $c_i \in C$ via φ_f is y_{gi}^* .*

Proof. Due to the capacity constraints of the arcs $\{(u_i, T)\}$, the number of points assigned to c_i must be at most y_{gi}^* . Also, by definition, $\sum_{i=1}^k y_{gi}^* = |P_g|$. As f has value $|P_g|$, the capacity of the arcs $\{(u_i, T)\}$ must be saturated, which completes the proof. ◀

Next, we analyze the load of P_g assigned to each center via φ_f .

► **Lemma 12.** *For each center $c_i \in C$, $\sum_{p_j \in P_g: \varphi_f(p_j) = c_i} \hat{d}(p_j, c_i) \leq \sum_t d_t \lambda_{it}^g + O(\epsilon)B$.*



■ **Figure 1** Figure showing the network G_N constructed using the solution (x^*, y^*) .

Proof. Consider the arcs $\{(w_{it}, u_i)\}$. The maximum flow corresponding to these arcs is bounded by the sum of the capacities $\sum_t \lceil \lambda_{it}^g \rceil$. Note that for every $0 \leq t \leq \Delta$, $\lambda_{it}^g = z_{i,t,g}$, which is an integer. Now,

$$\sum_{t < 0} d_t = \sum_{t < 0} (1 + \epsilon)^t \epsilon^2 B < \epsilon^2 B ((1 + \epsilon)/\epsilon) = O(\epsilon)B \quad (9)$$

Hence,

$$\sum_{t < 0} d_t \lceil \lambda_{it}^g \rceil \leq \sum_{t < 0} d_t (\lambda_{it}^g + 1) \leq \sum_{t < 0} d_t \lambda_{it}^g + \sum_{t < 0} d_t = \sum_{t < 0} d_t \lambda_{it}^g + O(\epsilon)B$$

It follows that,

$$\begin{aligned} \sum_{p_j \in P_g: \varphi_f(p_j) = c_i} \hat{d}(p_j, c_i) &\leq \sum_t d_t \lceil \lambda_{it}^g \rceil = \sum_{t \geq 0} d_t \lambda_{it}^g + \sum_{t < 0} d_t \lceil \lambda_{it}^g \rceil \\ &= \sum_{t \geq 0} d_t \lambda_{it}^g + \sum_{t < 0} d_t \lambda_{it}^g + O(\epsilon)B = \sum_t d_t \lambda_{it}^g + O(\epsilon)B \quad \blacktriangleleft \end{aligned}$$

We repeat the above rounding process for all groups g . We combine the assignment functions φ_f corresponding to the ℓ disjoint groups to obtain a single assignment for the points in P . For simplicity, we also refer to this combined assignment as φ_f . By Observation 11, φ_f is feasible, as for each center c_i and each group g , the weight of the points in P_g assigned to c_i is exactly y_{gi}^* as in (x^*, y^*) . Next, we analyze the total load of each center.

► **Lemma 13.** For each center $c_i \in C$, $\sum_{p_j: \varphi_f(p_j) = c_i} \hat{d}(p_j, c_i) \leq (1 + O(\epsilon\ell))B$

Proof.

$$\begin{aligned} \sum_{p_j: \varphi_f(p_j) = c_i} \hat{d}(p_j, c_i) &= \sum_{g=1}^{\ell} \sum_{p_j \in P_g: \varphi_f(p_j) = c_i} \hat{d}(p_j, c_i) \\ &= \sum_{g=1}^{\ell} \left(\sum_t d_t \lambda_{it}^g + O(\epsilon)B \right) \quad (\text{By Lemma 12}) \\ &= \sum_{g=1}^{\ell} \left(\sum_{t \geq 0} d_t \lambda_{it}^g + \sum_{t < 0} d_t \lambda_{it}^g \right) + O(\epsilon\ell)B \end{aligned}$$

$$\begin{aligned}
 &= \sum_{g=1}^{\ell} \left(\sum_{t \geq 0} d_t z_{i,t,g} + \sum_{t < 0} d_t \left(\sum_{p_j \in P_g \cap S_{it}} x_{ij}^* \right) \right) + O(\epsilon \ell) B \\
 &= \left(\sum_{g=1}^{\ell} \sum_{t \geq 0} d_t z_{i,t,g} + \sum_{t < 0} d_t \sum_{p_j \in S_{it}} x_{ij}^* \right) + O(\epsilon \ell) B \\
 &\leq (1 + \epsilon) B + O(\epsilon \ell) B && \text{(By Constraint 6 of Fair-LP)} \\
 &= (1 + O(\epsilon \ell)) B && \blacktriangleleft
 \end{aligned}$$

By scaling ϵ down by a factor of $\Omega(\ell)$, we obtain the desired approximate bound. Thus, by Observation 8, it follows that the number of distinct possible choices of the $z_{i,t,g}$ values is $(\ell/\epsilon)^{O((k\ell^2/\epsilon) \log(\ell/\epsilon))}$. For each such choice, solving the MILP and rounding takes $(k\ell)^{O(k\ell)} n^{O(1)}$ time. Thus, the algorithm for solving the budgeted version runs in time $k^{O(k\ell)} 2^{O((k\ell^2/\epsilon) \log^2(\ell/\epsilon))} n^{O(1)}$. The following lemma completes the proof of Theorem 4.

► **Lemma 14.** *The above MILP based algorithm for budgeted (α, β) -fair assignment, in time $k^{O(k\ell)} 2^{O((k\ell^2/\epsilon) \log^2(\ell/\epsilon))} n^{O(1)}$, either returns a feasible assignment of budget at most $(1 + \epsilon)B$, or correctly detects that there is no feasible assignment of budget B .*

4 Approximation Algorithms for FMLkC

In this section, we describe the FPT approximation algorithms for the FMLkC problem, both in the general metric case and in the Euclidean case. In the general metric case, we aim for a $(3 + \epsilon)$ -approximation, and in the Euclidean case for a $(1 + \epsilon)$ approximation, for a given $0 < \epsilon < 1$. Essentially, we obtain these algorithms as a combination of our assignment algorithm presented before, and known generic results for constrained clustering problems that follow the framework of [17]. Our general metric algorithm employs the result of [23], and in the Euclidean case we use the result of [10]. Both of these provide algorithms that in FPT time produce a reasonably short list of candidate sets of k centers, such that for each possible clustering of the input points one of the sets in the list provides the desired approximation, with good probability. Note that the results mentioned above are stated in fact for the k -median objective, and not the minimum-load clustering that we study in this work. However, by tweaking the error guarantees in the respective proofs we can show that these results hold in the minimum-load setting as well. Next, we present these in detail.

We start with the Euclidean case and show the following analogue of Theorem 1 in [10] for the minimum-load objective.

► **Theorem 15.** *Given a set of n points $P \subset \mathbb{R}^d$, parameters k and $0 < \epsilon < 1$, there is a randomized algorithm that in time $2^{\tilde{O}(k/\epsilon^{O(1)})} nd$ outputs a list L of $2^{\tilde{O}(k/\epsilon^{O(1)})}$ k -sized sets of centers such that for any partition $\mathbb{P}^* = \{P_1^*, \dots, P_k^*\}$ of P the following event occurs with probability at least $1/2$: there is a set C in L such that*

$$\text{cost}_C(\mathbb{P}^*) \leq (1 + \epsilon) \max_{i \in [k]} \text{cost}(P_i^*).$$

Proof. The algorithm proceeds exactly as Algorithm 5.1 in [10]. For the analysis, we observe that Bhattacharya et al. prove the following statement (follows immediately from invariant $P(i)$ in [10]): With constant probability, there is a set of centers $C = \{c_1, \dots, c_k\}$ in the output of the algorithm and the permutation i_1, \dots, i_k of the clusters in \mathbb{P}^* such that for each $j \in [k]$,

$$\text{cost}_{c_j}(P_{i_j}^*) \leq (1 + \frac{\epsilon}{2}) \cdot \text{cost}(P_{i_j}^*) + \frac{\epsilon}{2k} \cdot \sum_{i=1}^k \text{cost}(P_i^*).$$

From here it easily follows that the set of centers C achieves $(1 + \epsilon)$ -approximation of the cost of \mathbb{P}^* with respect to the minimum-load objective:

$$\text{cost}_C(\mathbb{P}^*) \leq \max_{j \in [k]} \text{cost}_{c_j}(P_{i_j}^*) \leq (1 + \frac{\epsilon}{2}) \cdot \max_{j \in [k]} \text{cost}(P_{i_j}^*) + \frac{\epsilon}{2k} \cdot \sum_{i=1}^k \text{cost}(P_i^*) \leq (1 + \epsilon) \cdot \max_{j \in [k]} \text{cost}(P_{i_j}^*),$$

$$\text{since } \sum_{i=1}^k \text{cost}(P_i^*) \leq k \max_{j \in [k]} \text{cost}(P_{i_j}^*). \quad \blacktriangleleft$$

In the general metric case, a similar result can be shown, however with the approximation factor of $(3 + \epsilon)$. Specifically, we show an analogue of Theorem 5 in [23] for the minimum-load objective. Similarly to Theorem 15, the algorithm and the analysis is identical to what is presented in [23], up to a different view on the cost upper bound.

► **Theorem 16.** *Given a set of n points P in a metric space, parameters k and $0 < \epsilon < 1$, there is a randomized algorithm that in time $(k/\epsilon)^{O(k)}n$ outputs a list L of $(k/\epsilon)^{O(k)}$ k -sized sets of centers such that for any partition $\mathbb{P}^* = \{P_1^*, \dots, P_k^*\}$ of P the following event occurs with probability at least $1/2$: there is a set C in L such that*

$$\text{cost}_C(\mathbb{P}^*) \leq (3 + \epsilon) \max_{i \in [k]} \text{cost}(P_i^*).$$

Proof. The algorithm proceeds exactly as Algorithm 1 in [23]. For the analysis, we observe that Goyal et al. prove the following statement (encapsulated by **Property-I** in [23]): With constant probability, there is a set of centers $C = \{c_1, \dots, c_k\}$ in the output of the algorithm and the permutation i_1, \dots, i_k of the clusters in \mathbb{P}^* such that for each $j \in [k]$,

$$\text{cost}_{c_j}(P_{i_j}^*) \leq (3 + \frac{\epsilon}{2}) \cdot \text{cost}(P_{i_j}^*) + \frac{\epsilon}{2k} \cdot \sum_{i=1}^k \text{cost}(P_i^*).$$

Now, analogously to the proof of Theorem 15, it follows that the set of centers C achieves $(3 + \epsilon)$ -approximation of the cost of \mathbb{P}^* with respect to the minimum-load objective:

$$\text{cost}_C(\mathbb{P}^*) \leq \max_{j \in [k]} \text{cost}_{c_j}(P_{i_j}^*) \leq (3 + \frac{\epsilon}{2}) \cdot \max_{j \in [k]} \text{cost}(P_{i_j}^*) + \frac{\epsilon}{2k} \cdot \sum_{i=1}^k \text{cost}(P_i^*) \leq (3 + \epsilon) \cdot \max_{j \in [k]} \text{cost}(P_{i_j}^*),$$

$$\text{since } \sum_{i=1}^k \text{cost}(P_i^*) \leq k \max_{j \in [k]} \text{cost}(P_{i_j}^*). \quad \blacktriangleleft$$

Now, Theorem 15 and Theorem 16 imply that for the Minimum-Load k -Clustering problem with any given set of constraints on the desired clustering, there exists a $(1 + \epsilon)$ -approximation algorithm in the Euclidean case, and a $(3 + \epsilon)$ -approximation algorithm in the general metric case. The running time is $2^{\tilde{O}(k/\epsilon^{O(1)})}(nd + T)$ for both algorithms, where T is the running time of an algorithm solving the respective assignment problem, either exact or $(1 + \epsilon)$ -approximate. In particular, combining the theorems with our approximation algorithm for (α, β) -fair assignment (Theorem 4), for the FMLkC problem we obtain a $(1 + \epsilon)$ -approximation in \mathbb{R}^d and a $(3 + \epsilon)$ -approximation in general metric in FPT time when parameterized by the number of clusters k and the number of protected groups ℓ .

► **Theorem 17.** *For any $0 < \epsilon < 1$, there exists a randomized $(1 + \epsilon)$ -approximation algorithm for (α, β) -Fair Minimum-Load k -Clustering in \mathbb{R}^d with running time $2^{\tilde{O}(k\ell^2/\epsilon^{O(1)})}n^{O(1)}d$. The same holds in general metric with the approximation factor of $(3 + \epsilon)$, where the running time becomes $2^{\tilde{O}(k\ell^2/\epsilon)}n^{O(1)}$.*

Proof. First, we deal with the Euclidean case. Fix an optimal fair min-load k -clustering $\mathbb{P}^* = \{P_1^*, \dots, P_k^*\}$ of P . Run the algorithm of Theorem 15 on P with error parameter ϵ_0 to obtain the list L of candidate sets of centers, here ϵ_0 is such that $(1 + \epsilon_0)^2 \leq (1 + \epsilon)$. In the following, assume that the event described in the statement of Theorem 15 occurs for the clustering \mathbb{P}^* , by constant number of repetitions the probability of this can be lifted arbitrarily close to one. That is, there exists a set of k centers C' in L such that

$$\text{cost}_{C'}(\mathbb{P}^*) \leq (1 + \epsilon_0) \max_{i \in [k]} \text{cost}(P_i^*). \quad (10)$$

Now, for each set of centers C in L , run the $(1 + \epsilon_0)$ -approximate assignment algorithm given by Theorem 4 on (P, C) , and choose the set of centers C'' that gives the best assignment cost among the considered sets, denote the computed assignment from P to C'' by φ . In what follows, we show that the set of centers C'' and the assignment $\varphi : P \rightarrow C''$ provide $(1 + \epsilon)$ -approximate solution to the given FMLkC instance. Denote by ψ the assignment from P to C' that the algorithm outputs,

$$\text{cost}(\varphi) \leq \text{cost}(\psi) \leq (1 + \epsilon_0) \text{cost}_{C'}(\mathbb{P}^*) \leq (1 + \epsilon_0)^2 \max_{i \in [k]} \text{cost}(P_i^*) \leq (1 + \epsilon) \max_{i \in [k]} \text{cost}(P_i^*),$$

where the first inequality is by the choice of C'' and φ , the second is by Theorem 4, and the third inequality is by (10).

Finally, we show that the running time bound holds. Invoking Theorem 15 takes time $2^{\tilde{O}(k/\epsilon_0^{O(1)})} nd$, and produces a list of $2^{\tilde{O}(k/\epsilon_0^{O(1)})}$ sets of centers. On each of them, running the algorithm of Theorem 4 takes time $k^{O(k\ell)} \ell^{O((k\ell^2/\epsilon_0) \log(\ell/\epsilon_0))} n^{O(1)} d$. Since $\epsilon_0 = O(\epsilon)$, the total running time can be bounded as

$$2^{\tilde{O}(k/\epsilon^{O(1)})} \left(nd + k^{O(k\ell)} \ell^{O((k\ell^2/\epsilon) \log(\ell/\epsilon))} n^{O(1)} d \right) = 2^{\tilde{O}(k\ell^2/\epsilon^{O(1)})} n^{O(1)} d.$$

The general metric case is identical, but to obtain the list of candidate sets of centers we use Theorem 16 instead of Theorem 15. The final cost bound changes to

$$\text{cost}(\varphi) \leq \text{cost}(\psi) \leq (1 + \epsilon_0) \text{cost}_{C'}(\mathbb{P}^*) \leq (3 + \epsilon_0) \cdot (1 + \epsilon_0) \max_{i \in [k]} \text{cost}(P_i^*) \leq (3 + \epsilon) \max_{i \in [k]} \text{cost}(P_i^*),$$

where ϵ_0 is chosen so that $(3 + \epsilon_0) \cdot (1 + \epsilon_0) \leq (3 + \epsilon)$. \blacktriangleleft

References

- 1 Mohsen Abbasi, Aditya Bhaskara, and Suresh Venkatasubramanian. Fair clustering via equitable group representations. In Madeleine Clare Elish, William Isaac, and Richard S. Zemel, editors, *FAccT '21: 2021 ACM Conference on Fairness, Accountability, and Transparency, Virtual Event / Toronto, Canada, March 3-10, 2021*, pages 504–514. ACM, 2021.
- 2 Sara Ahmadian, Babak Behsaz, Zachary Friggstad, Amin Jorati, Mohammad R. Salavatipour, and Chaitanya Swamy. Approximation algorithms for minimum-load k -facility location. *ACM Trans. Algorithms*, 14(2):16:1–16:29, 2018.
- 3 Sara Ahmadian, Alessandro Epasto, Ravi Kumar, and Mohammad Mahdian. Clustering without over-representation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 267–275, 2019.
- 4 Esther M. Arkin, Refael Hassin, and Asaf Levin. Approximations for minimum and min-max vehicle routing problems. *J. Algorithms*, 59(1):1–18, 2006.

- 5 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k -median and facility location problems. *SIAM J. Comput.*, 33(3):544–562, 2004.
- 6 Arturs Backurs, Piotr Indyk, Krzysztof Onak, Baruch Schieber, Ali Vakilian, and Tal Wagner. Scalable fair clustering. In *International Conference on Machine Learning*, pages 405–413, 2019.
- 7 Sayan Bandyapadhyay, 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. doi:10.4230/LIPICs.ICALP.2021.23.
- 8 Suman Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. In *Advances in Neural Information Processing Systems*, pages 4954–4965, 2019.
- 9 Ioana O Bercea, Martin Groß, Samir Khuller, Aounon Kumar, Clemens Rösner, Daniel R Schmidt, and Melanie Schmidt. On the cost of essentially fair clusterings. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
- 10 Anup Bhattacharya, Ragesh Jaiswal, and Amit Kumar. Faster Algorithms for the Constrained k -means Problem. *Theory of Computing Systems*, 62(1):93–115, 2018.
- 11 Jaroslav Byrka, Thomas W. Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k -median and positive correlation in budgeted optimization. *ACM Trans. Algorithms*, 13(2):23:1–23:31, 2017.
- 12 Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k -median problem (extended abstract). In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 1–10, 1999.
- 13 Xingyu Chen, Brandon Fain, Liang Lyu, and Kamesh Munagala. Proportionally fair clustering. In *International Conference on Machine Learning*, pages 1032–1041, 2019.
- 14 Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In *Advances in Neural Information Processing Systems*, pages 5029–5037, 2017.
- 15 Vincent Cohen-Addad and Jason Li. On the fixed-parameter tractability of capacitated clustering. 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 41:1–41:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- 16 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.
- 17 Hu Ding and Jinhui Xu. A unified framework for clustering constrained data without locality property. *Algorithmica*, 82(4):808–852, 2020.
- 18 Guy Even, Naveen Garg, Jochen Könnemann, R. Ravi, and Amitabh Sinha. Covering graphs using trees and stars. In Sanjeev Arora, Klaus Jansen, José D. P. Rolim, and Amit Sahai, editors, *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques, 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2003 and 7th International Workshop on Randomization and Approximation Techniques in Computer Science, RANDOM 2003, Princeton, NJ, USA, August 24–26, 2003, Proceedings*, volume 2764 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2003.
- 19 Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 434–444, 1988.

- 20 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, March 1987.
- 21 Mehrdad Ghadiri, Samira Samadi, and Santosh S. Vempala. Socially fair k-means clustering. In Madeleine Clare Elish, William Isaac, and Richard S. Zemel, editors, *FACCT '21: 2021 ACM Conference on Fairness, Accountability, and Transparency, Virtual Event / Toronto, Canada, March 3-10, 2021*, pages 438–448. ACM, 2021.
- 22 Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
- 23 Dishant Goyal, Ragesh Jaiswal, and Amit Kumar. FPT Approximation for Constrained Metric k-Median/Means. In *15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*, volume 180 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.
- 24 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k -median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.
- 25 Ravi Kannan. Minkowski’s convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, August 1987.
- 26 Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. *Comput. Geom.*, 28(2-3):89–112, 2004.
- 27 Matthäus Kleindessner, Pranjal Awasthi, and Jamie Morgenstern. Fair k-center clustering for data summarization. In *36th International Conference on Machine Learning, ICML 2019*, pages 5984–6003. International Machine Learning Society (IMLS), 2019.
- 28 Matthäus Kleindessner, Samira Samadi, Pranjal Awasthi, and Jamie Morgenstern. Guarantees for spectral clustering with fairness constraints. *arXiv preprint*, 2019. [arXiv:1901.08668](https://arxiv.org/abs/1901.08668).
- 29 Richard E. Korf. A complete anytime algorithm for number partitioning. *Artificial Intelligence*, 106(2):181–203, 1998.
- 30 H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, November 1983.
- 31 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. *SIAM J. Comput.*, 45(2):530–547, 2016.
- 32 Yury Makarychev and Ali Vakilian. Approximation algorithms for socially fair clustering. 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 3246–3264. PMLR, 2021. URL: <http://proceedings.mlr.press/v134/makarychev21a.html>.
- 33 Clemens Rösner and Melanie Schmidt. Privacy preserving clustering with constraints. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- 34 David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62:461–474, 1993. doi:10.1007/BF01585178.

On Sparse Hitting Sets: From Fair Vertex Cover to Highway Dimension

Johannes Blum   

Universität Konstanz, Germany

Yann Disser  

Technische Universität Darmstadt, Germany

Andreas Emil Feldmann   

Charles University, Prague, Czechia

Siddharth Gupta   

University of Warwick, Coventry, UK

Anna Zych-Pawlewicz  

University of Warsaw, Poland

Abstract

We consider the SPARSE HITTING SET (SPARSE-HS) problem, where we are given a set system $(V, \mathcal{F}, \mathcal{B})$ with two families \mathcal{F}, \mathcal{B} of subsets of the universe V . The task is to find a hitting set for \mathcal{F} that minimizes the maximum number of elements in any of the sets of \mathcal{B} . This generalizes several problems that have been studied in the literature. Our focus is on determining the complexity of some of these special cases of SPARSE-HS with respect to the *sparseness* k , which is the optimum number of hitting set elements in any set of \mathcal{B} (i.e., the value of the objective function).

For the SPARSE VERTEX COVER (SPARSE-VC) problem, the universe is given by the vertex set V of a graph, and \mathcal{F} is its edge set. We prove NP-hardness for sparseness $k \geq 2$ and polynomial time solvability for $k = 1$. We also provide a polynomial-time 2-approximation algorithm for any k . A special case of SPARSE-VC is FAIR VERTEX COVER (FAIR-VC), where the family \mathcal{B} is given by vertex neighbourhoods. For this problem it was open whether it is FPT (or even XP) parameterized by the sparseness k . We answer this question in the negative, by proving NP-hardness for constant k . We also provide a polynomial-time $(2 - \frac{1}{k})$ -approximation algorithm for FAIR-VC, which is better than any approximation algorithm possible for SPARSE-VC or the VERTEX COVER problem (under the Unique Games Conjecture).

We then switch to a different set of problems derived from SPARSE-HS related to the *highway dimension*, which is a graph parameter modelling transportation networks. In recent years a growing literature has shown interesting algorithms for graphs of low highway dimension. To exploit the structure of such graphs, most of them compute solutions to the r -SHORTEST PATH COVER (r -SPC) problem, where $r > 0$, \mathcal{F} contains all shortest paths of length between r and $2r$, and \mathcal{B} contains all balls of radius $2r$. It is known that there is an XP algorithm that computes solutions to r -SPC of sparseness at most h if the input graph has highway dimension h . However it was not known whether a corresponding FPT algorithm exists as well. We prove that r -SPC and also the related r -HIGHWAY DIMENSION (r -HD) problem, which can be used to formally define the highway dimension of a graph, are both W[1]-hard. Furthermore, by the result of Abraham et al. [ICALP 2011] there is a polynomial-time $O(\log k)$ -approximation algorithm for r -HD, but for r -SPC such an algorithm is not known. We prove that r -SPC admits a polynomial-time $O(\log n)$ -approximation algorithm.

2012 ACM Subject Classification Theory of computation \rightarrow Fixed parameter tractability; Theory of computation \rightarrow Approximation algorithms analysis

Keywords and phrases sparse hitting set, fair vertex cover, highway dimension

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.5

Related Version *Full Version:* <https://arxiv.org/abs/2208.14132>



© Johannes Blum, Yann Disser, Andreas Emil Feldmann, Siddharth Gupta, and Anna Zych-Pawlewicz; licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 5; pp. 5:1–5:23

Leibniz International Proceedings in Informatics



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

Funding *Andreas Emil Feldmann*: Supported by the Czech Science Foundation GAČR (grant #19-27871X).

Siddharth Gupta: Supported by the Engineering and Physical Sciences Research Council (EPSRC) grant no: EP/V007793/1.

Anna Zych-Pawlewicz: This work is part of the project CUTACOMBS that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 714704).



1 Introduction

In this paper, we study the problem of finding a sparse hitting set. That is, we are given a set system $(V, \mathcal{F}, \mathcal{B})$ on universe V with two set families $\mathcal{F}, \mathcal{B} \subseteq 2^V$, and a feasible solution is a set $H \subseteq V$ that hits (i.e., intersects) every set of \mathcal{F} . Instead of minimizing the overall size of the solution however, we think of the sets of \mathcal{B} as being small and we would like to distribute the solution H among the sets in \mathcal{B} as evenly as possible. Intuitively and depending on the context, the sets in \mathcal{B} are balls in some metric and the hitting set should be sparse within them. That is, we want to find a hitting set for \mathcal{F} that minimizes the largest intersection with the sets of \mathcal{B} . Formally, the SPARSE HITTING SET (SPARSE-HS) problem with input $(V, \mathcal{F}, \mathcal{B})$ is defined by the following integer linear program (ILP) with indicator variables x_v for each $v \in V$ encoding membership in the solution $H \subseteq V$.

$$\begin{aligned} \min k \text{ such that: } \quad & \sum_{v \in F} x_v \geq 1 & \forall F \in \mathcal{F} & \quad (\text{SPARSE-HS-ILP}) \\ & \sum_{v \in B} x_v \leq k & \forall B \in \mathcal{B} \\ & x_v \in \{0, 1\} & \forall v \in V \end{aligned}$$

The SPARSE-HS problem generalizes several problems studied in the literature, with applications in for instance cellular [24], communication [25], and transportation [2, 22] networks. Our aim in this paper is to determine the complexity of some basic variants of SPARSE-HS, and we are specifically interested in the complexity depending on the *sparseness*, which is the solution value k of (SPARSE-HS-ILP). In general, SPARSE-HS contains the HITTING SET problem by setting $\mathcal{B} = \{V\}$, and thus does not admit any $g(k)$ -approximation in $f(k) \cdot n^{O(1)}$ time [21], for any computable functions f and g , where $n = |V|$, under ETH.

Sparse Vertex Cover. A much easier special case of HITTING SET is the well-known VERTEX COVER problem: for the SPARSE VERTEX COVER (SPARSE-VC) problem the set system is given by a graph $G = (V, E)$ so that $\mathcal{F} = E$ and $\mathcal{B} \subseteq 2^V$. We show that this problem is NP-hard for any $k \geq 2$, even on very simple input graphs.

► **Theorem 1.** *SPARSE-VC is NP-hard for any $k \geq 2$, even if the input graph is a matching.*

Note that this hardness result implies that, unless $P=NP$, SPARSE-VC does not admit an *XP algorithm* with runtime $n^{f(k)}$ for any function f (the problem is paraNP-hard parameterized by the sparseness k). This is in contrast to the VERTEX COVER problem, which is known to be *fixed-parameter tractable (FPT)* parameterized by the solution size s , which means that it can be solved much more efficiently in $f(s) \cdot n^{O(1)}$ time for some function f (which can be shown [10] to be 1.2738^s). On the other hand, we will show that for $k = 1$ the SPARSE-VC problem is polynomial-time solvable, which together with the previous hardness result settles the complexity of the SPARSE-VC problem for every sparseness value k .

► **Theorem 2.** *SPARSE-VC is polynomial time solvable for $k = 1$.*

As we will see, Theorem 1 also implies that SPARSE-VC does not admit a polynomial-time $(3/2 - \varepsilon)$ -approximation algorithm for any $\varepsilon > 0$, unless $P=NP$. Moreover, As VERTEX COVER is a special case of SPARSE-VC with $\mathcal{B} = \{V\}$, any polynomial time $(2 - \varepsilon)$ -approximation algorithm for SPARSE-VC would refute the Unique Games Conjecture (UGC) [28]. On the positive side, we show that we can match this conditional approximation lower bound with a 2-approximation algorithm. This means that SPARSE-VC can be approximated as well as the VERTEX COVER problem, which also admits a 2-approximation [28] in polynomial time.

► **Theorem 3.** *SPARSE-VC admits a polynomial time 2-approximation algorithm.*

Fair Vertex Cover. A special case of SPARSE-VC is the FAIR VERTEX COVER (FAIR-VC) problem where the family of sets \mathcal{B} is given by closed neighbourhoods, i.e., if $N[v]$ is the set containing vertex v and all neighbours of v in G then $\mathcal{B} = \{N[v] \mid v \in V\}$ (alternatively, \mathcal{B} contains all balls of radius 1). The fairness constraint was introduced by Lin and Sahni [25] in the context of communication networks, and has since then been studied for several types of problems (cf. Section 1.1), including VERTEX COVER [22, 18, 26]. In contrast to this paper, in [22, 18] the problem is defined slightly differently by considering open neighbourhoods, i.e., $\mathcal{B} = \{N[v] \setminus \{v\} \mid v \in V\}$, and we call this version OPEN-FAIR-VC. Notably, the parameterized complexity of OPEN-FAIR-VC has been studied for a plethora of parameters, including treedepth, treewidth, feedback vertex set, modular width [22], and the total solution size $|H|$ [18], and most of these results also apply to FAIR-VC with closed neighbourhoods.

Jacob et al. [18] observe that it is NP-hard to decide if a vertex cover of size s exists, if every neighbourhood is allowed to only contain at most k vertices of the solution H , for a given constant $k \geq 3$: this follows from the fact that VERTEX COVER is NP-hard on sub-cubic graphs [17]. While the authors of [18] call this problem FAIR VERTEX COVER as well, note that this is significantly different from the FAIR-VC problem studied in this paper as well as the OPEN-FAIR-VC problem studied in [22]. In particular, on sub-cubic graphs both of these problems as defined here always trivially have a solution for $k \geq 3$,¹ and thus the NP-hardness of VERTEX COVER on sub-cubic graphs does not immediately imply NP-hardness of FAIR-VC or OPEN-FAIR-VC. In fact, for the natural parameterization by the sparseness k the complexity of OPEN-FAIR-VC (and also FAIR-VC) has so far been unknown.² We answer this open problem by showing NP-hardness of FAIR-VC for $k \geq 3$ and of OPEN-FAIR-VC for $k \geq 4$ on more complex input graphs when compared to SPARSE-VC.

► **Theorem 4.** *FAIR-VC is NP-hard for any $k \geq 3$ and OPEN-FAIR-VC is NP-hard for any $k \geq 4$, even on planar input graphs.*

Thus, as for SPARSE-VC, we can conclude that FAIR-VC and OPEN-FAIR-VC do not admit XP algorithms parameterized by k . For the cases when $k \leq 2$, Jacob et al. [18] provide a polynomial time algorithm that solves their version of FAIR VERTEX COVER, which however also works for the FAIR-VC and OPEN-FAIR-VC problems as defined in this paper. Hence this settles the complexity of FAIR-VC for every value of k , and only leaves the value $k = 3$ open for OPEN-FAIR-VC.

¹ Observe that it is never necessary to pick a vertex v and all its neighbours.

² Tomáš Masařík, personal communication.

In terms of approximation, interestingly we are able to obtain a slightly better algorithm for FAIR-VC than for SPARSE-VC, namely a $(2 - \frac{1}{k})$ -approximation. This beats the best possible approximation for SPARSE-VC and VERTEX COVER under UGC [28]. In particular, the following theorem implies that for the smallest value $k = 3$ for which FAIR-VC is NP-hard, we can obtain a solution of sparseness 5. We leave open whether a solution of sparseness 4 can be computed in polynomial time for FAIR-VC if $k = 3$, and whether better approximation algorithms are possible for OPEN-FAIR-VC.

► **Theorem 5.** *FAIR-VC admits a polynomial time $(2 - \frac{1}{k})$ -approximation algorithm.*

Shortest Path Cover and Highway Dimension. We now turn to a different set of problems derived from SPARSE-HS, which as we shall see generalize FAIR-VC. Given a value $r > 0$ and an edge-weighted graph G , for the r -SHORTEST PATH COVER (r -SPC) problem the family \mathcal{F} is given by shortest paths of length between r and $2r$ and the family \mathcal{B} is given by balls of radius $2r$. That is, let \mathcal{P}_r contain $S \subseteq V$ if and only if S is the vertex set of a path in G , which is a shortest path (according to the edge weights) and whose length is in the range $(r, 2r]$. Furthermore, let $\text{dist}(u, v)$ be the length of a shortest u - v -path and let $B_r(v) = \{u \in V \mid \text{dist}(u, v) \leq r\}$ denote the ball of radius r centered at v . Then for the r -SPC problem, $\mathcal{F} = \mathcal{P}_r$ and $\mathcal{B} = \{B_{2r}(v) \mid v \in V\}$.

The r -SPC problem finds applications in the context of the *highway dimension*, which is a graph parameter introduced by Abraham et al. [2] to model transportation networks. To define the highway dimension, we define a problem related to r -SPC called r -HIGHWAY DIMENSION (r -HD), where for each vertex $v \in V$ the task is to find a hitting set for all shortest paths of length in $(r, 2r]$ intersecting the ball $B_{2r}(v)$, and we need to minimize the largest such hitting set. Note that compared to r -SPC the quantification is reversed, i.e., for r -SPC there is a hitting set that is small in every ball, while for r -HD for every ball there is a small hitting set (thus r -HD is not a special case of SPARSE-HS). The *highway dimension* of an edge-weighted graph G is the smallest integer h such that there is a solution to r -HD of value at most h in G for every $r > 0$.

There is empirical evidence [4] that road networks have small highway dimension, and it has been conjectured [13] that public transportation networks (especially those stemming from airplane networks) have small highway dimension as well.³ Therefore, there has been some effort to devise algorithms [13, 2, 12, 14, 11, 11, 5, 19, 8, 7] for problems on low highway dimension graphs that naturally arise in transportation networks. It is known [1] that if the highway dimension of a graph G is h , then the r -SPC problem on G has sparseness at most h for every $r > 0$, but not vice versa, as the sparseness of r -SPC can be much smaller than h . However, since a solution to r -SPC consists of one hitting set $H \subseteq V$ for the whole graph, it is more convenient to work with algorithmically than the n hitting sets for all balls of radius $2r$ that form a solution to r -HD. Therefore, algorithms exploiting the structure of graphs of low highway dimension typically compute a solution to the r -SPC problem for each of the $O(n^2)$ relevant values of r given by the pairwise distances between vertices.

For graphs of low highway dimension, Abraham et al. [1] give an algorithm that for each relevant value of r computes a solution to the r -HD problem, in order to obtain a solution to r -SPC with sparseness at most the value of the r -HD solution. While Abraham et al. [1] propose to use an approximation algorithm for r -HD (see below), note that the

³ In fact there are several definitions of the highway dimension, with the one presented here being well-suited for public transportation networks, cf. [13, 6]

r -HD problem admits an XP algorithm with runtime $n^{O(k)}$, since for any ball $B_{2r}(v)$ it can construct the set system given by all shortest paths of length in $(r, 2r]$ intersecting $B_{2r}(v)$, for which it can then try every possible k -tuple of vertices as a solution. This algorithm can thus be used to compute solutions to r -SPC of sparseness at most h in $n^{O(h)}$ time if the input graph has highway dimension h . Interestingly, it is not possible to compute solutions of optimum sparseness for r -SPC using an XP algorithm due to the NP-hardness of FAIR-VC: consider an r -SPC instance with unit edge weights and value $r = 1/2$. Since every edge is a shortest path between its endpoints, the r -SPC problem on this instance is equivalent to FAIR-VC. As argued above however, no XP algorithm exists for FAIR-VC, unless $P=NP$.

In light of the growing amount of work on problems on low highway dimension graphs, it would be very useful to have a faster algorithm to solve r -HD in order to compute a hitting set for r -SPC of corresponding sparseness. While it is known that computing the highway dimension is NP-hard [13] and this also implies that r -HD is NP-hard, r -HD might still be FPT and allow algorithms with runtime $f(k) \cdot n^{O(1)}$ for some function f . However, we will show that it is unlikely that such algorithms exist. In particular, we prove that r -HD is W[1]-hard parameterized by the solution value k . We also prove that r -SPC does not admit FPT algorithms (in particular, k here denotes the optimum sparseness and not just an upper bound that we would obtain by solving r -HD, as suggested above). While already the above reduction from FAIR-VC to r -SPC excludes FPT algorithms for r -SPC, this only excludes such algorithms for very small values of r , in fact the smallest relevant value for r (as the problem becomes trivial for even smaller values). A priori it is not clear whether r -SPC admits FPT (or XP) algorithms for large values of r . In our reduction however, the value of r takes the largest relevant value, so that there exists a ball of radius $2r$ containing the whole graph.

► **Theorem 6.** *Both r -HD and r -SPC are W[1]-hard parameterized by their solution values k , where $2r$ is the radius of the input graph.*

One caveat of this hardness result is that it does *not* answer the question of whether computing the highway dimension is FPT or not. This is because the presented reduction only shows hardness of r -HD for a large value r . However, for smaller values of r the solution value to r -HD is unbounded in the constructed graph, and thus the graph does not have bounded highway dimension. This means that it might still be possible to compute the highway dimension in FPT time, but not using the existing tools provided by Abraham et al. [1], where each value r is considered separately. Instead, if such an algorithm exists it must consider the structure of the whole graph. We leave open whether there is such an algorithm.

As mentioned above, Abraham et al. [1] propose an approximation algorithm for r -HD: under the assumption that all shortest paths are unique (which can always be achieved by slightly perturbing the edge lengths), r -HD admits a polynomial time $O(\log k)$ -approximation algorithm. Due to the fact that the sparseness of r -SPC can be a lot smaller than the solution value to r -HD,⁴ it is not known how to obtain such an algorithm for r -SPC. However, we prove the existence of a weaker $O(\log n)$ -approximation algorithm.

► **Theorem 7.** *r -SPC admits a polynomial time $O(\log n)$ -approximation algorithm.*

⁴ as for instance witnessed by the graphs constructed in the reduction for Theorem 4 and value $r = 1/2$.

Dense Matching. Finally, in light of the above results for SPARSE-VC, we also consider the dual DENSE MATCHING problem, where we are given a graph $G = (V, E)$ and the task is to find a matching $M \subseteq E$ maximizing the smallest number of matching edges induced by a set in the family \mathcal{B} , i.e., the minimum $|M \cap E(B)|$ over all $B \in \mathcal{B}$, where $E(B) = \{\{u, v\} \in E \mid u, v \in B\}$. Despite the MAXIMUM MATCHING problem being polynomial-time solvable, we show that DENSE MATCHING does not admit a polynomial time $(2 - \varepsilon)$ -approximation, even if \mathcal{B} is restricted to balls of radius two, unless $P=NP$. Interestingly, a matching 2-approximation seems a lot harder to come by compared to SPARSE-VC, and we leave open whether a constant approximation is possible for DENSE MATCHING.

► **Theorem 8.** *It is NP-hard to approximate DENSE MATCHING within $2 - \varepsilon$ for any $\varepsilon > 0$, even if $\mathcal{B} = \{B_2(v) \mid v \in V\}$ where all edges have weight 1.*

Due to space restrictions, the proof of Theorem 8 can be found in the appendix.

1.1 Related Work

Apart from the work cited above, we here list some additional related work. Kanesh et al. [20] study the FAIR FEEDBACK VERTEX SET problem, where the family \mathcal{F} contains all vertex sets of cycles of the input graph (in this case \mathcal{F} is not part of the input). They prove results on the parameterized complexity of several versions of this problem, where the considered parameters include treewidth, treedepth, neighbourhood diversity, the total solution size, and the maximum vertex degree. Jacob et al. [18] consider the parameterized complexity of the FAIR SET and FAIR INDEPENDENT SET problems, but also Π -FAIR VERTEX DELETION, where Π is any property expressible in first order (FO) logic. Knop et al. [22] study Π -FAIR VERTEX DELETION for properties Π expressible in monadic second order (MSO_1) logic parameterized by the twin cover number. They also consider FAIR-VC parameterized by treedepth, feedback vertex number, and modular width. Agrawal et al. [3] study the parameterized complexity of the MINIMUM MEMBERSHIP DOMINATING SET problem, where $\mathcal{F} = \mathcal{B} = \{N[v] \mid v \in V\}$, and consider parameterizations by pathwidth, sparseness, and vertex cover number.

While in this paper we study SPARSE-HS problems on graphs where the universe is the set of vertices, another line of work studies variants of SPARSE-HS when the universe is the edge set. For instance, the work of Lin and Sahni [25] that introduced the fairness constraint, studies the FAIR FEEDBACK EDGE SET problem, where the family \mathcal{F} contains the edge sets of all cycles of the input graph. Masařík and Toufar [26] consider the parameterized complexity of the Π -FAIR EDGE DELETION problem, where Π is a property expressible in FO logic or in MSO logic. For each of these problems they study parameterizations by the treewidth, pathwidth, treedepth, feedback vertex set number, neighbourhood diversity, and vertex cover number. Kolman et al. [23] study the Π -FAIR EDGE DELETION problem on graphs of bounded treewidth, where Π is any property expressible in MSO logic. They also give tight polynomial-time $O(\sqrt{n})$ -approximation algorithms for FAIR ODD CYCLE TRANSVERSAL and FAIR MIN CUT, where the family \mathcal{F} contains all edge sets of odd cycles and (s, t) -paths for given vertices s and t , respectively. Another notable problem is MIN DEGREE SPANNING TREE, where the family \mathcal{F} consists of every edge cut under the fairness constraint. Fürer and Raghavachari [15] prove that the problem is NP-hard but a solution of sparseness $k + 1$ can be computed in polynomial-time.

Regarding the complexity of computing the highway dimension, it is interesting to note that Abraham et al. [1] show that any set system given by unique shortest paths has VC-dimension 2 (this observation also leads to the above mentioned $O(\log k)$ -approximation

algorithm for r -HD). At the same time, Bringmann et al. [9] prove that the HITTING SET problem is W[1]-hard for set systems of VC-dimension 2. Hence it is intriguing to think that the latter reduction could possibly be modified to also prove W[1]-hardness for r -HD or r -SPC. However, it seems that shortest paths exhibit a lot more structure than general set systems of VC-dimension 2, and thus it is unclear how to obtain a hardness result for r -HD or r -SPC based on [9]. Instead, a more careful reduction as provided in Theorem 6 seems necessary.

2 Sparse Vertex Cover

In this section we consider the SPARSE-VC problem and start by proving NP-hardness for any $k \geq 2$.

► **Theorem 1.** *SPARSE-VC is NP-hard for any $k \geq 2$, even if the input graph is a matching.*

Proof. We reduce from a variant of the satisfiability problem called EXACTLY-3-SAT, meaning that all clauses contain exactly three literals. This problem was shown to be NP-complete in [16]. For a set of variables $X = \{x_i\}_i$, we use the notation $\bar{X} := \{\bar{x}_i\}_i$.

Let an instance of EXACTLY-3-SAT be given by a set of variables $X = \{x_i\}_{i=1,\dots,n}$ and a set of clauses $\mathcal{C} = \{C_j\}_{j=1,\dots,m}$ with $C_j \subset X \cup \bar{X}$, $|C_j| = 3$. We define the graph $G = (V, E)$ by $V = X \cup \bar{X} \cup \{y_i, \bar{y}_i \mid i \in \{1, \dots, k-1\}\}$ and $E = \{\{x_i, \bar{x}_i\} \mid i \in \{1, \dots, n\}\} \cup \{\{y_i, \bar{y}_i\} \mid i \in \{1, \dots, k-1\}\}$. We further let $\tilde{C} = C \cup \{y_1, \bar{y}_1, \dots, y_{k-2}, \bar{y}_{k-2}\}$ and choose $\mathcal{B} = \{\{x_i, \bar{x}_i, y_1, \bar{y}_1, \dots, y_{k-1}, \bar{y}_{k-1}\} \mid i \in \{1, \dots, n\}\} \cup \{\tilde{C} \mid C \in \mathcal{C}\}$. This construction can be carried out in linear time and G is a matching. For NP-hardness, it remains to show that G has a vertex cover $H \subseteq V$ satisfying $|H \cap B| \leq k$ for every $B \in \mathcal{B}$ if and only if the given EXACTLY-3-SAT instance has a satisfying assignment.

To see this, first assume that the given EXACTLY-3-SAT instance has a satisfying assignment $\alpha: X \rightarrow \{0, 1\}$ and extend α to \bar{X} by letting $\alpha(\bar{x}) := 1 - \alpha(x)$. We construct the vertex cover $H = \{x \in X \cup \bar{X} \mid \alpha(x) = 0\} \cup \{y_1, \dots, y_{k-1}\}$. Indeed, H is a vertex cover, since every edge of G is covered by exactly one of its endpoints. It also follows that, for every $i \in \{1, \dots, n\}$, we have $|\{x_i, \bar{x}_i, y_1, \bar{y}_1, \dots, y_{k-1}, \bar{y}_{k-1}\} \cap H| = k$. By definition of α , for every $C \in \mathcal{C}$, we have $\sum_{x \in C} \alpha(x) \geq 1$, hence $|H \cap \tilde{C}| = |H \cap C| + k - 2 = \sum_{x \in C} \alpha(\bar{x}) + k - 2 = 3 - \sum_{x \in C} \alpha(x) + k - 2 \leq k$.

Conversely, suppose that there exists a vertex cover $H \subseteq V$ with $|H \cap B| \leq k$ for all $B \in \mathcal{B}$. We claim that $\alpha(x) = |\{\bar{x}\} \cap H|$ defines a satisfying assignment for the given EXACTLY-3-SAT instance. Observe that we must have $|\{x_i, \bar{x}_i\} \cap H| \geq 1$ for all $i \in \{1, \dots, n\}$ and $|\{y_i, \bar{y}_i\} \cap H| \geq 1$ for all $i \in \{1, \dots, k-1\}$, since H needs to cover all edges. Since $\{x_i, \bar{x}_i, y_1, \bar{y}_1, \dots, y_{k-1}, \bar{y}_{k-1}\} \in \mathcal{B}$, it follows that $|\{x_i, \bar{x}_i, y_1, \bar{y}_1, \dots, y_{k-1}, \bar{y}_{k-1}\} \cap H| \leq k$ for all $i \in \{1, \dots, n\}$. Together, we obtain $|\{x_i, \bar{x}_i\} \cap H| = 1$ for all $i \in \{1, \dots, n\}$. We can therefore extend α to \bar{X} by setting $\alpha(\bar{x}) = 1 - \alpha(x) = |\{x\} \cap H|$. Finally, for $C \in \mathcal{C}$, we have $\tilde{C} \in \mathcal{B}$ and thus $|H \cap \tilde{C}| \leq k$ and moreover $|H \cap \tilde{C}| \geq |H \cap C| + k - 2$, which implies $|H \cap C| \leq 2$. It follows that $\sum_{x \in C} \alpha(x) = 3 - \sum_{x \in C} \alpha(\bar{x}) = 3 - \sum_{x \in C} |\{x\} \cap H| = 3 - |H \cap C| \geq 1$, thus α is a satisfying assignment. ◀

We can observe that Theorem 1 also shows that SPARSE-VC does not admit a $(3/2 - \varepsilon)$ -approximation algorithm for any $\varepsilon > 0$, unless P=NP. This follows from the fact that for $k = 2$, such an algorithm would be able to determine whether a given instance of SPARSE-VC admits a solution of sparseness $2 \cdot (3/2 - \varepsilon) < 3$, i.e., of optimal sparseness 2.

Let us now consider the SPARSE-VC problem for sparseness $k = 1$. We show that in this case, SPARSE-VC can be reduced to the 2-SAT problem, which is commonly known to admit a linear time algorithm. This yields the following theorem.

► **Theorem 2.** *SPARSE-VC is polynomial time solvable for $k = 1$.*

Proof. The instance of the SPARSE-VC problem is given by a graph $G = (V, E)$ and a set of balls $\mathcal{B} \subseteq 2^V$. Given this instance, we construct a 2-SAT formula ϕ , which is solvable if and only if the SPARSE-VC instance has a solution. Moreover, we can reconstruct the fair vertex cover for (V, E, \mathcal{B}) given a satisfying assignment to ϕ .

To construct ϕ , we first assign a variable x_v to each vertex $v \in V$. Next, for every edge $\{u, v\} \in E$ we create a clause $(x_u \vee x_v)$ and add it to ϕ , so that we are guaranteed that any satisfying assignment will correspond to a valid vertex cover. Now we have to enforce, that for each ball $B \in \mathcal{B}$, at most one variable in the set $\{x_v\}_{v \in B}$ is set to true. This is done by adding $\binom{|B|}{2}$ clauses: for each pair $v, u \in B, u \neq v$ we add a clause $(\bar{x}_v \vee \bar{x}_u)$ to enforce that x_v and x_u cannot be both true. In this way, we ensure that only one variable of the set $\{x_v\}_{v \in B}$ is set to true. Thus, the final formula ϕ takes the following form:

$$\phi = \bigwedge_{\{u,v\} \in E} (x_u \vee x_v) \wedge \bigwedge_{B \in \mathcal{B}, u,v \in B: u \neq v} (\bar{x}_v \vee \bar{x}_u)$$

Given a satisfying assignment for ϕ , we reconstruct the solution to (V, E, \mathcal{B}) by taking the vertices whose variable was set to true. We already argued that such a solution is feasible for SPARSE-VC with $k = 1$. In the opposite direction, given a solution to SPARSE-VC with $k = 1$, we find an assignment by setting the variables corresponding to the vertices of the solution to true: the clauses corresponding to edges are then satisfied due to the solution being a vertex cover, and the remaining clauses corresponding to \mathcal{B} are satisfied because the solution picks at most one vertex from each $B \in \mathcal{B}$. ◀

Finally, we show how to obtain a 2-approximation algorithm for SPARSE-VC. This approximation factor is optimal unless the Unique Games Conjecture fails, as VERTEX COVER is a special case of SPARSE-VC with $\mathcal{B} = \{V\}$.

► **Theorem 3.** *SPARSE-VC admits a polynomial time 2-approximation algorithm.*

Proof. We consider the relaxation of (SPARSE-HS-ILP) for a given graph $G = (V, E)$:

$$\min k \text{ such that: } \quad x_u + x_v \geq 1 \quad \forall uv \in E \quad (1)$$

$$\sum_{v \in B} x_v \leq k \quad \forall B \in \mathcal{B} \quad (2)$$

$$x_v \geq 0 \quad \forall v \in V \quad (3)$$

Note that in any feasible solution to this LP, for any edge $\{u, v\}$ at least one of the two variables x_u and x_v has value at least $1/2$, due to constraints (1) and (3). Thus the set $W = \{v \in V \mid x_v \geq 1/2\}$ of all vertices with value at least $1/2$, is a vertex cover for the input graph. The sparseness of this solution can be bounded using (2) for any set $B \in \mathcal{B}$:

$$|W \cap B| \leq 2 \sum_{v \in B} x_v \leq 2k$$

Thus solving the above LP relaxation optimally in polynomial time and then outputting the set W , gives a 2-approximation algorithm for SPARSE-VC. ◀

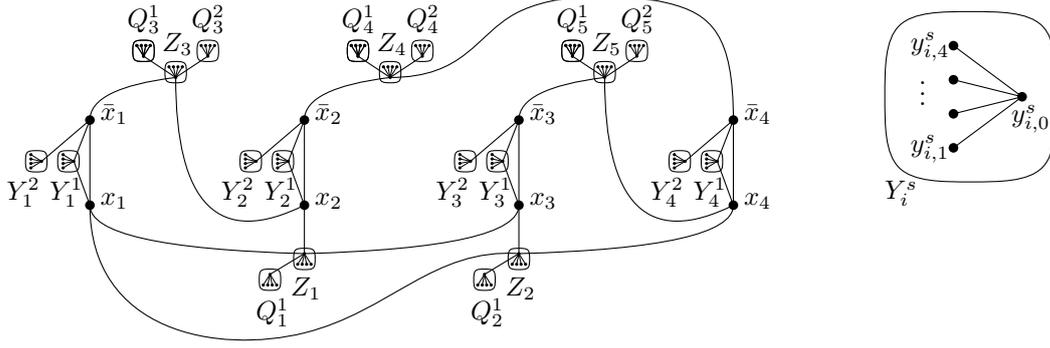


Figure 1 Left: The graph G for the formula $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee \bar{x}_4) \wedge (\bar{x}_3 \vee x_4)$ and $k = 4$. Right: A star Y_i^s with center $y_{i,0}^s$ and leaves $y_{i,1}^s, \dots, y_{i,4}^s$. Similarly, Z_j and Q_j^s denote stars with centers $z_{j,0}$ and $q_{j,0}^s$, and leaves $z_{j,1}, \dots, z_{j,4}$ and $q_{j,1}^s, \dots, q_{j,4}^s$, respectively.

3 Fair Vertex Cover

Let us now consider the (OPEN-)FAIR-VC problem, where the balls \mathcal{B} are given by (open) vertex neighborhoods. We first show NP-hardness of FAIR-VC and OPEN-FAIR-VC for $k \geq 3$ and $k \geq 4$, respectively.

► **Theorem 4.** *FAIR-VC is NP-hard for any $k \geq 3$ and OPEN-FAIR-VC is NP-hard for any $k \geq 4$, even on planar input graphs.*

Proof. We reduce from the PLANAR 2P1N-3-SAT problem. In this variant of satisfiability, all clauses contain two or three literals, and we may assume that every variable appears exactly twice as a positive literal and exactly once as a negative literal over all clauses. In addition, we may assume that the bipartite graph connecting clauses to the variables they contain is planar. This variant of satisfiability was shown to be NP-complete in [27].

We first consider the FAIR-VC problem and later show how to modify our reduction for OPEN-FAIR-VC. Let an instance of PLANAR 2P1N-3-SAT be given by a set of variables $X = \{x_i\}_{i=1, \dots, n}$ and a set of clauses $\mathcal{C} = \{C_j\}_{j=1, \dots, m}$ with $C_j \subset X \cup \bar{X}$, $|C_j| \in \{2, 3\}$. We define the graph $G = (V, E)$ by

$$V = \bigcup_{i=1}^n \left(\{x_i, \bar{x}_i\} \cup \bigcup_{s=1}^{k-2} \bigcup_{r=0}^k \{y_{i,r}^s\} \right) \cup \bigcup_{j=1}^m \left(\bigcup_{r=0}^k \{z_{j,r}\} \cup \bigcup_{s=1}^{k-|C_j|} \bigcup_{r=0}^k \{q_{j,r}^s\} \right)$$

and

$$E = \bigcup_{i=1}^n \left(\{x_i, \bar{x}_i\} \cup \bigcup_{s=1}^{k-3} \{x_i, y_{i,0}^s\} \cup \bigcup_{s=1}^{k-2} \{\bar{x}_i, y_{i,0}^s\} \cup \bigcup_{s=1}^{k-2} \bigcup_{r=1}^k \{y_{i,0}^s, y_{i,r}^s\} \right) \cup \bigcup_{j=1}^m \left(\bigcup_{x \in C_j} \{x, z_{j,0}\} \cup \bigcup_{r=1}^k \{z_{j,0}, z_{j,r}\} \cup \bigcup_{s=1}^{k-|C_j|} \left(\{z_{j,0}, q_{j,0}^s\} \cup \bigcup_{r=1}^k \{q_{j,0}^s, q_{j,r}^s\} \right) \right).$$

This construction (illustrated in Figure 1) can be carried out in linear time and G is planar.

For NP-hardness of FAIR-VC, it remains to show that G has a vertex cover $H \subset V$ satisfying $|H \cap N[v]| \leq k$ for every $v \in V$ if and only if the given PLANAR 2P1N-3-SAT instance has a satisfying assignment.

To see this, first assume that the given PLANAR 2P1N-3-SAT instance has a satisfying assignment $\alpha: X \rightarrow \{0, 1\}$ and extend α to \bar{X} by letting $\alpha(\bar{x}) := 1 - \alpha(x)$. We construct the vertex cover $H = \{x \in X \cup \bar{X} \mid \alpha(x) = 0\} \cup \bigcup_{i=1}^n \bigcup_{s=1}^{k-2} \{y_{i,0}^s\} \cup \bigcup_{j=1}^m \left(\{z_{j,0}\} \cup \bigcup_{s=1}^{k-|C_j|} \{q_{j,0}^s\} \right)$. Indeed, H is a vertex cover, since all edges are of the form $\{x, \bar{x}\}$ or incident to some $y_{i,0}^s$, $z_{j,0}$, or $q_{j,0}^s$. Also, $|H \cap N[v]| = 1$ for $v \in \bigcup_{i=1}^n \bigcup_{s=1}^{k-2} \bigcup_{r=1}^k \{y_{i,r}^s\} \cup \bigcup_{j=1}^m \bigcup_{r=1}^k \{z_{j,r}\} \cup \bigcup_{j=1}^m \bigcup_{s=1}^{k-|C_j|} \bigcup_{r=1}^k \{q_{j,r}^s\}$, $|H \cap N[v]| = 2$ for $v \in \bigcup_{i=1}^n \bigcup_{s=1}^{k-2} \{y_{i,0}^s\} \cup \bigcup_{j=1}^m \bigcup_{s=1}^{k-|C_j|} \{q_{j,0}^s\}$, and $|H \cap N[v]| = k$ for $v \in X \cup \bar{X}$. It remains to consider $v = z_{j,0}$ for some $j \in \{1, \dots, m\}$. Observe that $\sum_{x \in C_j} \alpha(x) \geq 1$ since α is a satisfying assignment. It holds that $|H \cap N[z_{j,0}]| = 1 + k - |C_j| + \sum_{x \in C_j} |\{x\} \cap H| = 1 + k - |C_j| + \sum_{x \in C_j} (1 - \alpha(x)) \leq 1 + k - |C_j| + |C_j| - 1 = k$. In either case, we conclude that $|H \cap N[v]| \leq k$.

Conversely, suppose that there exists a vertex cover $H \subseteq V$ with $|H \cap N[v]| \leq k$ for all $v \in V$. Let $i \in \{1, \dots, n\}$ and $s \in \{1, \dots, k-2\}$. Since H is a vertex cover with $|H \cap N[y_{i,0}^s]| \leq k$ and $\deg(y_{i,0}^s) > k$, we must have $y_{i,0}^s \in H$. Similarly, we must have $z_{j,0} \in H$ for all $j \in \{1, \dots, m\}$ and $q_{j,0}^s \in H$ for all $j \in \{1, \dots, m\}, s \in \{1, \dots, k-|C_j|\}$. We claim that $\alpha(x) = |\{\bar{x}\} \cap H|$ defines a satisfying assignment for the given PLANAR 2P1N-3-SAT instance. To see this, first observe that $|\{x, \bar{x}\} \cap H| = 1$ for all $x \in X$ since H is a vertex cover and $|H \cap N[x]| \leq k$. We can therefore extend α to \bar{X} by setting $\alpha(\bar{x}) = 1 - \alpha(x) = |\{x\} \cap H|$. Recall that for all $j \in \{1, \dots, m\}$ we have $|H \cap N[z_{j,0}]| \leq k$ and $\left\{ z_{j,0}, q_{j,0}^1, \dots, q_{j,0}^{k-|C_j|} \right\} \subseteq H$, which implies $\left| H \cap \left(N[z_{j,0}] \setminus \left\{ z_{j,0}, q_{j,0}^1, \dots, q_{j,0}^{k-|C_j|} \right\} \right) \right| \leq k - (1 + k - |C_j|) = |C_j| - 1$. With this, for $j \in \{1, \dots, m\}$, we have $\sum_{x \in C_j} \alpha(x) = |C_j| - \sum_{x \in C_j} \alpha(\bar{x}) = |C_j| - \sum_{x \in C_j} |\{x\} \cap H| \geq |C_j| - \left| \left(N[z_{j,0}] \setminus \left\{ z_{j,0}, q_{j,0}^1, \dots, q_{j,0}^{k-|C_j|} \right\} \right) \cap H \right| \geq |C_j| - (|C_j| - 1) = 1$. We conclude that α is a satisfying assignment.

We now turn to OPEN-FAIR-VC and modify the above reduction as follows. The first difference is that there is now only one Y_i^s gadget for each variable x_i , so we call it Y_i . The second difference is that Y_i is different from Y_i^s from the previous reduction, because now Y_i is responsible for picking only one vertex among $\{x_i, \bar{x}_i\}$ to the solution. To be more precise Y_i is now a star with a center $y_{i,0}$ connected to $y_{i,1}, \dots, y_{i,k-1}$, but now also each $y_{i,j}, j \in \{1, \dots, k-1\}$ is a center of a star, connected to $y_{i,1}^j, \dots, y_{i,k}^j$. In other words, Y_i is a tree of depth two rooted at $y_{i,0}$, where the root has $k-1$ children and each child of the root has k children. In the constructed graph, for each variable x_i both literals x_i and \bar{x}_i are now connected to $y_{i,0}$ instead of $y_{i,0}^s$ vertices from the previous reduction. The remaining part of the graph is precisely the same as in the reduction for Theorem 1.

Assume we have an OPEN-FAIR-VC solution for the constructed graph and the given parameter $k \geq 4$. Observe, that for each i the vertices $y_{i,0}$ and $y_{i,1} \dots y_{i,k-1}$ must be taken to the solution since their degree is $k+1$. Therefore, since vertex $y_{i,0}$ has $k-1$ neighbors other than x_i and \bar{x}_i , only one vertex among x_i and \bar{x}_i can be taken to the solution. The remaining part of the argument is the same as in the proof for FAIR-VC: we construct a satisfying assignment by setting this literal to false, whose corresponding vertex was taken to the solution. Similarly as before, each clause has at most two negated literals. The opposite direction is analogous. \blacktriangleleft

The previous reduction actually also shows that for any $\varepsilon > 0$, it is NP-hard to compute a $(4/3 - \varepsilon)$ -approximation for FAIR-VC, as for $k = 3$, a $(4/3 - \varepsilon)$ -approximate solution actually has sparseness 3. Still, we are able to compute a $(2 - \frac{1}{k})$ -approximation for FAIR-VC, which is slightly better than our result for SPARSE-VC and also better than the best possible approximation ratio for VERTEX COVER (and thus SPARSE-VC) under UGC. In particular, our algorithm implies that for the smallest value $k = 3$ for which FAIR-VC is NP-hard, we

can obtain a solution of sparseness 5. We leave open whether a solution of sparseness 4 can be computed in polynomial time for FAIR-VC if $k = 3$, and whether better approximation algorithms are possible for OPEN-FAIR-VC.

► **Theorem 5.** *FAIR-VC admits a polynomial time $(2 - \frac{1}{k})$ -approximation algorithm.*

Proof. As for the 2-approximation algorithm for SPARSE-VC (cf. Theorem 3), we consider the relaxation of (SPARSE-HS-ILP). However, in order to improve the approximation ratio, observe that in any solution of cost k to FAIR-VC, every vertex of degree more than k must be contained in the solution (otherwise some edge incident to such a vertex is not covered). Thus we may guess the optimum sparseness k^* , define the set of high degree vertices $D = \{v \in V \mid \deg(v) > k^*\}$ and add the constraint $x_v = 1$ for every $v \in D$ to the above LP relaxation. We again let W be the set of vertices with value at least $1/2$, which is a feasible vertex cover. If for any closed neighbourhood $N[v] \in \mathcal{B}$ we have $N[v] \subseteq W$ then all neighbours of v are contained in W , and thus we may remove v from W and still obtain a vertex cover. We repeat this iteratively for each vertex until we obtain a vertex cover W for which no closed neighbourhood is entirely contained in W . In particular, for any $v \notin D$ we have $|W \cap N[v]| \leq k^*$. For $v \in D$ on the other hand, since $x_v = 1$ we get

$$|W \cap N[v]| \leq |D \cap N[v]| + 2 \sum_{u \in N[v] \setminus D} x_u \leq (2x_v - 1) + 2 \sum_{u \in N[v] \setminus \{v\}} x_u \leq 2k - 1.$$

This means that the set W yields a $(2 - \frac{1}{k})$ -approximation. ◀

4 Hardness of Highway Dimension and Shortest Path Cover

In this section, we study the parameterized complexity of r -HD and r -SPC, and show the following theorem.

► **Theorem 6.** *Both r -HD and r -SPC are $W[1]$ -hard parameterized by their solution values k , where $2r$ is the radius of the input graph.*

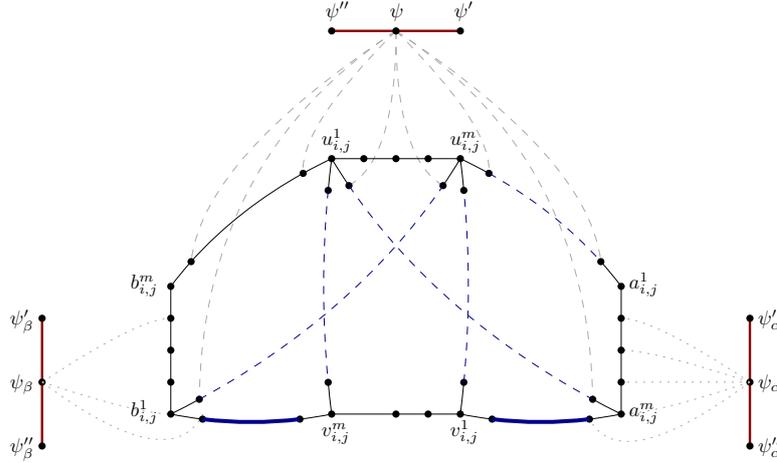
To prove this, we present a parameterized reduction from CLIQUE to r -HD. This reduction also shows $W[1]$ -hardness for r -SPC, as the constructed graph G has radius at most $2r$, i.e., any solution for r -HD is also a solution for r -SPC of the same cost and vice versa.

Let $H = (V, E)$ be a graph and let $k \in \mathbb{N}$. Denote the number of vertices and edges of H by n and m , respectively. For convenience we treat H as a bidirected graph, i.e. we replace every edge $\{u, v\} \in E$ with directed edges (u, v) and (v, u) . Let C be a constant whose value will be determined later on. We construct a graph G such that r -HD has a solution of value $k' = 4Ck(k - 1) + \binom{k}{2} + k + 3$ on G for $r = 2^m$ if and only if H contains a clique of size k . In the following, we call the individual elements of a solution for r -HD also *hubs*.

The graph G contains $k(k - 1)$ gadgets: For all $1 \leq i, j \leq k$ satisfying $i \neq j$ there is a gadget $G_{i,j}$. Choosing a certain set of hubs from $G_{i,j}$ means that $G_{i,j}$ represents a pair (w_i, w_j) of adjacent vertices of H . The idea of the reduction is to have a pair (w_i, w_j) from every $G_{i,j}$ such that

- (i) if $G_{i,j}$ represents (x, y) , then $G_{j,i}$ represents (y, x) , and
- (ii) if $G_{i,j}$ represents (x, y) and $G_{i,j'}$ represents (x', y') , then $x = x'$.

If these two conditions are fulfilled, it follows that there are k distinct vertices w_1, \dots, w_k which are pairwise adjacent, i.e. $\{w_1, \dots, w_k\}$ is a clique of size k .



■ **Figure 2** A gadget $G_{i,j}$ and the global vertices. Solid black edges have length 1, dashed blue edges have length $r - m + 1$, thick blue edges have length $r - 2m + 2$, dotted gray edges have length $m - 1$, dashed gray edges have length $r/2$ and red edges have length r .

Every gadget $G_{i,j}$ contains a path $u_{i,j}^1, \dots, u_{i,j}^m$, a path $v_{i,j}^1, \dots, v_{i,j}^m$, a path $a_{i,j}^1, \dots, a_{i,j}^m$, and a path $b_{i,j}^1, \dots, b_{i,j}^m$, each consisting of $m - 1$ edges of length 1. We identify every vertex of these paths with a pair (x, y) of adjacent vertices in H as follows: Fix any ordering \prec on V and denote the resulting lexicographic ordering on $V \times V$ also by \prec . Define $\tau: E \rightarrow \{1, \dots, m\}$ as

$$\tau(x, y) = |\{(u, v) \in E \mid (u, v) \prec (x, y)\}| + 1,$$

i.e. (x, y) is the $\tau(x, y)$ -th edge according to \prec . This allows us for instance to associate the vertex $u_{i,j}^{\tau(x,y)}$ of $G_{i,j}$ with the edge (x, y) of H .

The four paths are connected as follows. For $z \in \{a, v, b\}$ we connect $u_{i,j}^m$ with $z_{i,j}^1$ and $z_{i,j}^m$ with $u_{i,j}^1$, each through a path of length $r - m + 3$. To that end we introduce vertices $u_{i,j}^{0,z}, u_{i,j}^{m+1,z}, z_{i,j}^{0,u}$ and $z_{i,j}^{m+1,u}$, and add the edges $\{u_{i,j}^1, u_{i,j}^{0,z}\}, \{u_{i,j}^m, u_{i,j}^{m+1,z}\}, \{z_{i,j}^1, z_{i,j}^{0,u}\}, \{z_{i,j}^m, z_{i,j}^{m+1,u}\}$ of length 1 and the edges $\{u_{i,j}^{m+1,z}, z_{i,j}^{0,u}\}, \{z_{i,j}^{m+1,u}, u_{i,j}^{0,z}\}$ of length $r - m + 1$.

Moreover, we add vertices $a_{i,j}^{m+1,v}, v_{i,j}^{0,a}, v_{i,j}^{m+1,b}, b_{i,j}^{0,v}$ and add edges $\{a_{i,j}^m, a_{i,j}^{m+1,v}\}, \{v_{i,j}^1, v_{i,j}^{0,a}\}, \{v_{i,j}^m, v_{i,j}^{m+1,b}\}, \{b_{i,j}^1, b_{i,j}^{0,v}\}$ of length 1 and edges $\{a_{i,j}^{m+1,v}, v_{i,j}^{0,a}\}, \{v_{i,j}^{m+1,b}, b_{i,j}^{0,v}\}$ of length $r - 2m + 2$. This is illustrated in Figure 2.

The idea is that the shortest path from $u_{i,j}^1$ to any of $a_{i,j}^{0,u}, v_{i,j}^{0,u}$, and $b_{i,j}^{0,u}$ has length $r + 1$ and that we will have to choose some pair (x, y) in order to hit these shortest paths through the hub $u_{i,j}^{\tau(x,y)}$. Still, the shortest paths between $a_{i,j}^{0,u}$ and $b_{i,j}^{0,u}$ and between $a_{i,j}^{m+1,u}$ and $b_{i,j}^{m+1,u}$ both have length $2r - 2m + 4 > r$, but are not hit, if we choose, e.g., the hub $u_{i,j}^2$. Hence, we introduce a shorter path between $a_{i,j}^{0,u}$ and $b_{i,j}^{0,u}$ and between $a_{i,j}^{m+1,u}$ and $b_{i,j}^{m+1,u}$, which will be hit by a global dummy hub: We add vertices ψ, ψ', ψ'' and the edges $\{\psi', \psi\}$ and $\{\psi'', \psi\}$, both of length r . Moreover, we add edges between ψ and $a_{i,j}^{0,u}, a_{i,j}^{m+1,u}, b_{i,j}^{0,u}$, and $b_{i,j}^{m+1,u}$, each of length $r/2$. The shortest $a_{i,j}^{0,u}$ - $b_{i,j}^{0,u}$ - and $a_{i,j}^{m+1,u}$ - $b_{i,j}^{m+1,u}$ -paths now have length r and pass through ψ . Furthermore, the shortest ψ' - ψ'' -path has length $2r$ and we may assume w.l.o.g. that it is hit through the hub ψ .

We will show that if the final graph G admits a solution of value k' , then there is a hitting set for \mathcal{P}_r containing four vertices from every gadget $G_{i,j}$, which represent a pair (x, y) of adjacent vertices of H . Our construction needs to ensure that for these pairs (x, y) conditions i and ii are fulfilled. First we create C copies $G_{i,j}^1, \dots, G_{i,j}^C$ of the graph $G_{i,j}$. For simplicity,

we confuse the graphs $G_{i,j}$ and $G_{i,j}^\lambda$ for $\lambda \in \{1, \dots, C\}$ when the context is clear. Our final construction will yield that if $G_{i,j}^\lambda$ represents (x, y) and $G_{i,j}^{\lambda'}$ represents (x', y') , then we have $(x, y) = (x', y')$. Note that the vertices ψ, ψ' , and ψ'' are not part of any gadget $G_{i,j}$ and hence, we do not create copies of them.

For condition i we have to synchronise the gadgets $G_{i,j}$ and $G_{j,i}$. To that end, for all $1 \leq i < j \leq k$ and all $(x, y) \in E$ we add a vertex $\alpha_{i,j}^{(x,y)}$. Moreover, we add edges of weight m from (all C copies of) $a_{i,j}^{\tau(x,y)+1}$ and from (all C copies of) $a_{j,i}^{\tau(y,x)+1}$ to $\alpha_{i,j}^{(x,y)}$. This is illustrated in the appendix (Figure 3). The idea is that all shortest paths between $G_{i,j}$ and $G_{j,i}$ contained in \mathcal{P}_r can be hit with one additional hub $\alpha_{i,j}^{(x,y)}$ if both gadgets agree on the pairs (x, y) and (y, x) .

Still, the newly added edges of length m add new shortest paths to \mathcal{P}_r . For instance, in any $G_{i,j}$, the shortest path between $u_{i,j}^{m+1,a}$ and $\alpha_{i,j}^{\tau^{-1}(1)}$ has length $r + 2$. To ensure that it suffices to choose only $u_{i,j}^{\tau(x,y)}, v_{i,j}^{\tau(x,y)}$, and $\alpha_{i,j}^{(x,y)}$ as hubs, we remove these paths from \mathcal{P}_r by creating a new shortest path between $u_{i,j}^{m+1,a}$ and $\alpha_{i,j}^{\tau^{-1}(1)}$, which passes through the dummy hub ψ . To that end, for all $1 \leq i, j \leq k$ satisfying $i \neq j$ we add an edge between ψ and $u_{i,j}^{0,a}, u_{i,j}^{m+1,a}$ and all $\alpha_{i,j}^{(x,y)}$, each of length $r/2$.

Similarly, we avoid "undesired" hubs covering shortest paths across different gadgets $G_{i,j}$ and $G_{j,i}$ by introducing new vertices $\psi_\alpha, \psi'_\alpha$ and ψ''_α and adding the edges $\{\psi'_\alpha, \psi_\alpha\}$ and $\{\psi''_\alpha, \psi_\alpha\}$ of length r and an edge of length $m - 1$ between ψ_α and all $a_{i,j}^{\tau(x,y)+1}$.

To fulfill condition ii we have to synchronise the gadget $G_{i,j}$ with every other gadget $G_{i,j'}$. To that end, for all $1 \leq i \leq k$ and all $x \in V$ we add a vertex β_i^x . Let y_0, \dots, y_d be the neighbors of x such that $y_0 \prec \dots \prec y_d$. For $1 \leq i, j \leq k, i \neq j$ we add an edge of weight $m + d$ between β_i^x and every (copy of) $b_{i,j}^{\tau(x,y_0)-1}$. Here the idea is that if two gadgets $G_{i,j}$ and $G_{i,j'}$ represent pairs (x, y) and (x', y') such that $x = x'$, then choosing β_i^x as a hub suffices to hit all relevant shortest paths between the two gadgets.

Again, we have to take care of newly created shortest paths. Therefore we add an edge of length $r/2$ between ψ and $u_{i,j}^{0,b}, u_{i,j}^{m+1,b}$ and all β_i^x . Moreover we handle shortest paths across different gadgets $G_{i,j}$ and $G_{i,j'}$ by introducing new vertices ψ_β, ψ'_β and ψ''_β and adding the edges $\{\psi'_\beta, \psi_\beta\}$ and $\{\psi''_\beta, \psi_\beta\}$ of length r and an edge of length $m + d - 1$ between ψ_β and every $b_{i,j}^{\tau(x,y_d)-1}$ where y_d is the maximum neighbor of x according to \prec . This concludes the construction of the graph G , which is also illustrated in the appendix (Figure 4).

We now show several properties of the graph G which allow us to prove Theorem 6. The following Lemma states that choosing four hubs from some gadget $G_{i,j}$ means that the gadget represents a unique pair (x, y) . A proof can be found in the appendix.

► **Lemma 9.** *Let $1 \leq i, j \leq k$ where $i \neq j$ and let $\mathcal{H}_{i,j}$ be a hitting set for all shortest paths from \mathcal{P}_r that are contained in $G_{i,j}$. It holds that $|\mathcal{H}_{i,j}| \geq 4$ and moreover, if $|\mathcal{H}_{i,j}| = 4$, then $G_{i,j}$ represents some (x, y) , that is $\mathcal{H}_{i,j} = \{u_{i,j}^{\tau(x,y)}, a_{i,j}^{\tau(x,y)}, v_{i,j}^{\tau(x,y)}, b_{i,j}^{\tau(x,y)}\}$.*

Moreover, we show that if a gadget $G_{i,j}$ represents some pair (x, y) , then two certain shortest paths are not hit by the hubs of $G_{i,j}$. To that end, for any $(x, y) \in E$ and all $1 \leq i < j \leq k$ and $\lambda \in \{1, \dots, C\}$, let $A_{i,j}^{(x,y),\lambda}$ be the shortest path between $\alpha_{i,j}^{(x,y)}$ and the λ -th copy of $v_{i,j}^{\tau(x,y)-1}$. The length of $A_{i,j}^{(x,y),\lambda}$ is

$$\begin{aligned} \text{dist}(\alpha_{i,j}^{(x,y)}, a_{i,j}^{\tau(x,y)+1}) + \text{dist}(a_{i,j}^{\tau(x,y)+1}, a_{i,j}^{m+1,v}) + \text{dist}(a_{i,j}^{m+1,v}, v_{i,j}^{0,a}) + \text{dist}(v_{i,j}^{0,a}, v_{i,j}^{\tau(x,y)-1}) = \\ m + m - \tau(x, y) + r - 2m + 2 + \tau(x, y) - 1 = r + 1. \end{aligned}$$

Similarly, we define $B_{i,j}^{x,\lambda}$ as the shortest path between β_i^x and the λ -th copy of $v_{i,j}^{\tau(x,y_d)+1}$, where y_d is the maximum neighbor of x . The path $B_{i,j}^{x,\lambda}$ consists of a $v_{i,j}^{\tau(x,y_d)+1}$ - $v_{i,j}^{m+1,b}$ -path of length $m - \tau(x, y_d)$, the edge $\{v_{i,j}^{m+1,b}, b_{i,j}^{0,v}\}$ of length $r - 2m + 2$, a $b_{i,j}^{0,v}$ - $b_{i,j}^{\tau(x,y_0)-1}$ -path of length $\tau(x, y_d) - d - 1$ and the edge $\{b_{i,j}^{\tau(x,y_0)-1}, \beta_i^x\}$ of length $m + d$. The path $B_{i,j}^{x,\lambda}$ has length

$$\begin{aligned} \text{dist}(\beta_i^x, b_{i,j}^{\tau(x,y_0)-1}) + \text{dist}(b_{i,j}^{\tau(x,y_0)-1}, b_{i,j}^{0,v}) + \text{dist}(b_{i,j}^{0,v}, v_{i,j}^{m+1,b}) + \text{dist}(v_{i,j}^{m+1,b}, v_{i,j}^{\tau(x,y_d)+1}) = \\ m + d + \tau(x, y_d) - d - 1 + r - 2m + 2 + m - \tau(x, y_d) = r + 1. \end{aligned}$$

Moreover the following Lemma holds. An illustration (Figure 5) and a proof can be found in the appendix.

► **Lemma 10.** *If the gadget $G_{i,j}^\lambda$ represents the pair (x, y) , then the hubs of $G_{i,j}^\lambda$ hit the shortest path $A_{i,j}^{(x',y'),\lambda}$ if and only if $(x, y) \neq (x', y')$, and the shortest path $B_{i,j}^{x',\lambda}$ if and only if $x \neq x'$.*

Let us now prove Theorem 6. We show that on G , r -HD has a solution of value $k' = 4Ck(k-1) + \binom{k}{2} + k + 3$ for $r = 2^m$ if and only if H contains a clique of size k .

Proof of Theorem 6. Let $r = 2^m$. Suppose that on the constructed graph G , there is a solution of value k' for r -HD. We observe that every vertex has distance less than $2r$ from the vertex ψ . This means that $B_{2r}(\psi)$ contains the entire graph, and therefore there is a hitting set \mathcal{H} of size $|\mathcal{H}| \leq k'$ for \mathcal{P}_r .

We will prove that for any $1 \leq i, j \leq k$ there is some (x, y) such that the hitting set \mathcal{H} contains four hubs $u_{i,j}^{\tau(x,y)}, v_{i,j}^{\tau(x,y)}, a_{i,j}^{\tau(x,y)}, b_{i,j}^{\tau(x,y)}$ from every (copy of the) gadget $G_{i,j}$, and that \mathcal{H} contains one hub $\alpha_{i,j}^{x,y}$ for every $1 \leq i < j \leq k$ and one hub β_i^x for every $1 \leq i \leq k$, such that conditions i and ii are satisfied. This implies that H contains a clique of size k .

Fix now i, j such that $1 \leq i < j \leq k$. We prove that the hitting set \mathcal{H} contains some hub $\alpha_{i,j}^{(x,y)}$. Let $\lambda \in \{1, \dots, C\}$ and denote the vertices of \mathcal{H} that are contained in $G_{i,j}^\lambda$ by $\mathcal{H}_{i,j}^\lambda$. For the sake of contradiction suppose that there is no (x, y) such that $\alpha_{i,j}^{(x,y)} \in \mathcal{H}$. Lemma 9 states that if $|\mathcal{H}_{i,j}^\lambda| \leq 4$, then $\mathcal{H}_{i,j}^\lambda = \{u_{i,j}^{\tau(x,y)}, v_{i,j}^{\tau(x,y)}, a_{i,j}^{\tau(x,y)}, b_{i,j}^{\tau(x,y)}\}$ for some (x, y) and therefore \mathcal{H} does not hit the path $A_{i,j}^{(x,y),\lambda}$ according to Lemma 10. Hence, we obtain that for all $\lambda \in \{1, \dots, C\}$ we have $|\mathcal{H}_{i,j}^\lambda| \geq 5$. Moreover, Lemma 9 states that from any gadget $G_{i',j'}, (i', j') \neq (i, j)$ we have to choose at least four hubs. This means however that $|\mathcal{H}| \geq 4Ck(k-1) + C$. If we choose $C = k^2$ we obtain that $4Ck(k-1) + C > k'$, so it cannot be that there is no hub $\alpha_{i,j}^{(x,y)} \in \mathcal{H}$. Analogously we can show that \mathcal{H} contains some hub β_i^x for every $1 \leq i \leq k$, if $C = k^2$. To that end, fix $1 \leq i \leq k$ and suppose that there is no x such that $\beta_i^x \in \mathcal{H}$. Again, it follows from Lemmas 9 and 10 that for all $\lambda \in \{1, \dots, C\}$ we have $|\mathcal{H}_{i,j}^\lambda| \geq 5$, where $\mathcal{H}_{i,j}^\lambda$ denotes the vertices of \mathcal{H} that are contained in $G_{i,j}^\lambda$. As we showed previously, for $C = k^2$ this means that $|\mathcal{H}| > k'$, and it follows that there must be some x such that $\beta_i^x \in \mathcal{H}$.

This means that \mathcal{H} contains at least one hub $\alpha_{i,j}^{(x,y)}$ for every $1 \leq i < j \leq k$, at least one hub β_i^x for every $1 \leq i \leq k$ and at least four hubs from every $G_{i,j}^\lambda$. None of these hubs hits the shortest paths $\psi'' - \psi - \psi''$, $\psi'_\alpha - \psi_\alpha - \psi''_\alpha$, or $\psi'_\beta - \psi_\beta - \psi''_\beta$. To hit these three paths, we need three additional hubs. As \mathcal{H} has size at most $k' = 4Ck(k-1) + \binom{k}{2} + k + 3$, it follows that \mathcal{H} contains precisely four hubs from every $G_{i,j}^\lambda$, so every gadget represents indeed a unique pair (x, y) . Moreover, for every $1 \leq i < j \leq k$ there is a unique hub $\alpha_{i,j}^{(x,y)}$ and for every $1 \leq i \leq k$ there is a unique hub β_i^x .

It remains to show that the pairs represented by the individual gadgets fulfill properties i and ii. Consider i, j such that $1 \leq i < j \leq k$ and let $\lambda, \lambda' \in \{1, \dots, C\}$. Let (x, y) and (x', y') be the pairs represented by $G_{i,j}^\lambda$ and $G_{j,i}^{\lambda'}$, respectively. Lemma 10 states that the hubs contained in $G_{i,j}^\lambda$ and $G_{j,i}^{\lambda'}$ do not hit the shortest paths $A_{i,j}^{(x,y),\lambda}$ and $A_{j,i}^{(x',y'),\lambda'}$. This means that the two paths must be hit through the hubs $\alpha_{i,j}^{(x,y)}$ and $\alpha_{i,j}^{(x',y')}$. Moreover, both hubs must coincide as \mathcal{H} has size k' , i.e. we have $(x, y) = (x', y')$, which implies condition i.

For condition ii, let $1 \leq i \leq k$, let $1 \leq j, j' \leq k$, and let $\lambda, \lambda' \in \{1, \dots, C\}$. Denote the pairs represented by $G_{i,j}^\lambda$ and $G_{i,j'}^{\lambda'}$ by (x, y) and (x', y') , respectively. It follows from Lemma 10 that the shortest paths $B_{i,j}^{x,\lambda}$ and $B_{i,j'}^{x',\lambda'}$ are not hit through the hubs contained in $G_{i,j}^\lambda$ and $G_{i,j'}^{\lambda'}$. This means that the paths must be covered through hubs β_i^x and $\beta_i^{x'}$, and as $|\mathcal{H}| = k'$, this is only possible if $x = x'$, i.e. condition ii is satisfied.

This implies that the graph H indeed contains a clique of size k . To prove the other direction, we refer to the appendix. ◀

5 Approximating Shortest Path Covers

In this section, we show how to approximate r -SPC.

► **Theorem 7.** *r -SPC admits a polynomial time $O(\log n)$ -approximation algorithm.*

We present an algorithm based on the following ideas. It is well-known that the SET COVER problem is equivalent to HITTING SET by swapping the roles of the elements of the universe and the sets in the given set family. Kuhn et al. [24] study the MINIMUM MEMBERSHIP SET COVER (MMSC) problem, where the aim is to minimize the maximum *membership* of any element of the given universe of the SET COVER instance. Here the *membership* of an element is the number of sets of the solution it is contained in. The MMSC problem finds applications in interference minimization in cellular networks, and Kuhn et al. [24] prove that it admits a polynomial-time $O(\log |U|)$ -approximation, where U is the given universe, and they show that this is best possible, unless $P=NP$. Translated to SPARSE-HS, this means that for an instance where $\mathcal{F} = \mathcal{B}$, an $O(\log |\mathcal{F}|)$ -approximation can be computed in polynomial time, and this is also best possible, unless $P=NP$. We show that r -SPC can be reduced to this version of SPARSE-HS.

We first give a simple observation about the SPARSE-HS problem which will be useful later in our proof. Let $(V, \mathcal{F}, \mathcal{B})$ be a set system and let $B, B' \in \mathcal{B}$ be two sets such that $B \subsetneq B'$. If B' contains at most k elements of the hitting set, then B also contains at most k such elements. Hence we obtain the following.

► **Observation 11.** *Let \mathcal{B} be a family containing two sets B, B' such that $B \subsetneq B'$. If there exists a solution to SPARSE-HS for $(V, \mathcal{F}, \mathcal{B} \setminus \{B\})$ of sparseness k , then there exists a solution to SPARSE-HS for $(V, \mathcal{F}, \mathcal{B})$ of sparseness k .*

We reduce the r -SPC problem to the MINIMUM MEMBERSHIP SET COVER (MMSC) problem. Formally, an instance of MMSC consists of a universe U and a family \mathcal{S} of subsets of U , and the goal is to choose a set $\mathcal{S}' \subseteq \mathcal{S}$ such that every element in U belongs to at least one set in \mathcal{S}' and that the maximum membership of any element u with respect to \mathcal{S}' is minimal, where the membership of u is defined as the number of sets in \mathcal{S}' containing u .

Recall that, given a weighted graph $G = (V, E)$ and a radius $r > 0$, the r -SPC problem for G is equivalent to the SPARSE-HS problem on universe V with $\mathcal{F} = \mathcal{P}_r$ and $\mathcal{B} = \{B_{2r}(v) \mid v \in V\}$. Based on Observation 11, we first show that if there exists a ball $B \in \mathcal{B}$ which does not contain any shortest path in \mathcal{P}_r completely, we can safely remove it without affecting the solution.

► **Lemma 12.** *Let $B \in \mathcal{B}$ which does not contain any shortest path in \mathcal{P}_r completely, i.e., $S \not\subseteq B$ for every $S \in \mathcal{P}_r$. If there exists a solution to r -SPC for $(V, \mathcal{P}_r, \mathcal{B} \setminus \{B\})$ of sparseness k , then there exists a solution to r -SPC for $(V, \mathcal{P}_r, \mathcal{B})$ of sparseness k .*

Proof. First, if $S \cap B = \emptyset$ for every $S \in \mathcal{P}_r$, then there exists a solution for $(V, \mathcal{P}_r, \mathcal{B})$ not containing any vertices of B , and the claim follows. Now assume there is some path $S_B \in \mathcal{P}_r$ intersecting B in some vertex w . We show that there exists a ball $B' \in \mathcal{B}$ such that $B \subsetneq B'$, and thus the lemma follows from Observation 11.

Let v be the center of the ball $B = B_{2r}(v)$. As $S \not\subseteq B$ for every $S \in \mathcal{P}_r$, $\text{dist}(u, v) \leq r$ for every $u \in B$, as otherwise the shortest u - v -path would be contained in the ball B of radius $2r$ with a length in $(r, 2r]$, which would then be in \mathcal{P}_r . Hence for any two vertices $u, u' \in B$, $\text{dist}(u, u') \leq \text{dist}(u, v) + \text{dist}(v, u') \leq 2r$, so we have $B \subseteq B_{2r}(w)$. Moreover, it holds that $S_B \subseteq B_{2r}(w)$ as S_B is the vertex set of a path containing w of length in $(r, 2r]$, and it holds that $S_B \not\subseteq B$, which implies $B \subsetneq B_{2r}(w)$. By definition of \mathcal{B} we have $B_{2r}(w) \in \mathcal{B}$, and thus by Observation 11 the lemma follows. ◀

Lemma 12 means that we may assume w.l.o.g. that for any $B \in \mathcal{B}$ there is some $S_B \in \mathcal{P}_r$ such that $S_B \subseteq B$. We now give the following observations about the relationship among \mathcal{P}_r , \mathcal{B} , and a hitting set H of \mathcal{P}_r .

► **Observation 13.** *Let $S \in \mathcal{P}_r$. As S is the set of vertices of a shortest path π of length $\ell(\pi) \in (r, 2r]$, there exists a ball $B_S \in \mathcal{B}$ of radius $2r$, which completely contains S . This, in turn, implies that $H \cap B_S \neq \emptyset$ and $|H \cap S| \leq |H \cap B_S|$.*

► **Observation 14.** *Let $B \in \mathcal{B}$. If B contains some shortest path set $S_B \in \mathcal{P}_r$, then we have $H \cap B \neq \emptyset$ and $|H \cap S_B| \leq |H \cap B|$.*

By Observations 13 and 14, we get the following.

► **Lemma 15.** *There exists a solution to SPARSE-HS for $(V, \mathcal{P}_r, \mathcal{B})$ of sparseness k if and only if there exists a solution to SPARSE-HS for $(V, \mathcal{P}_r \cup \mathcal{B}, \mathcal{B} \cup \mathcal{P}_r)$ of sparseness k .*

Proof. Observe that any solution to SPARSE-HS for $(V, \mathcal{P}_r \cup \mathcal{B}, \mathcal{B} \cup \mathcal{P}_r)$ is also a solution to SPARSE-HS for $(V, \mathcal{P}_r, \mathcal{B})$, as $\mathcal{P}_r \subseteq \mathcal{P}_r \cup \mathcal{B}$ and $\mathcal{B} \subseteq \mathcal{B} \cup \mathcal{P}_r$. We now prove that any solution H to SPARSE-HS for $(V, \mathcal{P}_r, \mathcal{B})$ of sparseness k is also a solution to SPARSE-HS for $(V, \mathcal{P}_r \cup \mathcal{B}, \mathcal{B} \cup \mathcal{P}_r)$ of sparseness k . For this, we need to show that for every $S \in \mathcal{P}_r$, $|H \cap S| \leq k$ and for every $B \in \mathcal{B}$, $H \cap B \neq \emptyset$. The former statement follows from Observation 13, while the latter follows from Observation 14 where we assume that B contains some $S_B \in \mathcal{P}_r$ due to Lemma 12. ◀

We now define an instance of the MINIMUM MEMBERSHIP SET COVER with $U = \mathcal{P}_r \cup \mathcal{B}$ and $\mathcal{S} = \{S_u \mid u \in V\}$, where $S_u = \{S \in U \mid u \in S\}$, and prove the following.

► **Lemma 16.** *There exists a solution to SPARSE-HS for $(V, \mathcal{P}_r \cup \mathcal{B}, \mathcal{B} \cup \mathcal{P}_r)$ of sparseness k if and only if there exists a solution to MMSC for (U, \mathcal{S}) of value k .*

Proof. We will prove that if there exists a solution to SPARSE-HS for $(V, \mathcal{P}_r \cup \mathcal{B}, \mathcal{B} \cup \mathcal{P}_r)$ of sparseness k , then there exists a solution to MMSC for (U, \mathcal{S}) of value k . The proof for the other direction is symmetric. Let H be a solution to SPARSE-HS for $(V, \mathcal{P}_r \cup \mathcal{B}, \mathcal{B} \cup \mathcal{P}_r)$ of sparseness k . We claim that the set $W = \{S_u \in \mathcal{S} \mid u \in H\}$ is a solution to MMSC for (U, \mathcal{S}) of value k . Let $E \in U$. Then, $H \cap E \neq \emptyset$. Let $u \in H \cap E$. By the definition of S_u , this implies that $E \in S_u$. Moreover, for any $B \in \mathcal{P}_r \cup \mathcal{B}$, we have that $|H \cap B| \leq k$. This implies that B belongs to at most k sets in W . Hence, W is a solution to MMSC for (U, \mathcal{S}) of value k . ◀

Since there exists a $O(\log |U|)$ -approximation algorithm for MMSC by Kuhn et al. [24] and $|U| = |\mathcal{P}_r \cup \mathcal{B}| = O(n^2)$, by the above lemma we get an $O(\log n)$ -approximation algorithm for r -SPC. This concludes the proof of Theorem 7.

References

- 1 Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. Vc-dimension and shortest path algorithms. In *International Colloquium on Automata, Languages, and Programming*, pages 690–699. Springer, 2011.
- 2 Ittai Abraham, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 782–793. SIAM, 2010.
- 3 Akanksha Agrawal, Pratibha Choudhary, NS Narayanaswamy, KK Nisha, and Vijayaragunathan Ramamoorthi. Parameterized complexity of minimum membership dominating set. In *International Conference and Workshops on Algorithms and Computation*, pages 288–299. Springer, 2022.
- 4 Holger Bast, Stefan Funke, Domagoj Matijevic, Peter Sanders, and Dominik Schultes. In transit to constant time shortest-path queries in road networks. In *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 46–59. SIAM, 2007.
- 5 Amariah Becker, Philip N Klein, and David Saulpic. Polynomial-time approximation schemes for k-center, k-median, and capacitated vehicle routing in bounded highway dimension. In *26th Annual European Symposium on Algorithms (ESA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- 6 Johannes Blum. Hierarchy of transportation network parameters and hardness results. In *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*, 2019.
- 7 Martin Böhm, Ruben Hoeksma, Nicole Megow, Lukas Nölke, and Bertrand Simon. On hop-constrained steiner trees in tree-like metrics. *SIAM Journal on Discrete Mathematics*, 36(2):1249–1273, 2022.
- 8 Vladimir Braverman, Shaofeng H-C Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for clustering in excluded-minor graphs and beyond. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2679–2696. SIAM, 2021.
- 9 Karl Bringmann, László Kozma, Shay Moran, and NS Narayanaswamy. Hitting set for hypergraphs of low vc-dimension. In *24th Annual European Symposium on Algorithms (ESA 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
- 10 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved parameterized upper bounds for vertex cover. 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 238–249. Springer, 2006. doi:10.1007/11821069_21.
- 11 Yann Disser, Andreas Emil Feldmann, Max Klimm, and Jochen Könnemann. Travelling on graphs with small highway dimension. *Algorithmica*, 83(5):1352–1370, 2021. doi:10.1007/s00453-020-00785-5.
- 12 Andreas Emil Feldmann. Fixed-parameter approximations for k-center problems in low highway dimension graphs. *Algorithmica*, 81(3):1031–1052, 2019. doi:10.1007/s00453-018-0455-0.
- 13 Andreas Emil Feldmann, Wai Shing Fung, Jochen Könnemann, and Ian Post. A $(1+\epsilon)$ -embedding of low highway dimension graphs into bounded treewidth graphs. *SIAM Journal on Computing*, 47(4):1667–1704, 2018.
- 14 Andreas Emil Feldmann and David Saulpic. Polynomial time approximation schemes for clustering in low highway dimension graphs. *J. Comput. Syst. Sci.*, 122:72–93, 2021. doi:10.1016/j.jcss.2021.06.002.

- 15 Martin Fürer and Balaji Raghavachari. Approximating the minimum degree spanning tree to within one from the optimal degree. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 317–324, 1992.
- 16 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman, 1979.
- 17 Michael R Garey, David S Johnson, and Larry Stockmeyer. Some simplified np-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63, 1974.
- 18 Ashwin Jacob, Venkatesh Raman, and Vibha Sahlot. Deconstructing parameterized hardness of fair vertex deletion problems. In *International Computing and Combinatorics Conference*, pages 325–337. Springer, 2019.
- 19 Aditya Jayaprakash and Mohammad R Salavatipour. Approximation schemes for capacitated vehicle routing on graphs of bounded treewidth, bounded doubling, or highway dimension. *arXiv preprint*, 2021. [arXiv:2106.15034](https://arxiv.org/abs/2106.15034).
- 20 Lawqueen Kanesh, Soumen Maity, Komal Muluk, and Saket Saurabh. Parameterized complexity of fair feedback vertex set problem. *Theoretical Computer Science*, 867:1–12, 2021.
- 21 CS Karthik, Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. *Journal of the ACM*, 66(5), 2019.
- 22 Dusan Knop, Tomáš Masařík, and Tomáš Toufar. Parameterized complexity of fair vertex evaluation problems. In *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
- 23 Petr Kolman, Bernard Lidický, and Jean-Sébastien Sereni. Fair edge deletion problems on tree-decomposable graphs and improper colorings, 2010. Technical report.
- 24 Fabian Kuhn, Pascal von Rickenbach, Roger Wattenhofer, Emo Welzl, and Aaron Zollinger. Interference in cellular networks: The minimum membership set cover problem. In Lusheng Wang, editor, *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*, volume 3595 of *Lecture Notes in Computer Science*, pages 188–198. Springer, 2005.
- 25 Lishin Lin and Sartaj Sahni. Fair edge deletion problems. *IEEE transactions on computers*, 38(5):756–761, 1989.
- 26 Tomáš Masařík and Tomáš Toufar. Parameterized complexity of fair deletion problems. *Discrete Applied Mathematics*, 278:51–61, 2020.
- 27 J. Mañuch and D. R. Gaur. Fitting protein chains to cubic lattice is NP-complete. *Journal of Bioinformatics and Computational Biology*, 06.01:93–106, 2008.
- 28 David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

A Supplementary figures from Section 4

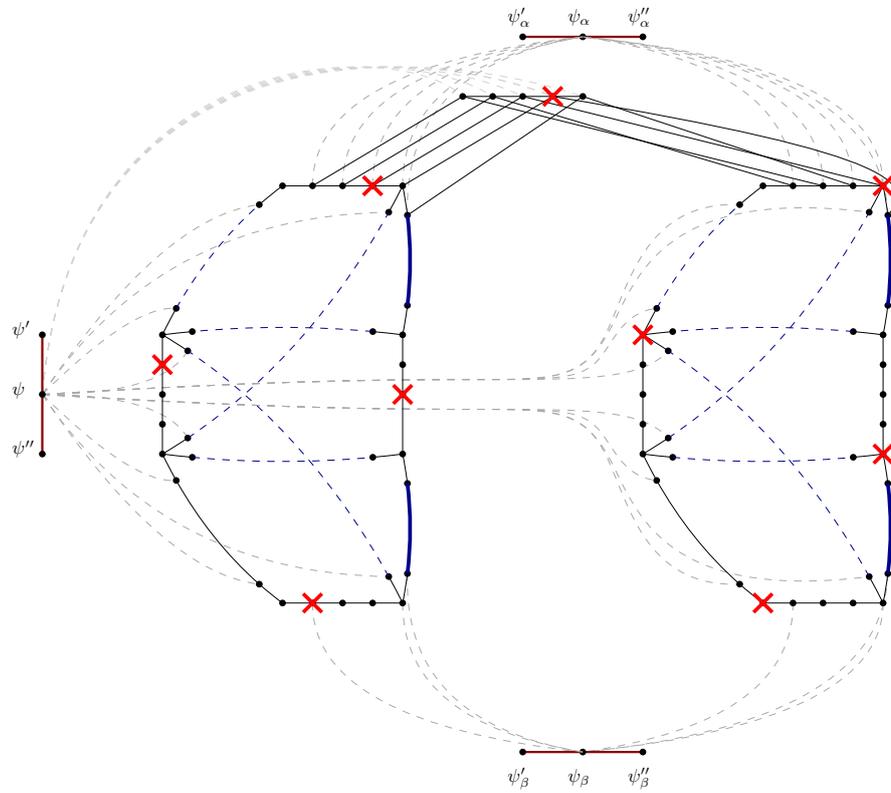


Figure 3 Two gadgets $G_{i,j}$ and $G_{j,i}$ and the connections between them. The marked vertices indicate that $G_{i,j}$ and $G_{j,i}$ represent the pairs (x,y) and (y,x) , respectively. Moreover, the vertex $\alpha_{i,j}^{(x,y)}$ is marked.

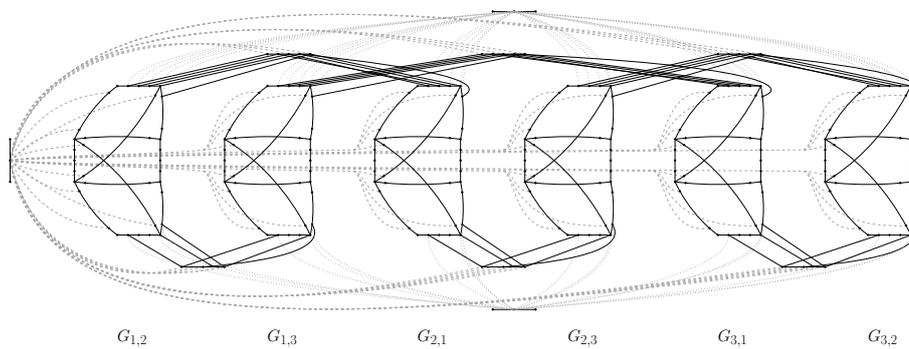
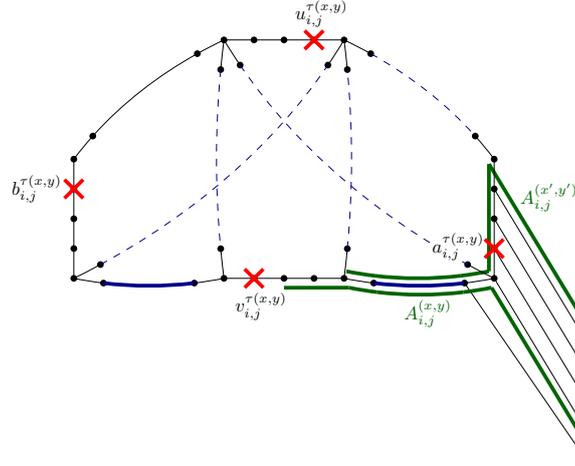


Figure 4 A (simplified) illustration of the whole construction. Note that only one copy $G_{i,j}^\lambda$ of every $G_{i,j}$ is shown.



■ **Figure 5** An illustration of Lemma 10. The gadget $G_{i,j}$ represents (x, y) , which means that the shortest path $A_{i,j}^{(x,y)}$ is not hit by the hubs of $G_{i,j}$, whereas any other shortest path $A_{i,j}^{(x',y')}$ is hit.

B Omitted proofs from Section 4

Proof of Lemma 9. Let $1 \leq \ell \leq m + 1$. For $z \in \{a, v, b\}$ we define the path $P^{uz}(\ell)$ as the shortest s - t -path path for

$$s = \begin{cases} u_{i,j}^{\ell} & \text{if } \ell \leq m \\ u_{i,j}^{m+1,z} & \text{else} \end{cases} \quad \text{and } t = \begin{cases} z_{i,j}^{\ell-1} & \text{if } \ell > 1 \\ z_{i,j}^{0,u} & \text{else} \end{cases}.$$

Similarly we define the path $P^{zu}(\ell)$ as the shortest s - t -path for

$$s = \begin{cases} z_{i,j}^{\ell} & \text{if } \ell \leq m \\ z_{i,j}^{m+1,u} & \text{else} \end{cases} \quad \text{and } t = \begin{cases} u_{i,j}^{\ell-1} & \text{if } \ell > 1 \\ u_{i,j}^{0,z} & \text{else} \end{cases}.$$

We observe that $P^{uz}(\ell)$ passes through the vertices $u_{i,j}^{m+1,z}$ and $z_{i,j}^{0,u}$, and that its length is $(m + 1 - \ell) + (r - m + 1) + (\ell - 1) = r + 1$. Similarly, $P^{zu}(\ell)$ passes through $z_{i,j}^{m+1,u}$ and $u_{i,j}^{0,z}$, and has also length $r + 1$. This means that both $P^{uz}(\ell)$ and $P^{zu}(\ell)$ need to be hit by $\mathcal{H}_{i,j}$. Consider the eight shortest paths $P^{ua}(1), P^{ua}(m + 1), P^{au}(1), P^{uv}(m + 1), P^{vu}(1), P^{bu}(1), P^{ub}(m + 1)$, and $P^{bu}(m + 1)$. It holds that every vertex of $G_{i,j}$ covers at most two of these paths, which implies $|\mathcal{H}_{i,j}| \geq 4$. To hit the shortest path $P^{uv}(m + 1)$ we have to choose one of the vertices $u_{i,j}^{m+1,v}, v_{i,j}^{0,u}, v_{i,j}^1, \dots, v_{i,j}^m$. However, the vertices $u_{i,j}^{m+1,v}$ and $v_{i,j}^{0,u}$ do not hit any of the other seven shortest paths. Hence, if we have $|\mathcal{H}_{i,j}| = 4$, then one of the four hubs must be the vertex $v_{i,j}^{\tau(x,y)}$ for some $(x, y) \in E$. Repeating the same argument for the paths $P^{au}(1), P^{ub}(m + 1)$, and $P^{ua}(1)$, one can show that if $|\mathcal{H}_{i,j}| = 4$, then $\mathcal{H}_{i,j}$ consists of four vertices $v_{i,j}^{\tau(x,y)}, a_{i,j}^{\tau(x',y')}, v_{i,j}^{\tau(x'',y'')}, b_{i,j}^{\tau(x''',y''')}$.

We now show that for these four vertices it holds that $(x, y) = (x', y') = (x'', y'') = (x''', y''')$. Suppose that for some (x, y) we have $v_{i,j}^{\tau(x,y)} \in \mathcal{H}_{i,j}$. Consider the two shortest paths $P^{uv}(\tau(x, y))$ and $P^{vu}(\tau(x, y))$. Our previous observations imply that both paths must be hit through some vertex $u_{i,j}^{\tau(x',y')}$. As $\mathcal{H}_{i,j}$ contains precisely one vertex $u_{i,j}^{\tau(x',y')}$ and as both paths intersect only in $u_{i,j}^{\tau(x,y)}$, it follows that $u_{i,j}^{\tau(x,y)} \in \mathcal{H}_{i,j}$. Similarly, it follows that $\mathcal{H}_{i,j}$ needs to contain the vertices $a_{i,j}^{\tau(x,y)}$ and $b_{i,j}^{\tau(x,y)}$. Hence, if $|\mathcal{H}_{i,j}| = 4$, it follows that there is a unique (x, y) such that $\mathcal{H}_{i,j} = \left\{ u_{i,j}^{\tau(x,y)}, a_{i,j}^{\tau(x,y)}, v_{i,j}^{\tau(x,y)}, b_{i,j}^{\tau(x,y)} \right\}$. ◀

Proof of Lemma 10. If the gadget $G_{i,j}^\lambda$ represents the pair (x, y) , then the hubs of $G_{i,j}^\lambda$ are $u_{i,j}^{\tau(x,y)}$, $v_{i,j}^{\tau(x,y)}$, $a_{i,j}^{\tau(x,y)}$, and $b_{i,j}^{\tau(x,y)}$. The shortest path $A_{i,j}^{(x',y'),\lambda}$ contains the vertices $v_{i,j}^{\tau(x',y')-1}, \dots, v_{i,j}^1$ and the vertices $a_{i,j}^m, \dots, a_{i,j}^{\tau(x',y')+1}$. This means that $A_{i,j}^{(x',y'),\lambda}$ is hit if and only if $(x, y) \neq (x', y')$. The shortest path $B_{i,j}^{x'}$ contains the vertices $v_{i,j}^{\tau(x',y_d)+1}, \dots, v_{i,j}^m$ and the vertices $b_{i,j}^1, \dots, a_{i,j}^{\tau(x',y_0)-1}$, which means that $B_{i,j}^{x'}$ is hit if and only if $x \neq x'$. ◀

Proof of Theorem 6 (continued). For the other direction suppose that the graph H contains a clique $\{w_1, \dots, w_k\}$ of size k . Consider the following set \mathcal{H} : For $1 \leq i, j \leq k, i \neq j$ it contains all C copies of the vertices $u_{i,j}^{\tau(w_i, w_j)}, v_{i,j}^{\tau(w_i, w_j)}, a_{i,j}^{\tau(w_i, w_j)}, b_{i,j}^{\tau(w_i, w_j)}$, for $1 \leq i < j \leq k$ it contains $\alpha_{i,j}^{(w_i, w_j)}$, for $1 \leq i \leq k$ it contains $\beta_{i,j}^{w_i}$, and moreover it contains the three vertices $\psi, \psi_\alpha, \psi_\beta$. It holds that \mathcal{H} has size k' .

We can observe that all shortest paths between different gadgets $G_{i,j}$ and $G_{i',j'}$ are hit by ψ_α or ψ_β . We now show that all shortest paths from \mathcal{P}_r that intersect only one gadget $G_{i,j}$ are hit by \mathcal{H} . Let $1 \leq i, j \leq k$ such that $i \neq j$. Suppose that $i < j$, the case $i > j$ can be shown similarly. Consider some vertex t contained in $G_{i,j}$ and denote the shortest path between $\alpha_{i,j}^{(x,y)}$ and t by P . Suppose that P is not hit by ψ . We can observe that the shortest path between $\alpha_{i,j}^{(x,y)}$ and $u_{i,j}^{m+1,a}$ or $b_{i,j}^{0,u}$ contains ψ . As P is not hit by ψ , it follows that

- (a) $t = a_{i,j}^\iota$ for some ι or $t \in \{a_{i,j}^{0,u}, a_{i,j}^{m+1,u}, a_{i,j}^{m+1,v}\}$,
- (b) $t = v_{i,j}^{\tau(x',y')}$ for some (x', y') , or
- (c) $t \in \{v_{i,j}^{0,a}, v_{i,j}^{0,u}, v_{i,j}^{m+1,u}, v_{i,j}^{m+1,b}\}$.

In case a it holds that P has length

$$\text{dist}(\alpha_{i,j}^{(x,y)}, a_{i,j}^{\tau(x,y)+1}) + \text{dist}(a_{i,j}^{\tau(x,y)+1}, t) \leq m + m + 1 = 2m + 1 < r,$$

so it does not need to be hit by \mathcal{H} . In case b, the length of P is

$$\begin{aligned} \text{dist}(\alpha_{i,j}^{(x,y)}, a_{i,j}^{\tau(x,y)+1}) + \text{dist}(a_{i,j}^{\tau(x,y)+1}, v_{i,j}^{0,a}) + \text{dist}(v_{i,j}^{0,a}, v_{i,j}^{\tau(x',y')}) = \\ m + m - \tau(x, y) + r - 2m + 2 + \tau(x', y') = r + 2 + \tau(x', y') - \tau(x, y). \end{aligned}$$

It follows that the length of P exceeds r if and only if $\tau(x', y') \geq \tau(x, y) - 1$, i.e. the path P contains $A_{i,j}^{(x,y)}$ as a subpath. Lemma 10 states that this subpath is hit by the hubs within $G_{i,j}$ if $(x, y) \neq (w_i, w_j)$, otherwise it is hit by $\alpha_{i,j}^{(x,y)}$. Finally, in case c it holds that P is shorter than r or that P contains $A_{i,j}^{(x,y)}$ as a subpath, which is hit by \mathcal{H} , as we just observed.

Analogously, consider some vertex t contained in $G_{i,j}$, denote the shortest path between β_i^x and t by P' and suppose that P' is not hit by ψ . As the shortest path between β_i^x and $u_{i,j}^{0,b}$ or $a_{i,j}^{m+1,u}$ contains ψ , it follows that

- (a) $t = b_{i,j}^\iota$ for some ι or $t \in \{b_{i,j}^{0,u}, b_{i,j}^{0,v}, b_{i,j}^{m+1,u}\}$,
- (b) $t = v_{i,j}^{\tau(x',y')}$ for some (x', y') , or
- (c) $t \in \{v_{i,j}^{0,a}, v_{i,j}^{0,u}, v_{i,j}^{m+1,u}, v_{i,j}^{m+1,b}\}$.

In case a it holds that P' is shorter than r . In case b, the length of P' is

$$\begin{aligned} \text{dist}(\beta_i^x, b_{i,j}^{\tau(x,y_0)-1}) + \text{dist}(b_{i,j}^{\tau(x,y_0)-1}, v_{i,j}^{m+1,b}) + \text{dist}(v_{i,j}^{m+1,b}, v_{i,j}^{\tau(x',y')}) = \\ m + d + \tau(x, y_0) - 1 + r - 2m + 2 + m + 1 - \tau(x', y') = \\ r + 2 + \tau(x, y_0) + d - \tau(x', y') = r + 2 + \tau(x, y_d) - \tau(x', y'). \end{aligned}$$

It holds that the length of P' exceeds r if and only if $\tau(x', y') \leq \tau(x, y_d) + 1$, which is the case if and only if P' contains $B_{i,j}^x$ as a subpath, which is hit by the hubs within $G_{i,j}$ or by $\beta_i^{(w_i, w_j)}$. In case c it holds that the length of P' is at most r or that P' contains $B_{i,j}^x$ as a subpath, which is hit by \mathcal{H} .

Consider now the shortest path between $u_{i,j}^{\tau(x,y)}$ and some vertex t of $G_{i,j}$ and denote it by P'' . Define the shortest paths $P^{uz}(\iota)$ and $P^{zu}(\iota)$ as in the proof of Lemma 9. If the length of P' exceeds r then for some $z \in \{a, v, b\}$, the path P'' contains the path $P^{uz}(\tau(x, y))$ to $z_{i,j}^{\tau(x,y)-1}$ or the path $P^{zu}(\tau(x, y) + 1)$ to $z_{i,j}^{\tau(x,y)+1}$ as a subpath. Suppose that P'' contains $P^{uz}(\tau(x, y))$, the other case is analogous. To show that $P^{uz}(\tau(x, y))$ (and hence also P'') is hit by \mathcal{H} , we distinguish two cases: If $(w_i, w_j) \prec (x, y)$, then $P^{uz}(\tau(x, y))$ is hit through $z_{i,j}^{\tau(w_i, w_j)-1}$, otherwise it is hit through $u_{i,j}^{\tau(w_i, w_j)-1}$. Similarly it can be shown that any shortest path between two vertices of $G_{i,j}$ whose length exceeds r is hit by \mathcal{H} , which means that \mathcal{H} is a solution for r -HD on G of value k' . \blacktriangleleft

C Dense Matching

► **Theorem 8.** *It is NP-hard to approximate DENSE MATCHING within $2 - \varepsilon$ for any $\varepsilon > 0$, even if $\mathcal{B} = \{B_2(v) \mid v \in V\}$ where all edges have weight 1.*

Proof. Consider the following reduction from 3-SAT. Let an instance of this problem be given by a set of variables $X = \{x_i\}_{i=1, \dots, n}$ and a set of clauses $\mathcal{C} = \{C_j\}_{j=1, \dots, m}$ with $C_j \subset X \cup \bar{X}$, $|C_j| \leq 3$. Let $\bar{x} = x$. We construct the graph $G = (V, E)$ given by

$$V = \bigcup_{i=1}^n (\{x_i, \bar{x}_i, x^0\} \cup \{x_i^\ell, \bar{x}_i^\ell \mid 1 \leq \ell \leq 7\}) \cup \bigcup_{j=1}^m \left(\{z_j\} \cup \bigcup_{x \in C_j} \{x^{j,\ell} \mid 1 \leq \ell \leq 4\} \right)$$

and

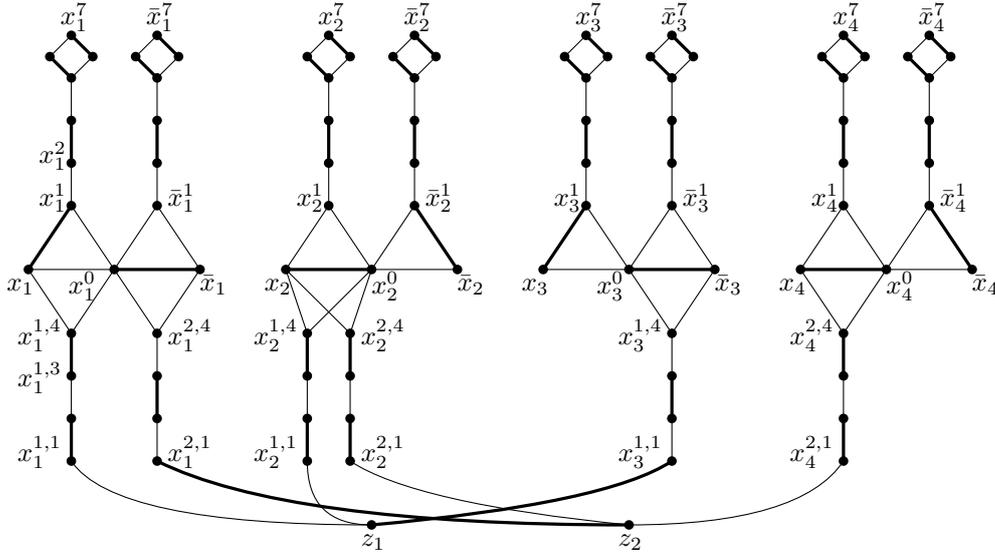
$$E = \bigcup_{i=1}^n (\{\{x_i, x_i^0\}, \{x_i, x_i^1\}, \{x_i^1, x_i^0\}, \{\bar{x}_i, x_i^0\}, \{\bar{x}_i, \bar{x}_i^1\}, \{\bar{x}_i^1, x_i^0\}\}) \cup \bigcup_{i=1}^n \left(\bigcup_{\ell=1}^6 \{\{x_i^\ell, x_i^{\ell+1}\}, \{\bar{x}_i^\ell, \bar{x}_i^{\ell+1}\}\} \cup \{\{x_i^7, x_i^4\}, \{\bar{x}_i^7, \bar{x}_i^4\}\} \right) \cup \bigcup_{j=1}^m \bigcup_{x \in C_j} \{\{z_j, x^{j,1}\}, \{x^{j,1}, x^{j,2}\}, \{x^{j,2}, x^{j,3}\}, \{x^{j,3}, x^{j,4}\}, \{x^{j,4}, x\}, \{x^{j,4}, x^0\}\}$$

The construction is illustrated in Figure 6.

We now show that the given 3-SAT formula is satisfiable if and only if there is a matching M such that $|M \cap E(B_2(v))| \geq 2$ for every ball $B_2(v)$ of radius 2, where we assume that edges have unit length. This means that if the given formula is not satisfiable, then there is a ball $B_2(v)$ such that $|M \cap E(B_2(v))| \leq 1$, which implies that it is NP-hard to obtain an approximation factor less than two.

Suppose that the given formula has a satisfying assignment $\alpha: X \rightarrow \{0, 1\}$ and extend α to \bar{X} by choosing $\alpha(\bar{x}) = 1 - \alpha(x)$. For $j = 1 \dots m$ let $y_j \in C_j$ be some literal satisfying C_j , i.e. $\alpha(y_j) = 1$. We construct the matching

$$M = \bigcup_{i=1}^n \{\{x_i^2, x_i^3\}, \{x_i^4, x_i^5\}, \{x_i^6, x_i^7\}, \{\bar{x}_i^2, \bar{x}_i^3\}, \{\bar{x}_i^4, \bar{x}_i^5\}, \{\bar{x}_i^6, \bar{x}_i^7\}\} \cup \bigcup_{x: \alpha(x)=1} \{\{x, x^0\}\} \cup \bigcup_{x: \alpha(x)=0} \{\{x, x^1\}\} \cup \bigcup_{j=1}^m \left(\{\{z_j, y_j^{j,1}\}, \{y_j^{j,2}, y_j^{j,3}\}\} \cup \bigcup_{x \in C_j \setminus \{y_j\}} \{\{x^{j,1}, x^{j,2}\}, \{x^{j,3}, x^{j,4}\}\} \right)$$



■ **Figure 6** The graph G for the formula $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$. The bold edges yield a matching that corresponds to the assignment $x_1 \mapsto 0, x_2 \mapsto 1, x_3 \mapsto 0, x_4 \mapsto 1$.

It is easy to verify that M is indeed a matching and that $|M \cap E(B_2(v))| \geq 2$ for every ball $B_2(v)$.

Suppose now that there is some matching M such that $|M \cap E(B_2(v))| \geq 2$ for every ball $B_2(v)$. Consider the assignment $\alpha: X \rightarrow \{0, 1\}, \alpha(x) = 1$ if and only if there is some j such that $\{z_j, x^{j,1}\} \in M$. To show that α is a satisfying assignment, consider some clause C_j and let $C_j = \{x_{i_1}, x_{i_2}, x_{i_3}\}$. Consider the ball $B_2(x_{i_1}^{j,1}) = \{z_j, x_{i_1}^{j,1}, x_{i_1}^{j,2}, x_{i_1}^{j,3}, x_{i_2}^{j,1}, x_{i_3}^{j,1}\}$. We show that M contains one of the edges $\{z_j, x_{i_1}^{j,1}\}, \{z_j, x_{i_2}^{j,1}\}$, and $\{z_j, x_{i_3}^{j,1}\}$. To prove this, suppose that M contains none of these three edges. This means that M has to contain the two remaining edges $\{x_{i_1}^{j,1}, x_{i_1}^{j,2}\}$ and $\{x_{i_1}^{j,2}, x_{i_1}^{j,3}\}$ contained in $E(B_2(x_{i_1}^{j,1}))$, which is not possible as both edges are incident to $x_{i_1}^{j,2}$.

Let now $\{z_j, x_{i_1}^{j,1}\}$ be the edge contained in M . If $x_{i_1} \in X$, i.e. x_{i_1} is a positive literal, it immediately follows that α satisfies the clause C_j . Suppose now that $x_{i_1} \in \bar{X}$. We show that in this case we have $\alpha(x_{i_1}) = 0$, i.e. there is no j' such that $\{z_{j'}, \bar{x}_{i_1}^{j',1}\} \in M$. For the sake of contradiction, suppose that $\{z_{j'}, \bar{x}_{i_1}^{j',1}\} \in M$ for some $j' \in \{1, \dots, m\}$. It follows from $|M \cap E(B_2(x_{i_1}^{j,1}))| \geq 2$ that $\{x_{i_1}^{j,2}, x_{i_1}^{j,3}\} \in M$. Consider the ball $B_2(x_{i_1}^{j,3}) = \{x_{i_1}^{j,1}, x_{i_1}^{j,2}, x_{i_1}^{j,3}, x_{i_1}^{j,4}, x_{i_1}, \bar{x}_{i_1}^0\}$. As M contains two edges from this ball and one of these edges is $\{x_{i_1}^{j,2}, x_{i_1}^{j,3}\}$, the other edge needs to be contained in the triangle $\{x_{i_1}^{j,4}, x_{i_1}, \bar{x}_{i_1}^0\}$. Consider now the ball $B_2(\bar{x}_{i_1}^0) = \{x_{i_1}^4, x_{i_1}^5, x_{i_1}^6, \bar{x}_{i_1}^0\}$. It holds that $\{x_{i_1}^4, x_{i_1}^5\} \in M$ or $\{x_{i_1}^4, x_{i_1}^6\} \in M$, which implies $\{x_{i_1}^3, x_{i_1}^4\} \notin M$. If we now consider the ball $B_2(x_{i_1}^2) = \{x_{i_1}, \bar{x}_{i_1}^0, x_{i_1}^1, x_{i_1}^2, x_{i_1}^3, x_{i_1}^4\}$, it follows that M needs to contain one edge from the triangle $\{x_{i_1}, \bar{x}_{i_1}^0, x_{i_1}^1\}$. As M also contains one edge from the triangle $\{x_{i_1}^4, x_{i_1}, \bar{x}_{i_1}^0\}$, we obtain that M contains $\{\bar{x}_{i_1}^0, x_{i_1}^4\}, \{\bar{x}_{i_1}^0, x_{i_1}\}$, or $\{x_{i_1}^0, x_{i_1}^1\}$.

However, as we also have $\{z_{j'}, \bar{x}_{i_1}^{j',1}\} \in M$, it follows analogously that M contains $\{\bar{x}_{i_1}^0, \bar{x}_{i_1}^4\}, \{\bar{x}_{i_1}^0, \bar{x}_{i_1}\}$, or $\{\bar{x}_{i_1}^0, \bar{x}_{i_1}^1\}$, which is not possible. This means that α is indeed a satisfying assignment, which concludes the proof. ◀

On the Complexity of Problems on Tree-Structured Graphs

Hans L. Bodlaender  

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Carla Groenland  

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Hugo Jacob  

ENS Paris-Saclay, France

Marcin Pilipczuk  

University of Warsaw, Poland

Michał Pilipczuk  

University of Warsaw, Poland

Abstract

In this paper, we introduce a new class of parameterized problems, which we call XALP: the class of all parameterized problems that can be solved in $f(k)n^{O(1)}$ time and $f(k)\log n$ space on a non-deterministic Turing Machine with access to an auxiliary stack (with only top element lookup allowed). Various natural problems on “tree-structured graphs” are complete for this class: we show that LIST COLORING and ALL-OR-NOTHING FLOW parameterized by treewidth are XALP-complete. Moreover, INDEPENDENT SET and DOMINATING SET parameterized by treewidth divided by $\log n$, and MAX CUT parameterized by cliquewidth are also XALP-complete.

Besides finding a “natural home” for these problems, we also pave the road for future reductions. We give a number of equivalent characterisations of the class XALP, e.g., XALP is the class of problems solvable by an Alternating Turing Machine whose runs have tree size at most $f(k)n^{O(1)}$ and use $f(k)\log n$ space. Moreover, we introduce “tree-shaped” variants of WEIGHTED CNF-SATISFIABILITY and MULTICOLOR CLIQUE that are XALP-complete.

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

Keywords and phrases Parameterized Complexity, Treewidth, XALP, XNLP

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.6

Related Version *Full Version*: <https://arxiv.org/abs/2206.11828> [8]

Funding *Carla Groenland*: Supported by the European Union’s Horizon 2020 research and innovation programme under the ERC grant CRACKNP (number 853234) and the Marie Skłodowska-Curie grant GRAPHCOSY (number 101063180).

Marcin Pilipczuk: This research is a part of a project that have received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme Grant Agreement 714704.

Michał Pilipczuk: This research is a part of a project that have received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme Grant Agreement 948057.

Acknowledgements We would like to thank the organizers of the workshop on Parameterized complexity and discrete optimization, organized at HIM in Bonn, for providing a productive research environment. We would also like to thank our referees for useful suggestions.



© Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Marcin Pilipczuk, and Michał Pilipczuk; licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 6; pp. 6:1–6:17

Leibniz International Proceedings in Informatics



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



1 Introduction

A central concept in complexity theory is *completeness* for a class of problems. Establishing completeness of a problem for a class pinpoints its difficulty, and gives implications on resources (time, memory or otherwise) to solve the problem (often, conditionally on complexity theoretic assumptions). The introduction of the W-hierarchy by Downey and Fellows in the 1990s played an essential role in the analysis of the complexity of parameterized problems [13, 14, 15]. Still, several problems are suspected not to be complete for a class in the W-hierarchy, and other classes of parameterized problems with complete problems were introduced, e.g., the A-, AW-, and M-hierarchies. (See e.g., [1, 15, 19].) In this paper, we introduce a new class of parameterized complexity, which appears to be the natural home of several “tree structured” parameterized problems. This class, which we call XALP, can be seen as the parameterized version of a class known in classic complexity theory as $\text{NAuxPDA}[\text{poly}, \log]$ (see [3]), or $\text{ASPSZ}(\log n, n^{O(1)})$ [24].

It can also be seen as the “tree variant” of the class XNLP, which is the class of parameterized problems that can be solved by a non-deterministic Turing machine using $f(k) \log n$ space in $f(k)n^{O(1)}$ time for some computable function f , where k denotes the parameter and n the input size. It was introduced in 2015 by Elberfeld et al. [17]. Recently, several parameterized problems were shown to be complete for XNLP [4, 9, 7]; in this collection, we find many problems for “path-structured graphs”, including well known problems that are in XP with pathwidth or other linear width measures as parameter, and linear ordering graph problems like BANDWIDTH.

Thus, we can view XALP as the “tree” variant of XNLP and as such, we expect that many problems known to be in XP (and expected not to be in FPT) when parameterized by treewidth will be complete for this class. We will prove the following problems to be XALP-complete in this paper:

- LIST COLORING and ALL-OR-NOTHING FLOW parameterized by treewidth;
- INDEPENDENT SET and DOMINATING SET parameterized by treewidth divided by $\log n$, where n is the number of vertices of the input graph;
- MAX CUT parameterized by cliquewidth.

The problems listed in this paper should be regarded as examples of a general technique, and we expect that many other problems parameterized by treewidth, cliquewidth and similar parameters will be XALP-complete. In many cases, a simple modification of an XNLP-hardness proof with pathwidth as parameter shows XALP-hardness for the same problem with treewidth as parameter.

In addition to pinpointing the exact complexity class for these problems, such results have further consequences. First, XALP-completeness implies XNLP-hardness, and thus hardness for all classes $W[t]$, $t \in \mathbb{N}$. Second, a conjecture by Pilipczuk and Wrochna [23], if true, implies that every algorithm for an XALP-complete problem that works in XP time (that is, $n^{f(k)}$ time) cannot simultaneously use FPT space (that is, $f(k)n^{O(1)}$ space). Indeed, typical XP algorithms for problems on graphs of bounded treewidth use dynamic programming, with tables that are of size $n^{f(k)}$.

Satisfiability on graphs of small treewidth. Real-world SAT instances tend to have a special structure to them. One of the measures capturing the structure is the *treewidth* $\mathcal{TW}(\phi)$ of the given formula ϕ . This is defined by taking the treewidth of an associated graph, usually a bipartite graph on the variables on one side and the clauses on the other, where there is an edge if the variable appears in the clause. Alekhovitch and Razborov [2]

raised the question of whether satisfiability of formulas of small treewidth can be checked in polynomial space, which was positively answered by Allender et al. [3]. However, the running time of the algorithm is $3^{\mathcal{TW}(\phi) \log |\phi|}$ rather than $2^{O(\mathcal{TW}(\phi))} |\phi|^{O(1)}$, where $|\phi| = n + m$ for n the number of variables and m the number of clauses. They also conjectured that the $\log |\phi|$ factor in the exponent for the running time cannot be improved upon without using exponential space.

To support this conjecture, Allender et al. [3] show that SATISFIABILITY where the treewidth of the associated graph is $O(\log n)$ is complete for a class of problems called SAC¹: these are the problems that can be recognized by “uniform” circuits with semi-unbounded fan-in of depth $O(\log n)$ and polynomial size. This class has also been shown to be equivalent to classes of problems that are defined using Alternating Turing Machines and non-deterministic Turing machines with access to an auxiliary stack [24, 26]. We define parameterized analogues of the classes defined using Alternating Turing Machines or non-deterministic Turing machines with access to an auxiliary stack, and show these to be equivalent. This is how we define our class XALP.

Allender et al. [3] considers SATISFIABILITY where the treewidth of the associated graph is $O(\log^k n)$ for all $k \geq 1$. We restrict ourselves to the case $k = 1$ since this is where we could find interesting complete problems, but we expect that a similar generalisation is possible in our setting.

The main contribution of our paper is to transfer definitions and results from the classical world to the parameterized setting, by which we provide a natural framework to establish the complexity of many well-known parameterized problems. We provide a number of natural XALP-complete problems, but we expect that in the future it will be shown that XALP is the “right box” for many more problems of interest.

Paper overview. In Section 2, we give a number of definitions, discuss the classical analogues of XALP, and formulate a number of key parameterized problems. Several equivalent characterizations of the class XALP are given in Section 3. In Section 4, we introduce a “tree variant” of the wellknown MULTICOLOR CLIQUE problem. We call this problem TREE-CHAINED MULTICOLOR CLIQUE, and show it to be XALP-hard with a direct proof from an acceptance problem of a suitable type of Turing Machine, inspired by Cook’s proof of the NP-completeness of SATISFIABILITY [11]. In Section 5, we build on this and give a number of other examples of XALP-complete problems, including tree variants of WEIGHTED SATISFIABILITY and several problems parameterized by treewidth or another tree-structured graph parameter.

2 Definitions

We assume that the reader is familiar with a number of well-known notions from graph theory and parameterized complexity, e.g., FPT, the W-hierarchy, clique, independent set, etc. (See e.g., [12].)

A *tree decomposition* of a graph $G = (V, E)$ is a pair $(T = (I, F), \{X_i \mid i \in T\})$ with $T = (I, F)$ a tree and $\{X_i \mid i \in I\}$ a family of (not necessarily disjoint) subsets of V (called *bags*) such that $\bigcup_{i \in I} X_i = V$, for all edges $vw \in E$, there is an i with $v, w \in X_i$, and for all v , the nodes $\{i \in I \mid v \in X_i\}$ form a connected subtree of T . The *width* of a tree decomposition $(T, \{X_i \mid i \in T\})$ is $\max_{i \in I} |X_i| - 1$, and the *treewidth* of a graph G is the maximum width over all tree decompositions of G . A *path decomposition* is a tree decomposition $(T = (I, F), \{X_i \mid i \in T\})$ with T a path, and the *pathwidth* is the minimum width over all path decompositions of G .

2.1 Turing Machines and Classes

We assume the reader to be familiar with the basic concept of a Turing Machine. Here, we consider TMs that have access to both a fixed input tape (where the machine can only read), and a work tape of specified size (where the machine can both read and write). We consider *Non-deterministic Turing Machines* (NTM), where the machine can choose between different transitions, and accepts, if at least one choice of transitions leads to an accepting state, and *Alternating Turing Machines* (ATM), where the machine can both make non-deterministic steps (accepting when at least one choice leads to acceptance), and *co-non-deterministic steps* (accepting when both choices lead to acceptance). We assume a co-non-deterministic step always makes a binary choice, i.e., there are exactly two transitions that can be done.

Acceptance of an ATM A can be modelled by a rooted binary tree T , sometimes called a *run* or a *computation tree* of the machine. Each node of T is labelled with a configuration of A : the 4-tuple consisting of the machine state, work tape contents, location of work tape pointer, and location of input tape pointer. Each edge of T is labelled with a transition. The starting configuration is represented by the root of T . A node with one child makes a non-deterministic step, and the arc is labelled with a transition that leads to acceptance; a node with two children makes a co-non-deterministic step, with the children the configurations after the co-non-deterministic choice. Each leaf is a configuration with an accepting state. The *time* of the computation is the depth of the tree; the *treewidth* is the total number of nodes in this computation tree. For more information, see e.g., [24, 23]. A *computation path* is a path from root to leaf in the tree.

We also consider NTMs which additionally have access to an auxiliary stack. For those, a transition can also move the top element of the stack to the current location of the work tape (“pop”), or put a symbol at the top of the stack (“push”). We stress that only the top element can be accessed or modified, the machine cannot freely read other elements on the stack.

We use the notation $N[t(n, k), s(n, k)]$ to denote languages recognizable by a NTM running in time $t(n, k)$ with $s(n, k)$ working space and $A[t(n, k), s(n, k)]$ to denote languages recognizable by an ATM running in treewidth $t(n, k)$ with $s(n, k)$ working space. We note that we are free to put the constraint that *all* runs have treewidth at most $t(n, k)$, since we can add a counter that keeps track of the number of remaining steps, and reject when this runs out (similar to what is done in the proof of Theorem 1). We write $NAuxPDA[t(n, k), s(n, k)]$ to denote languages recognizable by a NTM with a stack (AUXiliary Push-Down Automaton) running in time $t(n, k)$ with $s(n, k)$ working space.

Ruzzo [24] showed that for any function $s(n)$, $NAuxPDA[n^{O(1)} \text{ time}, s(n) \text{ space}] = A[n^{O(1)} \text{ treewidth}, s(n) \text{ space}]$. Allender et al. [3] provided natural complete problems when $s(n) = \log^k(n)$ for all $k \geq 1$ (via a circuit model called SAC, which we will not use in our paper). Our interest lies in the case $k = 1$, where it turns out the parameterized analogue is the natural home of “tree-like” problems.

Another related work by Pilipczuk and Wrochna [23] shows that there is a tight relationship between the complexity of 3-COLORING on graphs of treedepth, pathwidth, or treewidth $s(n)$ and problems that can be solved by TMs with adequate resources depending on $s(n)$.

2.2 From classical to parameterized

In this paper we introduce the class $XALP = NAuxPDA[f \text{poly}, f \log]$. Following [9], we use the name $XNLP$ for the class $N[f \text{poly}, f \log]$; $f \text{poly}$ is shorthand notation for $f(k)n^{O(1)}$ for

some computable function f , and $f \log$ shorthand notation for $f(k) \log n$.

The crucial difference between the existing classical results and our results is that we consider parameterized complexity classes. These classes are closed under parameterized reductions, i.e. reductions where the parameter of the reduced instance must be bounded by the parameter of the initial instance. In our context, we have an additional technicality due to the relationship between time and space constraints. While a logspace reduction is also a polynomial time reduction, a reduction using $f(k) \log n$ space (XL) could use up to $n^{f(k)}$ time (XP). XNLP and XALP are closed under *pl-reductions* where the space bound is $f(k) + O(\log n)$ (which implies FPT time), and under *ptl-reductions* running in $f(k)n^{O(1)}$ time and $f(k) \log n$ space.

We now give formal definitions.

A *parameterized reduction* from a parameterized problem $Q_1 \subseteq \Sigma_1^* \times \mathbb{N}$ to a parameterized problem $Q_2 \subseteq \Sigma_2^* \times \mathbb{N}$ is a function $f: \Sigma_1^* \times \mathbb{N} \rightarrow \Sigma_2^* \times \mathbb{N}$ such that the following holds.

1. For all $(x, k) \in \Sigma_1^* \times \mathbb{N}$, $(x, k) \in Q_1$ if and only if $f((x, k)) \in Q_2$.
2. There is a computable function g such that for all $(x, k) \in \Sigma_1^* \times \mathbb{N}$, if $f((x, k)) = (y, k')$, then $k' \leq g(k)$.

If there is an algorithm that computes $f((x, k))$ in space $O(g(k) + \log n)$, with g a computable function and $n = |x|$ the number of bits to denote x , then the reduction is a *parameterized logspace reduction* or *pl-reduction*.

If there is an algorithm that computes $f((x, k))$ in time $g(k)n^{O(1)}$ and space $O(h(k) \log n)$, with g, h computable functions and $n = |x|$ the number of bits to denote x , then the reduction is a *parameterized tractable logspace reduction* or *ptl-reduction*.

3 Equivalent characterisations of XALP

In this section, we give a number of equivalent characterisations of XALP.

► **Theorem 1.** *The following parameterized complexity classes are all equal.*

1. $NAuxPDA[f \text{ poly}, f \log]$, the class of parameterized decision problems for which instances of size n with parameter k can be solved by a non-deterministic Turing machine with $f(k) \log n$ memory in $f(k)n^{O(1)}$ time when given a stack, for some computable function f .
2. The class of parameterized decision problems for which instances of size n with parameter k can be solved by an alternating Turing machine with $f(k) \log n$ memory whose computation tree is a binary tree on $f(k)n^{O(1)}$ nodes, for some computable function f .
3. The class of parameterized decision problems for which instances of size n with parameter k can be solved by an alternating Turing machine with $f(k) \log n$ memory whose computation tree is obtained from a binary tree of depth $O(\log n) + f(k)$ by subdividing each edge $f(k)n^{O(1)}$ times, for some computable function f .
4. The class of parameterized decision problems for which instances of size n with parameter k can be solved by an alternating Turing machine with $f(k) \log n$ memory, for which the computation tree has size $f(k)n^{O(1)}$ and uses $O(\log n) + f(k)$ co-non-deterministic steps per computation path, for some computable function f .

Proof. The proof is similar to the equivalence proofs for the classical analogues, and added for convenience of the reader. We prove the theorem by proving the series of inclusions $1 \subseteq 2 \subseteq 3 \subseteq 4 \subseteq 1$.

6:6 On the Complexity of Problems on Tree-Structured Graphs

$1 \subseteq 2$. Consider a problem that can be solved by a non-deterministic Turing Machine T with a stack and $f(k) \log n$ memory in $f(k)n^{O(1)}$ time. We will simulate T using an alternating Turing machine T' .

We place three further assumptions on T , which can be implemented by changing the function f slightly if needed.

- The Turing machine T has two counters. One keeps track of the height of the stack, and the other keeps track of the number of computation steps. A single computation step may involve several operations; we just need that the running time is polynomially bounded in the number of steps.
- We assume that T only halts with acceptance when the stack is empty. (Otherwise, do not yet accept, but pop the stack using the counter that tells the height of the stack, until the stack is empty.)
- Each pop operation performed by T is a deterministic step. This can be done by adding an extra state to T and splitting a non-deterministic step into a non-deterministic step and a deterministic step if needed.

We define a *configuration* as a tuple which includes the state of T , the value of the two pointers and the content of the memory. In particular, this does not contain the contents of the stack and so a configuration can be stored using $O(f(k) \log n)$ bits. (Note that the value of both pointers is bounded by $f(k)n^{O(1)}$.)

We will build a subroutine $A(c_1, c_2)$ which works as follows.

- The input c_1, c_2 consists of two configurations with the same stack height.
- The output is whether T has an accepting run from c_1 to c_2 without popping the top element from the stack in c_1 ; the run may pop elements that have yet to get pushed.

We write $\text{Apply}(c, \text{POP}(s))$ for the configuration that is obtained when we perform a pop operation in configuration c and obtain s from the stack. This is only defined if T can do a pop operation in configuration c (e.g. it needs to contain something on the stack). We define the configuration $\text{Apply}(c, \text{PUSH}(s))$ in a similar manner, where this time s gets pushed onto the stack.

We let T' simulate T starting from configuration c_s as follows. Our alternating Turing machine T' will start with the following non-deterministic step: guess the c_a configuration that accepts at the end of the run. It then performs the subroutine $A(c_s, c_a)$.

We implement $A(c_s, c_a)$ as follows. A deterministic or non-deterministic step of T is carried out as usual.

If T is in some configuration c and wants to push s to the stack, then let $c' = \text{Apply}(c, \text{PUSH}(s))$ and let T' perform a non-deterministic step that guesses a configuration c'_2 with the same stack height as c' for which the next step is to pop (and the number of remaining computation steps is plausible). Let $c_2 = \text{Apply}(c'_2, \text{POP}(s))$. We make T' do a co-non-deterministic step consisting of two branches:

- T' performs the subroutine $A(c', c'_2)$.
- T' performs the subroutine $A(c'_2, c_a)$.

We ensure that in configuration c'_2 , the number of steps taken is larger than in configuration c' . This ensures that T' will terminate.

Since a configuration can be stored using $O(f(k) \log n)$ and T' always stores at most a bounded number of configurations, T' requires only $O(f(k) \log n)$ bits of memory. The computation tree for T' is binary. The total number of nodes of the computation tree of T' is $f(k)n^{O(1)}$ since each computation step of T appears at most once in the tree (informally: our co-non-deterministic steps split up the computation path of T into two disjoint parts), and we have added at most a constant number of steps per step of T . To see this, the computation

tree of T' may split a computation path $c \rightarrow_{\text{push}} c' \rightarrow \cdots \rightarrow c'_2 \rightarrow_{\text{pop}} c_2 \rightarrow \cdots \rightarrow c_a$ of T into two parts: one branch will simulate $c' \rightarrow \cdots \rightarrow c'_2$ and the other branch will simulate $c_2 \rightarrow \cdots \rightarrow c_a$. At most a constant number of additional nodes (e.g. the node which takes the co-non-deterministic step) are added to facilitate this. Importantly, the configurations implicitly stored a number of remaining computation steps, and so T' can calculate from c', c'_2 how many steps T is supposed to take to move between c' and c'_2 .

2 \subseteq 3. The intuition behind this proof is to use that any n -vertex tree has a tree decomposition of bounded treewidth of depth $O(\log n)$.

Let A be an alternating Turing machine for some parameterized problem with a computation tree of size $f(k)n^{O(1)}$ and $f(k)\log n$ bits of memory.

We build an alternating Turing machine B that simulates A for which the computation tree is a binary tree which uses $O(f(k) + \log n)$ co-non-deterministic steps per computation branch and $O(f(k)\log n)$ memory. We can after that ensure that there are $f(k)n^{O(1)}$ steps between any two co-non-deterministic steps by adding “idle” steps if needed.

We ensure that B always has *advice* in memory: 1 configuration for which A accepts. In particular, if c' is the configuration stored as advice when A is in configuration c with a bound of n steps, then B checks if A can get from c to c' within n steps.

We also maintain a counter for the number of remaining steps: the number of nodes that are left in the computation tree of A , when rooted at the current configuration c not counting the node of c itself. In particular, the counter is 0 if c is supposed to be a leaf.

We let B simulate A as follows. Firstly, if no advice is in memory, it makes a non-deterministic step to guess a configuration as advice.

Suppose that A is in configuration c with n_0 steps left. We check the following in order. If c equals the advice, then we accept. If $n_0 \leq 0$, then we reject. If the next step of A is non-deterministic or deterministic step, then we perform the same step. The interesting things happen when A is about to perform a co-non-deterministic step starting from c with n_0 steps left. If $n_0 \leq 1$, then we reject: there is no space for such a step. Otherwise, we guess $n_1, n_2 \geq 0$ such that $n_1 + n_2 = n_0 - 2$, and children c_1, c_2 of c in the computation tree of A . Renumbering if needed, we may assume that the advice c' is supposed to appear in the subtree of c_1 . We also guess an advice c'_2 for c_2 . We create a co-non-deterministic step with two branches, one for the computation starting from c_1 with n_1 steps and the other from c_2 with n_2 . We describe how we continue the computation starting from c_1 ; the case in which c_2 is analogous.

Recall that some configuration c' has been stored as advice. We want to ensure that the advice is limited to one configuration. First, we non-deterministically guess a configuration c'' . We non-deterministically guess whether c'' is an ancestor of c' . We perform different computation depending on the outcome.

- Suppose that we guessed that c'' is an ancestor of c' . We guess integers $\frac{1}{3}n_1 \leq a, b \leq \frac{2}{3}n_1$ with $a + b = n_1$. We do a co-non-deterministic step: one branch starts in c_1 with c' as advice and a steps, the other branch starts in c' with c'' as advice and b steps.
- Suppose that c'' is not an ancestor of c' . We guess a configuration ℓ , corresponding to the least common ancestor of c' and c'' in the computation tree. We guess integers $0 \leq a, b, a', b' \leq \frac{2}{3}n_1$ with $a + b + a' + b' = n_1$. We perform a co-non-deterministic branch to obtain four subbranches: starting in c with ℓ as advice and a steps, ℓ with c' as advice and b steps, starting in ℓ with c'' as advice and a' steps and starting in c'' with no advice and b' steps.

6:8 On the Complexity of Problems on Tree-Structured Graphs

In order to turn our computation tree into a binary tree, we may choose to split the single co-non-deterministic step into two steps.

Since at any point, we store at most a constant number of configurations, this can be performed using $O(f(k) \log n)$ bits in memory.

It remains to show that B performs $O(\log n + f(k))$ co-non-deterministic steps per computation path. The computation of B starts with a counter for the number of steps which is at most $f(k)n^{O(1)}$; every time B performs a co-non-deterministic step, this counter is multiplied by a factor of at most $\frac{2}{3}$. The claim now follows from the fact that $\log(f(k)n^{O(1)}) = O(\log n + \log f(k))$.

3 \subseteq 4. Let T be an alternating Turing machine using $f(k) \log n$ memory whose computation fits in a tree obtained from a binary tree of depth d by subdividing each edge $f(k)n^{O(1)}$ times. Then T uses $f(k)n^{O(1)}$ time (with possibly a different constant in the $O(1)$ -term) and performs at most d co-non-deterministic steps per computation path. Hence this inclusion is immediate.

4 \subseteq 1. We may simulate the alternating Turing machine using a non-deterministic Turing machine stack as follows. Each time we wish to do a co-non-deterministic branch, we put the current configuration c onto our stack and continue to the left-child of c . Once we have reached an accepting state, we pop an element c of the stack and next continue to the right child of c . The total computation time is bounded by the number of nodes in the computation tree and the memory requirement does not increase by more than a constant factor. (Note that in particular, our stack will never contain more than $\log n + f(k)$ elements.) \blacktriangleleft

Already in the classical setting, it is expected that $\text{NL} \subsetneq \text{A}[\text{poly treesize}, \log \text{space}]$. We stress the fact that this would imply $\text{XNLP} \subsetneq \text{XALP}$, since we can always ignore the parameter. It was indeed noted in [3, Corollary 3.13] that the assumption $\text{NL} \subsetneq \text{A}[\text{poly treesize}, \log \text{space}]$ separates the complexity of SAT instances of logarithmic pathwidth from SAT instances of logarithmic treewidth. Allender et al. [3] formulates this result in terms of SAC^1 instead of the equivalent $\text{A}[\text{poly treesize}, \log \text{space}]$. We expect that a parameterized analogue of SAC^1 can be added to the equivalent characterization above, but decided to not pursue this here. The definition of such a circuit class requires a notion of “uniformity” that ensures that the circuits have a “small description”, which makes it more technical.

4 XALP-completeness for a tree-chained variant of Multicolor Clique

Our first XALP-complete problem is a “tree” variant of the well-known MULTICOLOR CLIQUE problem.

TREE-CHAINED MULTICOLOR CLIQUE

Input: A binary tree $T = (I, F)$, an integer k , and for each $i \in I$, a collection of k pairwise disjoint sets of vertices $V_{i,1}, \dots, V_{i,k}$, and a graph G with vertex set $V = \bigcup_{i \in I, j \in [1,k]} V_{i,j}$.

Parameter: k .

Question: Is there a set of vertices $W \subseteq V$ such that W contains exactly one vertex from each $V_{i,j}$ ($i \in I, j \in [1, k]$), and for each pair $V_{i,j}, V_{i',j'}$ with $i = i'$ or $ii' \in F$, $j, j' \in [1, k]$, $(i, j) \neq (i', j')$, the vertex in $W \cap V_{i,j}$ is adjacent to the vertex in $W \cap V_{i',j'}$?

This problem is the XALP analogue of the XNLP-complete problem CHAINED MULTICOLOR CLIQUE, in which the input tree T is a path instead. This change of “path-like” computations to “tree-like” computations is typical when going from XNLP to XALP.

For the TREE-CHAINED MULTICOLOR INDEPENDENT SET problem, we have a similar input and question except that we ask for the vertex in $W \cap V_{i,j}$ and the vertex in $W \cap V_{i',j'}$ *not* to be adjacent. In both cases, we may assume that edges of the graphs are only between vertices of $V_{i,j}$ and $V_{i',j'}$ with $i = i'$ or $ii' \in F$, $j, j' \in [1, k]$, $(i, j) \neq (i', j')$. We call *tree-chained multicolor clique* (resp. *independent set*) a set of vertices satisfying the respective previous conditions.

The problems above can be seen as binary CSPs by replacing vertex choice by assignment choice.

Membership of these problems in XNLP seems unlikely, since it is difficult to handle the “branching” of the tree. However, in XALP this is easy to do using the co-non-deterministic steps and indeed the membership follows quickly.

► **Lemma 2.** TREE-CHAINED MULTICOLOR CLIQUE *is in XALP.*

Proof. We simply traverse the tree T with an alternating Turing machine that uses a co-non-deterministic step when it has to check two subtrees. When at $i \in I$, the machine first guesses a vertex for each $V_{i,j}$, $j \in [k]$. It then checks that these vertices form a multicolor clique with the vertices chosen for the parent of i . The vertices chosen for the parent can now be forgotten and the machine moves to checking children of i . The machine works in polynomial treesize, and uses only $O(k \log n)$ space to keep the indices of chosen vertices for up to two nodes of T , the current position on T . ◀

We next show that TREE-CHAINED MULTICOLOR CLIQUE is XALP-hard. We will use the characterization of XALP where the computation tree of the alternating Turing machine is a specific tree (3), which allows us to control when co-non-deterministic steps can take place.

Let \mathcal{M} be an alternating Turing machine with computation tree $T = (I, F)$, let x be its input of size n , and k be the parameter. The plan is to encode the configuration of \mathcal{M} at the step corresponding to node $i \in V(T)$ by the choice of the vertices in $V_{i,1}, \dots, V_{i,k'}$ (for some $k' = f(k)$). The possible transitions of the Turing Machine are then encoded by edges between V_i and $V_{i'}$ for $ii' \in F$, where $V_j = \bigcup_{\ell \in [1, k']} V_{j,\ell}$.

A configuration of \mathcal{M} contains the same elements as in the proof of Theorem 1:

- the current state of \mathcal{M} ,
- the position of the head on the input tape,
- the working space which is $f(k) \log n$ bits long, and
- the position of the head on the work tape.

We partition the working space in $k' = f(k)$ pieces of $\log n$ consecutive bits, and have a set of vertices $V_{i,j}$ for each. Formally, we have a vertex $v_{q,p,b,w}$ in $V_{i,j}$ for each tuple (q, p, b, w) where q is the state of the machine, p is the position of the head on the input tape, $b \in \{\mathbf{after}, \mathbf{before}\} \uplus [\log n]$ indicates if the block of the work tape is before or after the head, or its position in the block, and w is the current content of the j th block of the work tape.

The edges between vertices of V_i enforce that possible choices of vertices correspond to valid configurations. There is an edge between $v \in V_{i,j}$ and $w \in V_{i,j+1}$ with corresponding tuples (q, p, b, w) and (q', p', b', w') , if and only if $q = q'$, $p = p'$, and either $b' = b \in \{\mathbf{after}, \mathbf{before}\}$, or $b \in [\log n]$ and $b' = \mathbf{after}$, or $b = \mathbf{before}$ and $b' \in [\log n]$.

► **Observation 3.** *If $v_1, \dots, v_{k'}$ is path with $v_j \in V_{i,j}$, then at most one of the v_j can encode a block with the work tape head, blocks before the head have $b = \mathbf{before}$, blocks after the head have $b = \mathbf{after}$, and all blocks encode the same state and position of the input tape head.*

6:10 On the Complexity of Problems on Tree-Structured Graphs

The edges between vertices of V_i and $V_{i'}$ for $ii' \in F$ enforce that the configurations chosen in V_i and $V_{i'}$ encode configurations with a transition from one to the other. There is an edge between $v \in V_{i,j}$ and $w \in V_{i',j}$ with corresponding tuples (q, p, b, w) and (q', p', b', w') , such that $(b, b') \in \{(\mathbf{after}, \mathbf{after}), (\mathbf{before}, \mathbf{before}), (\mathbf{after}, 1), (\mathbf{before}, \log n)\}$ if and only if $w = w'$. There is an edge between $v \in V_{i,j}$ and $w \in V_{i',j}$ with corresponding tuples (q, p, b, w) and (q', p', b', w') , such that $b \in [\log n]$, if and only if, there is a transition of \mathcal{M} from state q to state q' that would write $w'[b]$ when reading $x[p]$ on the input tape and $w[b]$ on the work tape, move the input tape head by $p' - p$ and the work tape by $b' - b$ (where $\mathbf{after} = 0$ and $\mathbf{before} = 1 + \log n$), and for $\ell \in [\log n] \setminus \{b\}$ $w[\ell] = w'[\ell]$.

▷ **Claim 4.** If $v_1, \dots, v_{k'}, v'_1, \dots, v'_{k'}$ induce a $2 \times k'$ “multicolor grid” (i.e. $v_1, \dots, v_{k'}$ is a path with $v_j \in V_{i,j}$, $v'_1, \dots, v'_{k'}$ is a path with $v'_j \in V_{i',j}$, there are edges $v_j v'_j$ for $j \in [k']$, $ii' \in F$, and $v'_1, \dots, v'_{k'}$ encodes a valid configuration), then $v_1, \dots, v_{k'}$ encodes a valid configuration that can reach the configuration encoded by $v'_1, \dots, v'_{k'}$ using one transition of \mathcal{M} .

Proof. This follows easily from the construction but we still detail why this is sufficient when the work tape head moves to a different block.

We consider the case when the head moves to the block before it. That is we consider the case where v'_j encodes $b' = \log n$ and v_j encodes $b = \mathbf{before}$. First, note that there is an edge from $v_j v'_j$ allowing this. We use Observation 3 and conclude that v'_{j+1} (if it exists) must encode head position \mathbf{after} for its block. The edge $v_{j+1} v'_{j+1}$ then enforces that v_{j+1} encodes head position 1 but it can also exist only if there is a transition of \mathcal{M} that moves the work tape head to the previous block and the written character at the beginning of the block encoded by v'_{j+1} corresponds to such transition. Moving to the next block is a symmetric case. ◁

We have further constraints on the vertices placed in each $V_{i,j}$ based on what i is in T .

- If i is in a leaf of T , then we only have vertices with a corresponding tuple (q, p, b, w) with q an accepting state.
- If i is in a “branching” vertex of T (i.e. i has two children), then we only have vertices with a corresponding tuple (q, p, b, w) with q a universal state.
- If i is the root, then only vertices corresponding to the initial configuration are allowed.
- Otherwise, we only have vertices with tuples encoding an existential state.

Furthermore, we have to make sure that when branching we take care of the two distinct transitions. We actually assume that T has an order on children for vertices with two children. Then for the edge of T to the first (resp. second) child, we only allow the first (resp. second) transition from the configuration of the parent (which must have a universal state).

We now complete the graph with edges that do not enforce constraints so that we may find a multicolor clique instead of only a $2 \times k'$ multicolor grid. For every $i \in I$, and $j, j' \in [k']$ such that $|j - j'| > 1$, we add all edges between $V_{i,j}$ and $V_{i,j'}$. For every $ii' \in F$, and $j, j' \in [k']$ such that $j \neq j'$, we add all edges between $V_{i,j}$ and $V_{i',j'}$. It should be clear that to find a multicolor clique for some edge ii' after adding these edges is equivalent to finding a “multicolor grid” before they were added¹.

▷ **Claim 5.** The constructed graph admits a tree-chained multicolor clique, if and only if, there is an accepting run for \mathcal{M} with input x and computation tree T .

¹ Asking for these multicolor grids for each edge of the tree instead of multicolor cliques also leads to an XALP-complete problem but we do not use this problem for further reductions. It could however be used as a starting point for new reductions.

Proof. The statement follows from a straight-forward induction on T showing that for each configuration C of \mathcal{M} that can be encoded by the construction at $i \in I$, its encoding can be extended to a tree-chained multicolor clique of the subtree of T rooted at i , if and only if there is an accepting run of \mathcal{M} from C with as computation tree the subtree of T rooted at i . \triangleleft

Each $V_{i,j}$ has $O(|Q|n^2 \log n)$ vertices (for Q the set of states). Edges are only between $V_{i,j}$ and $V_{i',j'}$ such that $ii' \in F$ or $i = i'$. We conclude that there are $g(k)n^{O(1)}$ vertices and edges in the constructed graph per vertex of T , which is itself of size $h(k)n^{O(1)}$ so the constructed instance has size $g(k)h(k)n^{O(1)}$, for g, h computable functions. The construction can even be performed using only $g'(k) + O(\log(n))$ space for some computable function g' . Note also that $k' = f(k)$: the new parameter is bounded by a function of the initial parameter. This shows that our reduction is a parameterized pl-reduction, and we conclude XALP-hardness. Combined with Lemma 2, we proved the following result.

► **Theorem 6.** TREE-CHAINED MULTICOLOR CLIQUE *is XALP-complete.*

One may easily modify this to the case where each color class has the same size, by adding isolated vertices.

By taking the complement of the graph, we directly obtain the following result.

► **Corollary 7.** TREE-CHAINED MULTICOLOR INDEPENDENT SET *is XALP-complete.*

5 More XALP-complete problems

In this section, we prove a collection of problems on graphs, given with a tree-structure, to be complete for the class XALP. The proofs are of different types: in some cases, the proofs are new, in some cases, reformulations of existing proofs from the literature, and in some cases, it suffices to observe that an existing transformation from the literature keeps the width-parameter at hand bounded.

5.1 List coloring

The problems LIST COLORING and PRE-COLORING EXTENSION with pathwidth as parameter are XNLP-complete [9]. A simple proof shows XALP-completeness with treewidth as parameter. Jansen and Scheffler [21] showed that these problem are in XP, and Fellows et al. [18] showed $W[1]$ -hardness.

► **Theorem 8.** LIST COLORING *and* PRE-COLORING EXTENSION *are XALP-complete with treewidth as parameter.*

Proof. Membership follows as usual. The color of (uncolored) vertices is non-deterministically chosen when they are introduced. We maintain the color of vertices of the current bag in the working space. We use co-non-deterministic steps when the tree decomposition branches. We check that introduced edges do not contradict the coloring being proper. This uses $O(k \log n)$ space, and runs in polynomial total time.

We first show XALP-hardness of LIST COLORING. We reduce from TREE-CHAINED MULTICOLOR INDEPENDENT SET. Suppose we have an instance of this problem. The set of colors equals the set of vertices V . For each class $V_{i,j}$, $i \in I$, $j \in [1, k]$, we take a vertex v_{ij} with set of colors V_{ij} .

For each pair of “incident classes” $V_{i,j}$, $V_{i',j'}$ with $i = i'$ or ii' an edge in F , $ij \neq i'j'$, and each edge $vw \in E \cap V_{ij} \times V_{i'j'}$, we add a new vertex with set of colors $\{v, w\}$, which is incident to v_{ij} and $v_{i'j'}$. Let H be the resulting graph.

Now, H has a list coloring, if and only if there is a tree-chained multicolor independent set in G . The transformation of solutions is straightforward: the chosen colors for v_{ij} are equal to the chosen vertices from V_{ij} . If we choose two adjacent vertices in incident classes, then we do not have a color available for a new vertex; if we have a tree-chained independent set, then each new vertex has at least one available color.

H has treewidth at most $2k - 1$: take a root of T , for each $i \in I$, let X_i consist of all v_{ij} and $v_{i'j}$, i' the parent of i , $j \in [1, k]$. Now, for each new vertex, we add a bag containing this vertex and its two neighbors, making it incident to a bag that contains its neighbors.

The standard reduction from PRE-COLORING EXTENSION to LIST COLORING that adds for each forbidden color c of a vertex v a new neighbor to v precolored with c does not increase the treewidth, which shows XALP-hardness for PRE-COLORING EXTENSION with treewidth as parameter. ◀

5.2 Tree variants of Weighted Satisfiability

From TREE-CHAINED MULTICOLOR INDEPENDENT SET, we can show XALP-completeness of tree variants of what in [9] was called CHAINED WEIGHTED CNF-SATISFIABILITY and its variants (which in turn are analogues of WEIGHTED CNF-SATISFIABILITY, see e.g. [15, 16]).

TREE-CHAINED WEIGHTED CNF-SATISFIABILITY

Input: A tree $T = (I, F)$, sets of variables $(X_i)_{i \in I}$, and clauses C_1, \dots, C_m , each with either only variables of X_i for some $i \in I$, or only variables of X_i and X_j for some $ij \in F$.

Parameter: k .

Question: Is there an assignment of at most k variables in each X_i that satisfies all clauses?

POSITIVE PARTITIONED TREE-CHAINED WEIGHTED CNF-SATISFIABILITY

Input: A tree $T = (I, F)$, sets of variables $(X_i)_{i \in I}$, and clauses of positive literals C_1, \dots, C_m , each with either only variables of X_i for some $i \in I$, or only variables of X_i and X_j for some $ij \in F$. Each X_i is partitioned into $X_{i,1}, \dots, X_{i,k}$.

Parameter: k .

Question: Is there an assignment of exactly one variable in each $X_{i,j}$ that satisfies all clauses?

NEGATIVE PARTITIONED TREE-CHAINED WEIGHTED CNF-SATISFIABILITY

Input: A tree $T = (I, F)$, sets of variables $(X_i)_{i \in I}$, and clauses of negative literals C_1, \dots, C_m , each with either only variables of X_i for some $i \in I$, or only variables of X_i and X_j for some $ij \in F$. Each X_i is partitioned into $X_{i,1}, \dots, X_{i,k}$.

Parameter: k .

Question: Is there an assignment of exactly one variable in each $X_{i,j}$ that satisfies all clauses?

► **Theorem 9.** POSITIVE PARTITIONED TREE-CHAINED WEIGHTED CNF-SATISFIABILITY, NEGATIVE PARTITIONED TREE-CHAINED WEIGHTED CNF-SATISFIABILITY, and TREE-CHAINED WEIGHTED CNF-SATISFIABILITY are XALP-complete.

Proof. We first show membership for TREE-CHAINED WEIGHTED CNF-SATISFIABILITY, which implies membership for the more structured versions. We simply follow the tree shape of our instance by branching co-non-deterministically when the tree branches. We keep the

indices of the $2k$ variables chosen non-deterministically for the “local” clauses in the working space. We then check that said clauses are satisfied.

We first show hardness for NEGATIVE PARTITIONED TREE-CHAINED WEIGHTED CNF-SATISFIABILITY by reducing from TREE-CHAINED MULTICOLOR INDEPENDENT SET. For each vertex v , we have a Boolean variable x_v . We denote by $X_{i,j}$ the set of variables $\{x_v : v \in V_{i,j}\}$, and by X_i the set of variables $\{x_v : v \in V_i\}$. This preserves the partition properties. For each edge uv , we add the clause $\neg x_u \vee \neg x_v$.

► **Observation 10.** U is multicolor independent set if and only if $\{x_u : u \in U\}$ is a satisfying assignment.

To reduce to POSITIVE PARTITIONED TREE-CHAINED WEIGHTED CNF-SATISFIABILITY, we simply replace negative literals $\neg x_v$ for $x_v \in X_{i,j}$ by a disjunction of positive literals $\bigvee_{y \in X_{i,j} \setminus \{x_v\}} y$. This works because, due to the partition constraint, a variable $x \in X_{i,j}$ is assigned \perp if and only if another variable $y \in X_{i,j} \setminus \{x\}$ is assigned \top .

To reduce to TREE-CHAINED WEIGHTED CNF-SATISFIABILITY, we simply express the partition constraints using clauses. For each $X_{i,j}$, we add the clauses $\bigvee_{y \in X_{i,j}} y$, and for each pair $\{x, y\} \subseteq X_{i,j}$ the clause $\neg x \vee \neg y$. This enforces that we pick at least one variable, and at most one variable, for each $X_{i,j}$. ◀

5.3 Logarithmic Treewidth

Although XALP-complete problems are in XP and not in FPT, there is a link between XALP and single exponential FPT algorithms on tree decompositions. Indeed, by considering instances with treewidth $k \log n$, where k is the parameter, the single exponential FPT algorithm becomes an XP algorithm. We call this parameter logarithmic treewidth.

INDEPENDENT SET parameterized by logarithmic treewidth

Input: A graph $G = (V, E)$, with a given tree decomposition of width at most $k \log |V|$, and an integer W .

Parameter: k .

Question: Is there an independent set of G of size at least W ?

► **Theorem 11.** INDEPENDENT SET with logarithmic treewidth as parameter is XALP-complete.

Proof. We start with membership which follows from the usual dynamic programming on the tree decomposition. We maintain for each vertex v in the current bag whether v is in the independent set or not. When introducing a vertex v , we non-deterministically decide if v is put in the independent set or not. We reject if an edge is introduced between two vertices of the independent set. We make a co-non-deterministic step whenever the tree decomposition is branching. Since we only need one bit of information per vertex in the bag, this requires only $O(k \log n)$ working space, as for the running time we simply do a traversal of the tree decomposition which is only polynomial treesize.

We show hardness by reducing from POSITIVE PARTITIONED TREE-CHAINED WEIGHTED CNF-SATISFIABILITY. We can simply reuse the construction from [9] and note that the constructed graph has bounded logarithmic treewidth instead of logarithmic pathwidth because we reduced from the tree-chained SAT variant instead of the chained SAT variant. We describe the gadgets for completeness. First, the SAT instance is slightly adjusted for technical reasons. For each $X_{i,j}$, we add a clause containing exactly its initial variables. This makes sure that the encoding of the chosen variable is valid. We assume the variables in each $X_{i,j}$ to be indexed starting from 0.

Variable gadget. For each $X_{i,j}$, let $t_{i,j} = \lceil \log_2 |X_{i,j}| \rceil$. We add edges $\widehat{0}_\alpha \widehat{1}_\alpha$, $\alpha \in [1, t_{i,j}]$.

Clause gadget. For each clause with ℓ literals, we assume ℓ to be even by adding a dummy literal if necessary. We add paths $p_0, \dots, p_{\ell+1}$, and p'_1, \dots, p'_ℓ . For $i \in [1, \ell]$, we add the edge $p_i p'_i$. We then add vertex v_i for $i \in [1, \ell]$, which represents the i th literal of the clause. Let $b_1 \dots b_{t_{i',j'}}$ be the binary representation of the index of the corresponding variable of $X_{i',j'}$. Then v_i is adjacent to p_i, p'_i and the vertices $\widehat{1 - b_\alpha}$ for $\alpha \in [1, t_{i,j}]$. For the dummy literal, there is no vertex v_i .

The clause gadget has an independent set of size $\ell + 2$ if and only if it contains a vertex v_i . When the variable gadgets have one vertex in the independent set on each edge, a vertex v_i of a clause can be added to the independent set only if the independent set contains exactly the vertices of the variable gadget that give the binary representation of the variable corresponding to v_i .

Hence, the SAT instance is satisfiable if and only if there is an independent set of size $\sum_{i,j} t_{i,j} + \sum_i 2 + \ell_i$ in our construction. ◀

► **Corollary 12.** *The following problems are XALP-complete with logarithmic treewidth as parameter: VERTEX COVER, RED-BLUE DOMINATING SET, DOMINATING SET.*

Proof. The result for VERTEX COVER follows directly from Theorem 11 and the well known fact that a graph with n vertices has a vertex cover of size at most L , iff it has an independent set of size at least $n - L$. Viewing VERTEX COVER as a special case of RED-BLUE DOMINATING SET gives the following graph: subdivide all edges of G , and ask if a set of K original (blue) vertices dominates all new (red) subdivision vertices; as the subdivision step does not increase the treewidth, XALP-hardness of RED-BLUE DOMINATING SET with treewidth as parameter follows. To obtain XALP-hardness of DOMINATING SET, add to the instance G' of RED-BLUE DOMINATING SET, two new vertices x_0 and x_1 and edges from x_1 to x_0 and all blue vertices; the treewidth increases by at most one, and the minimum size of a dominating set in the new graph is exactly one larger than the minimum size of a red-blue dominating set in G' . Membership in XALP is shown similar as in the proof of Theorem 11. ◀

5.4 Other problems

Several XALP-hardness proofs follow from known reductions. Membership is usually easy to prove, by observing that the known XP-algorithms can be turned into XALP-membership by guessing table entries, and using the stack to store the information for a left child when processing a right subtree.

► **Corollary 13.** *The following problems are XALP-complete:*

1. CHOSEN MAXIMUM OUTDEGREE, CIRCULATING ORIENTATION, MINIMUM MAXIMUM OUTDEGREE, OUTDEGREE RESTRICTED ORIENTATION, and UNDIRECTED FLOW WITH LOWER BOUNDS, with the treewidth as parameter.
2. MAX CUT and MAXIMUM REGULAR INDUCED SUBGRAPH with cliquewidth as parameter.

Proof.

1. The reductions given in [4] and [25] can be used; one easily observes that these reductions keep the treewidth of the constructed instance bounded by a function of the treewidth of the original instance (often, a small additive constant is added.)

2. The reductions given in [7] can be reused with minimal changes, only the bound on linear clique-width becomes a bound on clique-width because of the “tree-shape” of the instance to reduce. ◀

CHOSEN MAXIMUM OUTDEGREE, CIRCULATING ORIENTATION, MINIMUM MAXIMUM OUTDEGREE, OUTDEGREE RESTRICTED ORIENTATION, and UNDIRECTED FLOW WITH LOWER BOUNDS, together with ALL-OR-NOTHING FLOW were shown to be XNLP-complete with pathwidth as parameter in [4]. Gima et al. [20] showed that MINIMUM MAXIMUM OUTDEGREE with vertex cover as parameter is $W[1]$ -hard. For related results, see also [25].

In [6], it is shown that TREE-PARTITION-WIDTH and DOMINO TREewidth are XALP-complete, which can be seen as an analog to BANDWIDTH being XNLP-complete.

6 Conclusions

We expect many (but not all) problems that are ($W[1]$ -)hard and in XP for treewidth as parameter to be XALP-complete; our paper gives good starting points for such proofs. Let us give an explicit example. The PEBBLE GAME PROBLEM [16, 22] parameterized by the number of pebbles is complete for XP, which is equal to $XAL=A[\infty, f \log]$. The problem corresponds to deciding whether there is a winning strategy in an adversarial two-player game with k pebbles on a graph where the possible moves depend on the positions of all pebbles. We can expect variants with at most $f(k) + O(\log n)$ moves to be complete for XALP.

Completeness proofs give a relatively precise complexity classification of problems. In particular, XALP-hardness proofs indicate that we do not expect a deterministic algorithm to use less than XP space if it runs in XP time. Indeed the inclusion of XNLP in XALP is believed to be strict, and already for XNLP-hard problems we have the following conjecture.

► **Conjecture 14** (Slice-wise Polynomial Space Conjecture [23]). *No XNLP-hard problem has an algorithm that runs in $n^{f(k)}$ time and $f(k)n^c$ space, with f a computable function, k the parameter, n the input size, and c a constant.*

While XNLP and XALP give a relatively simple framework to classify problems in terms of simultaneous bound on space and time, the parameter is allowed to blow up along the reduction chain. One may want to mimic the fine grained time complexity results based on the (Strong) Exponential Time Hypothesis. In this direction, one could assume that Savitch’s theorem is optimal as was done in [10].

Since XNLP is above the W -hierarchy, it could be interesting to study the relationship of XALP with some other hierarchies like the A -hierarchy and the AW -hierarchy. It is also unclear where to place LIST-COLORING parameterized by tree-partition-width². It was shown to be in XL and $W[1]$ -hard [5] but neither look like good candidates for completeness.

References

- 1 Karl A. Abrahamson, Rodney G. Downey, and Michael R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for $W[P]$ and PSPACE analogues. *Ann. Pure Appl. Log.*, 73:235–276, 1995. doi:10.1016/0168-0072(94)00034-Z.

² A tree-partition of a graph G is a decomposition of $V(G)$ into bags $(B_i)_{i \in V(T)}$, where T is a tree, such that $uv \in E(G)$ implies that the bags of u and v are the same or adjacent in T . The width is the size of the largest bag, and the tree-partition-width of G is found by taking the minimum width over all tree-partitions of G .

- 2 Michael Alekhovich and Alexander A. Razborov. Satisfiability, branch-width and tseitin tautologies. In *Proceedings of the 43rd Symposium on Foundations of Computer Science, FOCS '02*, pages 593–603, USA, 2002. IEEE Computer Society.
- 3 Eric Allender, Shiteng Chen, Tiancheng Lou, Periklis A. Papakonstantinou, and Bangsheng Tang. Width-parametrized SAT: time–space tradeoffs. *Theory of Computing*, 10:297–339, 2014. doi:10.4086/toc.2014.v010a012.
- 4 Hans L. Bodlaender, Gunther Cornelissen, and Marieke van der Wegen. Problems hard for treewidth but easy for stable gonality. *arXiv*, abs/2202.06838, 2022. Extended abstract to appear in Proceedings WG 2022. arXiv:2202.06838.
- 5 Hans L. Bodlaender, Carla Groenland, and Hugo Jacob. List colouring trees in logarithmic space. *arXiv*, abs/2206.09750, 2022. arXiv:2206.09750.
- 6 Hans L. Bodlaender, Carla Groenland, and Hugo Jacob. On the parameterized complexity of computing tree-partitions. *arXiv*, abs/2206.11832, 2022. arXiv:2206.11832.
- 7 Hans L. Bodlaender, Carla Groenland, and Hugo Jacob. XNLP-completeness for parameterized problems on graphs with a linear structure. *arXiv*, abs/2201.13119, 2022. arXiv:2201.13119.
- 8 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Marcin Pilipczuk, and Michał Pilipczuk. On the complexity of problems on tree-structured graphs. *CoRR*, 2022. doi:10.48550/arXiv.2206.11828.
- 9 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, 2021. doi:10.1109/FOCS52979.2021.00027.
- 10 Yijia Chen, Michael Elberfeld, and Moritz Müller. The parameterized space complexity of model-checking bounded variable first-order logic. *Log. Methods Comput. Sci.*, 15(3), 2019. doi:10.23638/LMCS-15(3:31)2019.
- 11 Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, STOC 1971*, pages 151–158. ACM, 1971. doi:10.1145/800157.805047.
- 12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 13 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM J. Comput.*, 24(4):873–921, 1995. doi:10.1137/S0097539792228228.
- 14 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. doi:10.1016/0304-3975(94)00097-3.
- 15 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999. doi:10.1007/978-1-4612-0515-9.
- 16 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.
- 17 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.
- 18 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Inf. Comput.*, 209(2):143–153, 2011. doi:10.1016/j.ic.2010.11.026.
- 19 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006. doi:10.1007/3-540-29953-X.

- 20 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theoretical Computer Science*, 918:60–76, 2022. doi:10.1016/j.tcs.2022.03.021.
- 21 Klaus Jansen and Petra Scheffler. Generalized coloring for tree-like graphs. *Discrete Applied Mathematics*, 75(2):135–155, 1997. doi:10.1016/S0166-218X(96)00085-6.
- 22 Takumi Kasai, Akeo Adachi, and Shigeki Iwata. Classes of pebble games and complete problems. *SIAM Journal on Computing*, 8(4):574–586, 1979. doi:10.1137/0208046.
- 23 Michal Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *ACM Transactions on Computation Theory*, 9(4):18:1–18:36, 2018. doi:10.1145/3154856.
- 24 Walter L. Ruzzo. Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21(2):218–235, 1980. doi:10.1016/0022-0000(80)90036-7.
- 25 Stefan Szeider. Not so easy problems for tree decomposable graphs. In *Advances in Discrete Mathematics and Applications: Mysore, 2008*, volume 13 of *Ramanujan Math. Soc. Lect. Notes Ser.*, pages 179–190. Ramanujan Math. Soc., Mysore, 2010. arXiv:1107.1177.
- 26 H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43(2):380–404, 1991. doi:10.1016/0022-0000(91)90020-6.

On the Parameterized Complexity of Computing Tree-Partitions

Hans L. Bodlaender  

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Carla Groenland  

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Hugo Jacob  

ENS Paris-Saclay, France

Abstract

We study the parameterized complexity of computing the tree-partition-width, a graph parameter equivalent to treewidth on graphs of bounded maximum degree.

On one hand, we can obtain approximations of the tree-partition-width efficiently: we show that there is an algorithm that, given an n -vertex graph G and an integer k , constructs a tree-partition of width $O(k^7)$ for G or reports that G has tree-partition width more than k , in time $k^{O(1)}n^2$. We can improve slightly on the approximation factor by sacrificing the dependence on k , or on n .

On the other hand, we show the problem of computing tree-partition-width exactly is XALP-complete, which implies that it is $W[t]$ -hard for all t . We deduce XALP-completeness of the problem of computing the domino treewidth. Finally, we adapt some known results on the parameter tree-partition-width and the topological minor relation, and use them to compare tree-partition-width to tree-cut width.

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

Keywords and phrases Parameterized algorithms, Tree partitions, tree-partition-width, Treewidth, Domino Treewidth, Approximation Algorithms, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.7

Related Version *Full Version*: <https://arxiv.org/abs/2206.11832> [8]

Funding *Carla Groenland*: Supported by the European Union's Horizon 2020 research and innovation programme under the ERC grant CRACKNP (number 853234) and the Marie Skłodowska-Curie grant GRAPHCOSY (number 101063180).

1 Introduction

Graph decompositions have been a very useful tool to draw the line between tractability and intractability of computational problems. There are many meta-theorems showing that a collection of problems can be solved efficiently if a decomposition of some form is given. By finding efficient algorithms to compute a decomposition if it exists, we deduce the existence of efficient algorithms even if the decomposition is not given. In particular, this proves useful when designing win-win arguments: for some problems, the existence of a solution and the existence of a decomposition are not independent, so that we can either use the decomposition for an efficient computation of the solution, or conclude that a solution must (or cannot) exist when there is no decomposition of small enough width.



© Hans L. Bodlaender, Carla Groenland, and Hugo Jacob;

licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 7; pp. 7:1–7:20

Leibniz International Proceedings in Informatics



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

The most successful notion of graph decomposition to date is certainly tree decompositions, and its corresponding parameter treewidth. Any problem expressible in MSO_2 ¹ can be solved in linear time in graphs of bounded treewidth due to a meta-theorem of Courcelle [13] and the algorithm of Bodlaender for computing an optimal tree decomposition [3]. Treewidth is a central tool in the study of minor-closed graph classes. A minor-closed graph class has bounded treewidth if and only if it contains no large grid minor.

In this paper, we focus on the parameter tree-partition-width (also called strong treewidth) which was independently introduced by Seese [27] and Halin [23]. It is known to have simple relations to treewidth [14, 29]: $\mathbf{tw} = O(\mathbf{tpw})$, and $\mathbf{tpw} = O(\Delta \mathbf{tw})$, where \mathbf{tw} , \mathbf{tpw} , Δ denote the treewidth, the tree-partition-width, and the maximum degree respectively. Applications of tree-partition-width include graph drawing and graph colouring [11, 22, 16, 17, 30, 2, 1]. Recently, Bodlaender, Cornelissen and Van der Wegen [5] showed for a number of problems (in particular, problems related to network flow) that these are intractable (XNLP-complete) when the pathwidth is used as parameter, but become fixed parameter tractable when parameterized by the width of a given tree-partition. This raises the question of the complexity of finding tree-partitions. We show that computing tree-partitions of approximate width is tractable.

► **Theorem 1.** *There is an algorithm that given an n -vertex graph G and an integer k , constructs a tree-partition of width $O(k^7)$ for G or reports that G has tree-partition width more than k , in time $k^{O(1)}n^2$.*

Thus, this removes the requirement from the results from [5] that a tree partition of small width is part of the input. Our technique is modular and allows us to also give alternatives running in FPT time or polynomial time with an improved approximation factor (see Theorem 10). Although not formulated as an algorithm, a construction of Ding and Oporowski [15] implies an FPT algorithm to compute tree-partitions of width $f(k)$ for graphs of tree-partition-width k , for some fixed computable function f . We adapt their construction and give some new arguments designed for our purposes. This significantly improves on the upper bounds to the width, and the running time.

The results from [5] are stated in terms of the notions of stable gonality, stable tree-partition-width and treebreadth. The notion of stability comes from the origin of the notion of gonality (from algebraic geometry); in graph terms, this implies that we look here at the minimum over all possible subdivision of edges. Tree-partition-width and stable tree-partition-width are bounded by polynomial functions of each other (see the appendix).

Related to tree-partition-width is the notion of domino treewidth, first studied by Bodlaender and Engelfriet [7]. A domino tree decomposition is a tree decomposition where each vertex is in at most two bags. Where graphs of small tree partition-width can have large degree, a graph of domino treewidth k has maximum degree at most $2k$. Bodlaender and Engelfriet show that DOMINO TREEWIDTH is hard for each class $W[t]$, $t \in \mathbb{N}$; we improve this result and show XALP -completeness.

► **Theorem 2.** *DOMINO TREEWIDTH and $\text{TREE PARTITION WIDTH}$ are XALP -complete.*

In [4], Bodlaender gave an algorithm to compute a domino tree decomposition of width $O(\mathbf{tw} \Delta^2)$ in $f(\mathbf{tw})n^2$ time for n -vertex graphs of treewidth \mathbf{tw} and maximum degree Δ , where f is a fixed computable function. This implies an approximation algorithm for domino treewidth.

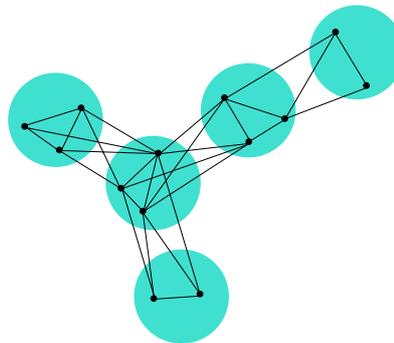
¹ Formulae with quantification over sets of edges or vertices, quantification over vertices and edges, and with the incidence predicate.

We also consider the parameter *tree-cut width* introduced by Wollan in [28]. As the tractability results of Bodlaender et al. [5] use techniques similar to a previous work on algorithmic applications of tree-cut width [21], one may wonder whether there is a relationship between tree-cut width and tree-partition-width. We answer this question in two steps. We obtain a parameter that is polynomially tied to tree-partition-width and is topological minor monotone. In particular, this shows that tree-partition-width is relatively stable with respect to subdivisions. Then, we show how to relate tree-cut width to the tree-partition-width of a subdivision. We show that a bound on tree-partition-width does not imply a bound on tree-cut width and that tree-partition-width is polynomially bounded by tree-cut width.

Paper overview

In Section 3, we provide our results on approximating the tree-partition-width. In Section 4, we show that computing the tree-partition-width is XALP-complete. We then derive XALP-completeness of computing the domino treewidth. Our results relating tree-cut width to tree-partition-width are in the appendix.

2 Preliminaries



■ **Figure 1** An example of tree-partition.

A *tree-partition* of a graph $G = (V, E)$ is a tuple $(T, (B_i)_{i \in V(T)})$, where $B_i \subseteq V(G)$, with the following properties.

- T is a tree.
- For each $v \in V$ there is a unique $i(v) \in V(T)$ such that $v \in B_{i(v)}$.
- For any edge $uv \in E$, either $i(u) = i(v)$ or $i(u)i(v)$ is an edge of T .

The *size* of a bag B_i is $|B_i|$, the number of vertices it contains. The *width* of the decomposition is given by $\max_{i \in V(T)} |B_i|$. The *tree-partition-width* (**tpw**) of a graph G is the minimum width of a tree-partition of G .

A *tree decomposition* of a graph $G = (V, E)$ is a pair $(T = (I, F), \{X_i \mid i \in I\})$ with $T = (I, F)$ a tree and $\{X_i \mid i \in I\}$ a family of (not necessarily disjoint) subsets of V (called *bags*) such that $\bigcup_{i \in I} X_i = V$, for all edges $vw \in E$, there is an i with $v, w \in X_i$, and for all v , the nodes $\{i \in I \mid v \in X_i\}$ form a connected subtree of T . The *width* of a tree decomposition $(T, \{X_i \mid i \in I\})$ is $\max_{i \in I} |X_i| - 1$, and the *treewidth* (**tw**) of a graph G is the minimum width over all tree decompositions of G . The *domino treewidth* is the minimum width over all tree decompositions of G such that each vertex appears in at most two bags.

We say that two parameters α, β are (*polynomially*) *tied* if there exist (polynomial) functions f, g such that $\alpha \leq f(\beta)$ and $\beta \leq g(\alpha)$.

3 Approximation algorithm for tree-partition-width

We first describe our algorithm, then prove correctness and finally discuss the trade-offs between running time and solution quality.

3.1 Description of the algorithm

Let G be a graph, and k any positive integer. We describe a scheme that produces a tree-partition of G of width $O(wbk^3) = k^{O(1)}$, or reports that $\mathbf{tpw}(G) > k$. We will use various different functions of k for b and w , depending on the quality/time trade-offs of the black-box algorithms inserted into our algorithm (e.g. for approximating treewidth).

Step 1. *Compute a tree decomposition for G of width $w(k)$ or conclude that $\mathbf{tpw}(G) > k$.* As mentioned above, we do not directly specify the function $w = w(k)$, since different algorithms for step 1 give different solution qualities (bounds for $w(k)$) and running times. Since $\mathbf{tw} + 1 \leq 2 \cdot \mathbf{tpw}$, if $\mathbf{tw}(G) > 2k - 1$ it follows that $\mathbf{tpw}(G) > k$. If we obtained a decomposition of width w , we also know that $\mathbf{tw}(G) \leq 2k - 1$, and hence there are at most $(2k - 1)n$ edges in G .

We set a threshold $b \geq \max\{2k - 1, w + 1\}$. We define an auxiliary graph G^b as follows. The vertex set of G^b is $V(G)$. The edges of G^b are given by the pairs of vertices $u, v \in V(G)$ with minimum u - v separator of size at least b .

Step 2. *Construct the auxiliary graph G^b with connected components of size at most k or report that $\mathbf{tpw}(G) > k$.*

We later describe several ways of computing the edges of G^b . We will show in Claim 3 that vertices in the same connected component of G^b must be in the same bag for any tree-partition of width at most k . For this reason, we conclude that $\mathbf{tpw}(G) > k$ if a component of G^b has more than k vertices.

We define H , the b -reduction of G , which is the graph obtained from G by identifying the connected components of G^b .

Step 3. *We compute a tree decomposition of width w for each 2-connected component of H .* Given the components of G^b , we can compute H , and its 2-connected components in time $O(k^{O(1)}n)$. Using Claim 4, we obtain a tree decomposition of H by replacing vertices of G , in the tree decomposition of G , by their component in G^b .

By Claim 5, the maximum degree Δ_H within the 2-connected components of H is at most Cbk^2 for some constant C when $\mathbf{tpw}(G) \leq k$.

Step 4. *If $\Delta_H > Cbk^2$, report $\mathbf{tpw}(G) > k$. Else, compute a tree partition of width $O(w\Delta_H) = O(wbk^2)$ for H .*

By rooting the decomposition of H in 2-connected components, we can define a *parent* cutvertex for each 2-connected component except the root. We separately compute tree partitions for each 2-connected component of H with the constraint that their parent cutvertex should be the single vertex of its bag. A construction of Wood [29] enables us to compute a tree-partition of width $O(\Delta w)$ for any graph of maximum degree Δ and treewidth w ; this can be adjusted to allow for this isolation constraint without increasing the upper bound on the width. We give the details of this in Corollary 7. After doing this, the partitions of each component can be combined without increasing the width. Indeed, although cutvertices are shared, only one 2-connected component will consider putting other vertices in its bag. We obtain a tree-partition of H of width $O(wbk^2)$.

Step 5. Deduce a tree partition of width $O(wbk^3)$ for G .

We “expand” the vertices of H . In the tree-partition of H , each vertex of H is replaced by the vertices of the corresponding connected component of G^b . This gives a tree-partition of G of width $O(wbk^3)$.

3.2 Correctness

For $s, t \in V(G)$ we denote by $\mu(s, t)$ the size of a minimum s - t separator in $G - st$.

▷ **Claim 3.** Let G be a graph and $s, t \in V(G)$.

- If $\mu(s, t) \geq k + 1$, then in any tree-partition of width at most k , s and t must be in adjacent bags or the same bag;
- If $\mu(s, t) \geq 2k - 1$, then in any tree-partition of width at most k , s and t must be in the same bag.

Proof. Assume that s and t are not in adjacent bags nor in the same bag of a tree-partition of width at most k , then any internal bag on the path between their respective bags is an s - t separator. In particular, $\mu(s, t) \leq k$. This proves the first point by contraposition.

Assume that s and t are in adjacent bags but not in the same bag for some tree-partition of width at most k . We denote their respective bags by B_s and B_t . Then, $(B_s \cup B_t) \setminus \{s, t\}$ is an s - t separator of $G - st$. Consequently, $\mu(s, t) \leq 2k - 2$. This proves the second point. ◁

▷ **Claim 4.** Consider $(T, (X_i)_{i \in V(T)})$ a tree decomposition of width w of G , $b \geq w + 1$, and let Y_i be the set of connected components of G^b that intersect with X_i . Then $(T, (Y_i)_{i \in V(T)})$ is a tree decomposition of the b -reduction H of G .

Proof. Every component of G^b appears in at least one Y_i , because it contains a vertex which must appear in at least one X_i . Furthermore, for each edge UV of H , there must be vertices $u \in U, v \in V$ such that uv is an edge of G . Hence, there is a bag X_i containing u and v so Y_i contains A and B . Finally, suppose that there is a bag Y_i not containing a component C of G^b , and several components of $T - i$ have bags containing C . There must be an edge of G^b connecting vertices u and v of C such that u is in bags of X and v is in X' , where X and X' are in different components of $T - i$. By definition of G^b there are at least b vertex disjoint u, v -paths in G , so the minimal size of a separator of u and v is at least $b \geq w + 1$. However, since the tree decomposition $(T, (X_i))$ has width w and the bags containing u are disjoint from the bags containing v (in particular X_i separates them), there is a separator of u and v of size at most w , a contradiction. This concludes the proof that $(T, (Y_i))$ is a tree decomposition of H . ◁

▷ **Claim 5.** If H is the b -reduction of G , $\text{tpw}(G) \leq k$, and B is one of its 2-connected components, then the maximum degree in B is $O(bk^2)$.

Proof. Consider u a vertex achieving maximum degree in B . By definition of B , $B - u$ is connected. We denote by N the neighbourhood of u in B . Let T be a spanning tree of $B - u$. We iteratively remove leaves that are not in N , and contract edges with an endpoint of degree 2 that is not in N . This produces the reduced tree T' . The maximum degree in this tree is $b - 1$ as the set of edges incident to a given vertex can be extended to disjoint paths leading to vertices in N .

We call the number of vertices in the component of G^b associated to a vertex v of H the weight of v and denote it $|v|$.

Clearly the neighbours of u must be either in the same bag as u or in a neighbouring bag. Since the bag of u will be a separator of vertices that are in distinct neighbouring bags, in particular, it splits the graph into several components each containing neighbours of u of total weight at most k .

There must exist a subset of vertices of T' of size at most $k - |u|$ whose removal splits T' in components containing vertices of N of total weight at most k . Since the degree of a vertex of T' is at most $b - 1$, removing one of its vertices adds at most $b - 2$ new components. Hence, after removing $k - |U| \leq k - 1$ vertices, there are at most $1 + (k - 1)(b - 2)$ components. We conclude that $|N| \leq k(1 + (k - 1)(b - 2))$. Since u had maximum degree in B , we conclude. \triangleleft

In [29], Wood shows the following lemma.

► **Lemma 6.** *Let $\alpha = 1 + 1/\sqrt{2}$ and $\gamma = 1 + \sqrt{2}$. Let G be a graph with treewidth at most $k \geq 1$ and maximum degree at most $\Delta \geq 1$. Then G has tree-partition-width $\text{tpw}(G) \leq \gamma(k + 1)(3\gamma\Delta - 1)$.*

Moreover, for each set $S \subseteq V(G)$ such that $(\gamma + 1)(k + 1) \leq |S| \leq 3(\gamma + 1)(k + 1)\Delta$, there is a tree-partition of G with width at most $\gamma(k + 1)(3\gamma\Delta - 1)$ such that S is contained in a single bag containing at most $\alpha|S| - \gamma(k + 1)$ vertices.

We deduce this slightly stronger version of [29, Theorem 1]

► **Corollary 7.** *From a tree decomposition of width w in a graph G of maximum degree Δ , for any vertex v of G , we can produce a tree-partition of G of width $O(\Delta w)$ in which v is the only vertex of its bag.*

Proof. We wish to apply Lemma 6 to $S \supseteq N(v)$. Let $\gamma = 1 + \sqrt{2}$. We have $|N(v)| \leq \Delta$, so in particular, $|N(v)| \leq 3(\gamma + 1)(w + 1)\Delta$. In case $|N(v)| < (\gamma + 1)(w + 1)$, we can add arbitrary vertices to $N(v)$ to form S satisfying $|S| \geq (\gamma + 1)(w + 1)$. Otherwise, we simply set $S = N(v)$. We then apply the lemma to S in $G - v$. There is a single bag that contains $N(v)$, and so we may add the bag $\{v\}$ adjacent to this in order to deduce a tree partition of G of width $O(\Delta w)$ in which v is the only vertex of its bag. \blacktriangleleft

3.3 Time/quality trade-offs

For Step 1, we consider the following algorithms to compute tree decompositions:

- An algorithm of Korhonen [25] computes a tree decomposition of width at most $2k + 1$ or reports that $\text{tw}(G) > k$ in time $2^{O(k)}n$.
- An algorithm of Fomin et al. [19] computes a tree decomposition of width $O(k^2)$ or reports that $\text{tw}(G) > k$ in time $O(k^7 n \log n)$.
- An algorithm of Feige et al. [18] computes a tree decomposition of width $O(k\sqrt{\log k})$ or reports that $\text{tw}(G) > k$ in time $O(n^{O(1)})$.

Recall that we denote by w the width of the computed tree decomposition of G .

We give two methods to compute G^b in step 2 of the algorithm.

- We can use a maximum-flow algorithm (e.g. Ford-Fulkerson [20]) to compute for each pair $\{s, t\}$ of vertices of G whether there are at least b vertex disjoint paths from s to t , in time $O(bkn)$. To compute a minimum vertex cut, replace each vertex v by two vertices $v_{\text{in}}, v_{\text{out}}$ with an arc from v_{in} to v_{out} . All arcs going to v should go to v_{in} , and all arcs leaving v should leave v_{out} . All arcs are given capacity 1. We may stop the maximum flow algorithm as soon as a flow of at least b was found. Furthermore, we can reduce the number of pairs $\{s, t\}$ of vertices to check to $O(wn)$, as each pair must be contained in a bag due to $b \geq w + 1$. This results in a total time of $O(wbkn^2)$.

- We can also use dynamic programming to enumerate all possible ways of connecting pairs of vertices that are in the same bag in time $2^{O(w^2)}n$, which is sufficient to compute G^b . A state of the dynamic programming consists of the subset of vertices of the bag that are used by the partial solution, a matching on some of these vertices, up to two vertices that were decided as endpoints of the constructed paths, the number of already constructed paths between the endpoints, and two disjoint subsets of the used vertices that are not endpoints, nor in the matching such that we found a disjoint path from the first or second endpoint to them. We first tabulate answers for each subtree of the decomposition by starting from the leaves, and then tabulate answers for each complement of a subtree by starting from the root and, when branching to some child, combining with the partial solutions of the subtree of the other child.

The b -reduction H of G and its 2-connected components can be computed in $O(k^{O(1)}n)$ time (see e.g. [24]), since the size of the graph is $O(k^{O(1)}n)$ here.

We will now make use of the following result due to Bodlaender and Hagerup [10]:

- **Lemma 8.** *There is an algorithm that given a tree decomposition of width k with $O(n)$ nodes of a graph G , finds a rooted binary tree of G of width at most $3k + 2$ with depth $O(\log n)$ in $O(kn)$ time.*

When implementing the construction of Wood for 2-connected components of H , the running time is dominated by $O(n)$ queries to find a balanced separator with respect to a set W of size $k^{O(1)}$. After a preprocessing in time $O(kn)$, we can do this in time $k^{O(1)}d$ where d is the diameter of our tree decomposition. We first obtain a binary balanced decomposition using Lemma 8, then reindex the vertices in such a way that we can check if a vertex is in some bag of a given subtree of tree decomposition in constant time. Using this, we can in time $k^{O(1)}$ determine whether a bag is a balanced separator of W , and if not move to the subtree containing the most vertices of W . This procedure will consider at most d bags, hence the total running time of $k^{O(1)}d$. Since the decomposition has depth $O(\log n)$ it also has diameter $d = O(\log n)$. Hence the construction of Wood can be executed in time $k^{O(1)}n \log n$.

- **Lemma 9.** *We can compute a tree partition of width $O(\Delta \mathbf{tw})$ in time $O(k^{O(1)}n \log n)$ when given a tree decomposition of width $k^{O(1)}$.*

By combining the previous algorithms we obtain the following theorem.

- **Theorem 10.** *There is a polynomial time algorithm that constructs a tree-partition of width $O(k^5 \log k)$ or reports that the tree-partition-width is more than k .*

There is an algorithm running in time $2^{O(k^2)}n + k^{O(1)}n \log n$ that computes a tree-partition of width $O(k^5)$ or reports that the tree-partition-width is more than k .

There is an algorithm running in time $k^{O(1)}n^2$ that computes a tree-partition of width $O(k^7)$ or reports that the tree-partition-width is more than k .

Proof. The first algorithm uses the algorithm of Feige et al. to compute the tree decomposition, then naively computes G^b , and then finds balanced separators for Wood's construction using the tree decomposition in polynomial time (no need to balance the decomposition).

The second algorithm uses Korhonen's algorithm to compute the tree decomposition, then computes G^b using the dynamic programming approach, and then finds balanced separators for Wood's construction as described.

The third algorithm uses the algorithm of Fomin et al. to compute the tree decomposition, then computes G^b via a maximum-flow algorithm in time $O(wbkn^2) = O(k^5n^2)$, and then finds balanced separators for Wood's construction as described.

The guarantees on the width follow from the analysis of our scheme. ◀

4 XALP-completeness of Tree Partition Width

In this section, we show that the TREE PARTITION WIDTH problem is XALP-complete, even when we use the width target and the degree as combined parameter. As a relatively simple consequence, we obtain that DOMINO TREEWIDTH is XALP-complete.

XALP is the class of all parameterized problems that can be solved in $f(k)n^{O(1)}$ time and $f(k)\log n$ space on a nondeterministic Turing Machine with access to a push-down automaton, or equivalently way of the class of problems that can be solved by an alternating Turing Machine in $f(k)n^{O(1)}$ treesize and $f(k)\log(n)$ space. An alternating Turing Machine (ATM) is nondeterministic Turing Machine with some extra states where we ask for all of the transitions to lead to acceptance. This creates independent configurations that must all lead to acceptance, and we call “co-nondeterministic step” the process of obtaining these independent configurations.

XALP is closed by reductions using at most $f(k)\log n$ space and running in FPT time. These two conditions are implied by using at most $f(k) + \log n$ space. We call reductions respecting the latter condition *parameterized logspace* reductions (or pl-reductions).

This class is relevant here because the problems we consider are complete for it. Completeness for XALP has the following consequences: $W[t]$ -hardness for all positive integers t , membership in XP, and there is a conjecture that XP space is required for algorithms running in XP time. If the conjecture holds, this roughly means that the dynamic programming algorithm used for membership is optimal.

The following problem is shown to be XALP-complete in [9] and starts as the starting point of our reduction.

TREE-CHAINED MULTICOLOR INDEPENDENT SET

Input: A tree $T = (V_T, E_T)$, an integer k , and for each $i \in V_T$, a collection of k pairwise disjoint sets of vertices $V_{i,1}, \dots, V_{i,k}$ and a graph G with vertex set $V = \bigcup_{i \in V_T, j \in [1,k]} V_{i,j}$

Parameter: k

Question: Is there a set of vertices $W \subset V$, such that W contains exactly one vertex from each $V_{i,j}$ ($i \in V_T, j \in [1,k]$), and for each pair $V_{i,j}, V_{i',j'}$ with $i = i'$ or $ii' \in E_T$, $j, j' \in [1,k]$, $(i,j) \neq (i',j')$, the vertex in $W \cap V_{i,j}$ is non-adjacent to the vertex in $W \cap V_{i',j'}$?

We further use that we can assume the tree T to be binary without loss of generality (see [9] for more details).

► **Lemma 11.** TREE PARTITION WIDTH is in XALP.

Proof. To keep things simple, we will use as a black box the fact that reachability in undirected graphs can be decided in logspace [26]. We assume that the vertices have some arbitrary ordering σ .

For now, assume that the given graph is connected.

We begin by guessing at most k vertices to form an initial bag B_0 , and have an empty parent bag P_0 . We will recursively extend a partial tree-partition in the following manner. Suppose that we have a bag B with parent bag P , we must find a child bag for B in each connected component of $G - B$ that does not contain a vertex of P . We use the fact that a connected component can be identified by its vertex appearing first in σ , that the restriction of σ to these representatives gives an ordering on σ , and that we can compute such representatives in logspace. Let us denote by c the current vertex representative of a connected component of $G - B$. c is initially the first vertex in σ that is not in B and

cannot reach P in $G - B$. We do a co-nondeterministic step so that in one branch of the computation we find a tree-partition for the connected component with representative c , and in the other branch we find the representative of the next connected component. The representative c' of the next component is the first vertex in σ such that it cannot reach a vertex appearing before c (inclusive) in σ , nor a vertex of P in $G - B$. When found, c is replaced by c' and we repeat this computation. If we don't find such a vertex c' , then c must have represented the last connected component, so we simply accept.

Let us now describe what happens in the computation branch where we compute a new bag. We can iterate on vertices in the component of c , by iterating on vertices of $G - B$ and then skipping if they are not reachable from c in $G - B$. In particular, we can guess a subset B' of size at most k of vertices from this component. We then check that the neighbourhood of B in this component is contained in B' . If it is the case, we can set $P := B$ and $B := B'$ and recurse. If not, we reject.

If the graph is not connected, we can iterate on its connected components by using the same technique of remembering a vertex representative. For each of these components, we apply the above algorithm, with the modification that in each enumeration of the vertices we skip the vertex if it is not contained in the current component.

During these computations, we store at most $3k + O(1)$ vertices and use logspace subroutines. Furthermore, the described computation tree is of polynomial size. ◀

We first give a brief sketch of the structure of the hardness proof. We have a *trunk* gadget to enforce the shape of the tree from the TREE-CHAINED MULTICOLOR INDEPENDENT SET. On the trunk are attached *clique chains* which are longer than the part of trunk between their endpoints, and have some wider parts at some specific positions. The length of the chain gives us some slack which will be used to encode the choice of a vertex for some subset $V_{i,j}$. Based on the edges of G , we adjust the width along the trunk so that only one clique chain may place its wider part on each position of the trunk. In other words, part of the trunk is a collection of gadgets representing edges of G that allow for only one incident vertex to be chosen.

► **Lemma 12.** TREE PARTITION WIDTH *with target width and maximum degree as combined parameter is XALP-hard.*

Proof. We reduce from TREE-CHAINED MULTICOLOR INDEPENDENT SET.

Suppose that we are given a binary tree $T = (V_T, E_T)$, and for each node $i \in V_T$, a k -colored vertex set V_i . We denote the colors by integers in $[1, k]$, and write $V_{i,j}$ for the set of vertices in V_i with color j . We are also given a set of edges E of size m . Each edge in E is a pair of vertices in $V_i \times V_{i'}$ with $i = i'$ or ii' an edge in E_T . We can assume the edges are numbered: $E = \{e_1, \dots, e_m\}$.

In the TREE-CHAINED MULTICOLOR INDEPENDENT SET problem, we want to choose one vertex from each set $V_{i,j}$, $i \in V_T$, $j \in [1, k]$, such that for each edge $ii' \in E_T$, the chosen vertices in $V_i \cup V_{i'}$ form an independent set (which thus will be of size $2k$).

We assume that each set $V_{i,j}$ is of size r . (If not, we can add vertices adjacent to all other vertices in $V_{i,j}$, $j \in [1, k]$.) Write $V_{i,j} = \{v_{i,j,1}, v_{i,j,2}, \dots, v_{i,j,r}\}$.

Let $N = (m + 1)r$. Let $L = 36k + 5$.

Cluster Gadgets. In the construction, we use a *cluster gadget*. Suppose Z is a clique. Adding a cluster gadget for Z is the following operation on the graph that is constructed. Add a clique with vertex set $C_Z = \{c_{Z,1}, c_{Z,2}, \dots, c_{Z,2L}\}$ of size $2L$ to the graph, and add an edge between each vertex in Z and each vertex $c_{Z,j}$, $1 \leq j \leq L$, i.e. Z with the first L vertices in C_Z forms a clique.

7:10 On the Parameterized Complexity of Computing Tree-Partitions

In a tree partition of a graph, the vertices of a clique can belong to at most two different bags. The cluster gadget ensures that the vertices of clique Z belong to exactly one bag. This cluster gadget will be used in two different steps in the construction of the reduction.

► **Lemma 13.** *Suppose a graph H contains a clique Z with the cluster gadget for Z . In each tree partition of H of width at most L , there is a bag that contains all vertices from Z .*

Proof. There must be two adjacent bags that contain the vertices of C_Z and no other vertices. Similarly, there must be two adjacent bags containing all vertices in $Z \cup \{c_{Z,1}, \dots, c_{Z,L}\}$. This forces all vertices in $\{c_{Z,1}, \dots, c_{Z,L}\}$ to be in a single bag, and all vertices in Z to be in a single adjacent bag. ◀

A subdivision of T . The first step in the construction is to build a tree $T' = (V_{T'}, E_{T'})$, as follows. Choose an arbitrary node i from V_T . Add a new neighbor i' to i . Add a new neighbor r_0 to i' . Now subdivide each edge N times. The resulting tree is $T' = (V_{T'}, E_{T'})$. We view T' as a rooted tree, with root r_0 . We will use the word *grandparent* to refer to the parent of the parent of a vertex. The nodes that do not result from the subdivisions are referred to as *original nodes*. Nodes $i \in V_T$ and their copies in T' will be denoted with the same name.

The graph H consists of two main parts: the *trunk* and the *clique chains*. To several cliques in these parts, we add cluster gadgets.

The trunk. The trunk is obtained by taking for each node $i \in V_T$ a clique A_i . We specify below the size of these cliques. For each edge ii' in T' , we add an edge between each vertex in A_i to each vertex in $A_{i'}$. We add for each A_i a cluster gadget.

To specify the sizes of sets A_i , we first need to give some definitions:

- For each node $i' \in V_{T'}$, we let $p(i')$ be the number of nodes $i \in V_T$ (i.e., “original nodes”), such that i' is on the path (including endpoints) in T' from i to the vertex that is the grandparent of i in T . I.e., for each original node i , we look to the grandparent of i (if it exists), and then add 1 to the count of each node i' on the path between them in T' .
- For each edge $e_j \in E$, let $g(e_j) = 2jr$.
- For each edge $e_j = \{v_{i,c,s}, v_{i',c',s'}\}$, we have that $i = i'$ or i' is a child of i . Let i_{e_j} be the node in T' , obtained by making $g(e_j)$ steps up in T' from i : i.e., i_{e_j} is the ancestor of i with distance $g(e_j)$ in T' .

Now, for all nodes $i \in V_{T'}$,

- $|A_i|$ equals $L - 6k \cdot p(i) - 1$, if $i = i_{e_j}$ for some $e_j \in E$. At this node, we will verify that a choice (encoded by the clique chains, explained below), indeed gives an independent set: we check that we did not choose both endpoints of e_j .
- $|A_i|$ equals $L - 6k \cdot p(i) - 2$, otherwise.

The clique chains. For each $i \in V_T$, and each color class $c \in [1, k]$, we have a clique chain with $2N + r + 5$ cliques, denoted $CC'_{i,c,j}$, $j \in [1, 2N + r + 5]$. All vertices in the first clique $CC'_{i,c,1}$ are made incident to all vertices in A_i . All vertices in the last clique $CC'_{i,c,2N+r+5}$ are made incident to all vertices in $A_{i'}$ with i' the parent of the parent (i.e., the grandparent) of i in T . (Notice that the distance from i to i' in T' equals $2(N+1)$.) All vertices in $CC'_{i,c,j}$ are made incident to all vertices in $CC'_{i,c,j+1}$, i.e., all vertices in a clique are adjacent to all vertices in the next clique in the chain.

To each clique $CC'_{i,c,j}$ we add a cluster gadget.

The cliques have different sizes, which we now specify. Consider $i \in V_T$, $c \in [1, k]$, $\gamma \in [1, 2N + r + 5]$. The size of $CC_{i,c,\gamma}$ equals:

- $L - 7$, if $\gamma = 1$ or $\gamma = 2N + r + 5$ (i.e., for the first and last clique in the chain.)
- 7, if there is an edge e_j with one endpoint in $V_{i,c}$ for which one of the following cases holds:
 - $e_j = \{v_{i,c,\alpha}, v_{i',c',\alpha'}\}$, $c \neq c'$, i.e., one endpoint is in $V_{i,c}$, and the other endpoint is in another color class in V_i , and $\gamma = g(e_j) + 1 + \alpha$.
 - $e_j = \{v_{i,c,\alpha}, v_{i',c',\alpha'}\}$, i' is a child of i , and $\gamma = g(e_j) + 1 + \alpha$.
 - $e_j = \{v_{i,c,\alpha}, v_{i',c',\alpha'}\}$, i is a child of i' , and $\gamma = g(e_j) + N + 2 + \alpha$.
- 6, otherwise

Let H be the resulting graph.

► **Lemma 14.** *H has tree partition width at most L , if and only if the given instance of TREE-CHAINED MULTICOLOR INDEPENDENT SET has a solution.*

Proof. Suppose we have a solution of the TREE-CHAINED MULTICOLOR INDEPENDENT SET. Suppose for each class $V_{i,c}$, we choose the vertex $v_{i,c,h(i,c)}$. Now, we can construct the tree partition as follows. First, we take the tree T' , and for each node i in T' , we take a bag initially containing the vertices in A_i ; we later add more vertices to these bags in the construction.

Now, we add the chains, one by one. For a chain for $V_{i,c}$, take a new bag that contains $CC_{i,j,1}$, and make this bag incident to the bag of i . We add the vertices of $CC_{i,c,h(i,c)+1}$ to the bag of i . If $h(i,c) > 1$, then we place the vertices of cliques $CC_{i,c,\alpha+1}$ with $1 < \alpha < h(i,c)$ in bags outside the trunk: $CC_{i,c,h(i,c)}$ goes to the bag with $CC_{i,c,1}$; to this bag, we add an adjacent bag with $CC_{i,c,2} \cup CC_{i,j,h(i,c)-1}$; to this, we add an adjacent bag with $CC_{i,j,3} \cup CC_{i,j,h(i,c)-2}$, etc.

Now, add the vertices of $CC_{i,c,h(i,c)+2}$ to the bag of the parent of i , and continue this: each next clique is added to the next parent bag, until we add a clique to the bag of the parent of the parent of i in T ; name this node here i'' . Add a new bag incident to i'' and put $CC_{i,c,2N+r+5}$ in this bag (i.e., the last clique of the chain). Similar as at the start of the chain, fold the end of the chain (with possibly some additional new bags) such that a bag containing $CC_{i,c,2N+r+4}$ is adjacent to the bag with $CC_{i,c,2N+r+5}$.

Finally, for each cluster gadget, add two new bags, with the first incident to the bag containing the respective clique.

One easily verifies that this gives a tree partition of H . For bags outside the trunk, one easily observes that the size is at most L . Bags i in the trunk contain a set A_i , and precisely $p(i) \cdot k$ cliques of the clique chains: for each path that counts for the bag, and each color class in $[1, k]$, we have one chain with one clique. Each of these cliques has size six or seven. Now, we can notice that a clique of size 7 corresponds to an edge $e_{j'}$ with endpoint in the class. This clique will be mapped to a node in the trunk that equals $i_{e_{j'}}$, if and only if this endpoint is chosen; otherwise, the clique will be mapped to a trunk node with distance less than r to $i_{e_{j'}}$. Thus, there are two cases for a trunk node i :

- There is no edge e_j with $i = i_{e_j}$. Then, a close observation of the clique chains shows that there are at most two clique chains with size 7 mapped to i . Indeed, the construction is such that each edge has its private interval, and affects the trunk both between i and its parent i' , and between i' and its parent i'' .
- $i = i_{e_j}$. Now, at most one endpoint of e_j is in the solution. The clique chain of the color class of that endpoint can have a clique of size 7 mapped to i . The “offset” of the clique chain for the color class of the other endpoint is such that there is a clique of size 6 for that chain at i .

7:12 On the Parameterized Complexity of Computing Tree-Partitions

In both cases, the total size of the bag at i is at most L . Thus, the width of the tree partition is at most L .

Suppose we have a tree partition of H of width L . First, by the use of the cluster gadgets, each clique A_i is in one bag. A bag cannot contain two cliques A_i as each has size larger than $L/2$. Now, the bags containing A_i form a subtree of the partition tree that is isomorphic to T' . For each clique chain of a class $V_{i,c}$, we have that the first clique $CC_{i,c,1}$ is in a bag incident to i , and the last clique $CC_{i,c,2N+r+5}$ is in a bag incident to the trunk bag that corresponds to the grandparent of i in T , say i'' . Each trunk bag from i to i'' thus must contain a clique (of size 6 or 7) from the clique chain of $V_{i,c}$. It follows that each trunk bag i' contains at least $p(i) \cdot k$ cliques of size at least 6 each of the clique chains. Now, $|A_{i'}| + 6p(i) \cdot k \in \{L-1, L-2\}$, and thus we cannot add another clique of a clique chain to a trunk bag.

For a clique chain of $V_{i,c}$, there is a clique mapped to the trunk bag of i . Suppose $CC_{i,c,h(i,c)}$ is mapped to i . We claim that choosing from each $V_{i,c}$ the vertex $v_{i,c,h(i,c)}$ gives an independent set, and thus, we have a solution of the TREE CHAINED MULTICOLOR INDEPENDENT SET problem.

The vertices of $CC_{i,c,h(i,c)+2}$ must be mapped to the bag of the parent of i , as otherwise, i will contain an additional clique of size at least 6, and the size of the bag of i will become larger than L . By induction, we have that the α th parent of i , $\alpha \in [1, 2N+2]$ contains the vertices of $CC_{i,c,h(i,c)+\alpha+1}$. (Note that the $(2N+2)$ nd parent equals the node corresponding the grandparent of i in T .)

We now consider the node i_{e_j} for edge $e_j \in E$. Suppose $e_j = v_{i,c,\alpha}v_{i',c',\alpha'}$. Without loss of generality, suppose $i = i'$ or i' is a child of i ; otherwise, switch roles of i and i' . For each endpoint of this edge, if the endpoint is chosen (i.e., $\alpha = h(i,c)$ or $\alpha' = h(i',c')$), then the corresponding clique chain has a clique of size 7 in the bag i_{e_j} . This can be seen by the following case analysis:

- By assumption, $CC_{i,c,h(i,c)+1}$ is placed in the bag of i . As each successive clique in the chain is placed in one higher bag along the path from i to the grandparent of i (in T), we have that $CC_{i,c,h(i,c)+g(e_j)+1}$ is placed in the bag of i_{e_j} , as this node is the $g(e_j)$ th parent of i in T' . This clique has size 7.
- If $i' = i$, the same argument shows that $CC_{i',c',h(i',c')+1}$ is a clique of size 7 placed in the bag of i_{e_j} .
- If i' is a child of i in T , then $CC_{i',c',h(i',c')+1}$ is placed in the bag of i' . Again, each successive clique in the chain of $V_{i',c'}$ is placed in the next parent bag, for all nodes on the path from i' to the grandparent of i' in T (which is the parent of i in T .) This implies that $CC_{i',c',h(i',c')+N+2}$ is placed in the bag of i and $CC_{i',c',h(i',c')+N+2+g(e_j)}$ is placed in the bag of i_{e_j} ; this bag has size 7.

Now, if both endpoints would be chosen, then the size of the bag of i_{e_j} is larger than L : it contains $A_{i_{e_j}}$ (which has size $L - 6kp(i_{e_j}) - 1$), $6p(i_{e_j})$ bags of clique chains, of which all have size at least 6 and two have size 7; contradiction. So, at most one endpoint is chosen, so choosing vertices $v_{i,c,h(i,c)}$ gives an independent set. ◀

The maximum degree of a vertex in H is less than $5kL + 5L = O(k^2)$:

- Vertices in cluster gadgets have maximum degree less than $2L$.
- A vertex in a trunk clique A_i of a node i that resulted from a subdivision have maximum degree less than $4L$ as i has two incident nodes, each with a trunk clique of size less than L , and there is a cluster gadget attached to A_i .

- A vertex in a trunk clique A_i of a node i that is an original node (i.e., also in T) have less than L neighbors in A_i , less than $3L$ neighbors in $A_{i'}$ with i' incident to i in T' , less than kL neighbors of cliques $CC_{i,c,1}$ (one clique of size $L - 7$ for each class $c \in [1, k]$), less than $4kL$ neighbors of cliques $CC_{i',c,2N+r+5}$ (one clique of size $L - 7$ for each node of which i is the grandparent in T for each class $c \in [1, k]$), and less than L neighbors in the cluster gadget attached to A_i .

Finally, we conclude that the transformation can be carried out in $f(k) \log n$ space, thus the result follows. ◀

From Lemmas 11 and Lemma 12, the following result directly follows.

► **Theorem 15.** TREE PARTITION WIDTH is XALP-complete, both when the target value, and when the target value plus the maximum degree is used as parameter.

► **Theorem 16.** DOMINO TREEWIDTH is XALP-complete.

Proof. Membership: We use the fact that the maximum degree of the graph is bounded by $2k$ where k is the domino treewidth. We can discard an instance where this condition on the maximum degree is not satisfied in logspace. We first assume that the given graph is connected.

The “certificate” used for this computation will be of size $O(k^2 + k \log(n))$ and consists of:

- The current bag and for each of its vertices whether it was contained in a previous bag or not. This requires at most $k + k \log(n)$ bits.
- For each neighbour of the bag, whether it was already covered by a bag. This requires $O(k^2)$ bits.

The algorithm works as follows. Given the current certificate, if all neighbours have been covered we accept. Otherwise, we guess a new child bag by picking a non-empty subset of $k + 1$ vertices among the vertices of the current bag that were contained only in this bag, and the neighbours that were not already covered. We then check that each vertex that is in both the current bag and child bag has all of its neighbours in these two bags. We then guess for each not already covered neighbour of the current bag if it should be covered by the subtree of this child. These vertices are then considered as covered in the current bag certificate. In the child bag certificate, the non covered neighbours are these vertices and the neighbours of the child bag that are not neighbours of the parent bag. We then recurse with both certificates, and accept if both recursions accept.

This computation uses $O(k^2 + k \log n)$ space and the computation tree has polynomial size.

We can handle disconnected graphs by iterating on component representatives and discarding vertices that are not reachable using the fact that reachability in undirected graphs can be computed in logspace (see membership for TREE PARTITION WIDTH for more details).

Hardness follows from a reduction from TREE PARTITION WIDTH when we use the target value and maximum degree as parameter.

Suppose we are given a graph $G = (V, E)$ of maximum degree d and an integer k .

Let $L = kd + 1$, and $M = (k + 1)L - 1$. Now, build a graph H as follows. For each vertex $v \in V$, we take a clique C_v with L vertices. For each vertex $w \in C_v$, we add a set S_w with $2M - 2$ vertices, and make one of the vertices in S_w incident to w and to all other vertices in S_w ; call this vertex y_w .

For each edge $e = vw \in E$, we add a vertex z_e , and make z_e incident to all vertices in C_v and all vertices in C_w . Let H be the resulting graph.

► **Lemma 17.** *G has tree partition width at most k , if and only if H has domino treewidth at most $M - 1$.*

Proof. Suppose H has domino treewidth at most $M - 1$. Suppose $(\{X_i | i \in I\}, T = (I, F))$ is a domino tree decomposition of H of width at most $M - 1$, i.e., each bag has size at most M .

First, consider a vertex w in some C_v . The vertex y_w has degree $2M - 2$, which implies that there are two adjacent bags that each contain y_w , and $M - 1$ neighbors of y_w . One of these bags contains w .

For each $v \in V$, there must be at least one bag that contains all vertices of C_v , by a well known property of tree decompositions. There can be also at most one such bag, because each vertex $w \in C_v$ is in another bag that is filled by w , y_w , and $M - 2$ other neighbors of y_w .

Let for $i \in I$, $Y_i \subseteq V$ be the set of vertices $v \in V$ with $C_v \subseteq X_i$. We claim that $(\{Y_i | i \in I\}, T = (I, F))$ is a tree partition of G of width at most k (some bags are empty). First, by the discussion above, each vertex $v \in V$ belongs to exactly one bag Y_i . Second, as $M < (k + 1)L$, each Y_i has size at most k . Third, if we have an edge $e = vw \in E$, then z_e is in the bag that contains C_v , and z_e is in the bag that contains C_w . As z_e is in at most two bags, these two bags must be the same, or adjacent, so in $(\{Y_i | i \in I\}, T = (I, F))$, v and w are in the same set Y_i or in sets Y_i and $Y_{i'}$ with i and i' adjacent in the decomposition tree T .

Now, suppose G has tree partition width at most k , say with tree partition $(\{Y_i | i \in I\}, T = (I, F))$. For each $i \in I$, let $X_i = \bigcup_{v \in Y_i} C_v \cup \{z_e \mid \exists v \in Y_i, w \in V : e = vw\}$. For each $v \in V$, $w \in C_v$, add two bags, one containing w , y_w , and $M - 2$ other neighbors of y_w , and the other containing y_w and the remaining $M - 1$ neighbors of y_w , and make these bags adjacent, and the first adjacent to the bag in T that also contains w . One easily verifies that this results in a domino tree decomposition of H with maximum bags size at most M , hence H has domino treewidth at most $M - 1$. ◀

It is easy to see that H can be constructed from G with $f(k) \log |V|$ memory. So, the hardness of DOMINO TREewidth follows from the previous lemma. ◀

5 Conclusion

We settle the question of the exact computation of tree-partition-width, and show that its approximation is tractable. However, many questions remain regarding approximation algorithms:

- Is a constant factor approximation tractable?
- Can we improve the approximation ratios with similar running times?
- Can we improve the dependence on n to something linear in n ?

Regarding the running time of the approximation algorithm, the following results could improve the final running times. Firstly, although the techniques of [6] are nontrivial, it seems reasonable to hope that they can be adapted to implement Wood's construction in time $f(k)n$. This would give an algorithm running in time $f(k)n$ to compute a tree-partition of width $k^{O(1)}$. Secondly, is there some value of b polynomial in w and k such that we can compute G^b in time $k^{O(1)}n \log n$ or in time $2^{o(k^2)}n$? This would directly give faster running times for the approximation algorithm, possibly at the cost of a worse approximation ratio.

The parameter treebreadth studied in [5] is polynomially tied to tree-partition-width, but might be easier to approximate.

Another interesting direction is to study the complexity of computing (approximate) tree decompositions on graphs of bounded tree-partition-width.

References

- 1 Noga Alon, Guoli Ding, Bogdan Oporowski, and Dirk Vertigan. Partitioning into graphs with only small components. *J. Comb. Theory, Ser. B*, 87(2):231–243, 2003. doi:10.1016/S0095-8956(02)00006-0.
- 2 János Barát and David R. Wood. Notes on nonrepetitive graph colouring. *Electron. J. Comb.*, 15(1), 2008. URL: http://www.combinatorics.org/Volume_15/Abstracts/v15i1r99.html.
- 3 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.
- 4 Hans L. Bodlaender. A note on domino treewidth. *Discret. Math. Theor. Comput. Sci.*, 3(4):141–150, 1999. URL: <http://dmtcs.episciences.org/256>.
- 5 Hans L. Bodlaender, Gunther Cornelissen, and Marieke van der Wegen. Problems hard for treewidth but easy for stable gonality. *arXiv*, abs/2202.06838, 2022. Extended abstract to appear in Proceedings WG 2022. arXiv:2202.06838.
- 6 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michał Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. doi:10.1137/130947374.
- 7 Hans L. Bodlaender and Joost Engelfriet. Domino treewidth. *J. Algorithms*, 24(1):94–123, 1997. doi:10.1006/jagm.1996.0854.
- 8 Hans L. Bodlaender, Carla Groenland, and Hugo Jacob. On the parameterized complexity of computing tree-partitions. *arXiv*, abs/2206.11832, 2022. URL: <https://arxiv.org/abs/2206.11832>.
- 9 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Marcin Pilipczuk, and Michał Pilipczuk. On the complexity of problems on tree-structured graphs. *CoRR*, 2022. doi:10.48550/ARXIV.2206.11828.
- 10 Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM J. Comput.*, 27(6):1725–1746, 1998. doi:10.1137/S0097539795289859.
- 11 Paz Carmi, Vida Dujmovic, Pat Morin, and David R. Wood. Distinct distances in graph drawings. *Electron. J. Comb.*, 15(1), 2008. URL: http://www.combinatorics.org/Volume_15/Abstracts/v15i1r107.html.
- 12 Julia Chuzhoy and Zihan Tan. Towards tight(er) bounds for the excluded grid theorem. *J. Comb. Theory, Ser. B*, 146:219–265, 2021. doi:10.1016/j.jctb.2020.09.010.
- 13 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.
- 14 Guoli Ding and Bogdan Oporowski. Some results on tree decomposition of graphs. *J. Graph Theory*, 20(4):481–499, 1995. doi:10.1002/jgt.3190200412.
- 15 Guoli Ding and Bogdan Oporowski. On tree-partitions of graphs. *Discret. Math.*, 149(1-3):45–58, 1996. doi:10.1016/0012-365X(94)00337-I.
- 16 Vida Dujmovic, Pat Morin, and David R. Wood. Layout of graphs with bounded tree-width. *SIAM J. Comput.*, 34(3):553–579, 2005. doi:10.1137/S0097539702416141.
- 17 Vida Dujmovic, Matthew Suderman, and David R. Wood. Graph drawings with few slopes. *Comput. Geom.*, 38(3):181–193, 2007. doi:10.1016/j.comgeo.2006.08.002.
- 18 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, 38(2):629–657, 2008. doi:10.1137/05064299X.
- 19 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michał Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Trans. Algorithms*, 14(3):34:1–34:45, 2018. doi:10.1145/3186898.
- 20 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.

- 21 Robert Ganian, Eun Jung Kim, and Stefan Szeider. Algorithmic applications of tree-cut width. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, volume 9235 of *Lecture Notes in Computer Science*, pages 348–360. Springer, 2015. doi:10.1007/978-3-662-48054-0_29.
- 22 Emilio Di Giacomo, Giuseppe Liotta, and Henk Meijer. Computing straight-line 3d grid drawings of graphs in linear volume. *Comput. Geom.*, 32(1):26–58, 2005. doi:10.1016/j.comgeo.2004.11.003.
- 23 Rudolf Halin. Tree-partitions of infinite graphs. *Discret. Math.*, 97(1-3):203–217, 1991. doi:10.1016/0012-365X(91)90436-6.
- 24 John Hopcroft and Robert Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973. doi:10.1145/362248.362272.
- 25 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 184–192. IEEE, 2021. doi:10.1109/FOCS52979.2021.00026.
- 26 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008. doi:10.1145/1391289.1391291.
- 27 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.
- 28 Paul Wollan. The structure of graphs not admitting a fixed immersion. *J. Comb. Theory, Ser. B*, 110:47–66, 2015. doi:10.1016/j.jctb.2014.07.003.
- 29 David R. Wood. On tree-partition-width. *Eur. J. Comb.*, 30(5):1245–1253, 2009. doi:10.1016/j.ejc.2008.11.010.
- 30 David R. Wood and Jan Arne Telle. Planar decompositions and the crossing number of graphs with an excluded minor. In Michael Kaufmann and Dorothea Wagner, editors, *Graph Drawing, 14th International Symposium, GD 2006, Karlsruhe, Germany, September 18-20, 2006. Revised Papers*, volume 4372 of *Lecture Notes in Computer Science*, pages 150–161. Springer, 2006. doi:10.1007/978-3-540-70904-6_16.

A Tree-cut width and the stability of tree-partition-width

In this section, we consider the relation of the notion of tree-cut width with (stable) tree-partition-width. Tree-cut width was introduced by Wollan [28]. Ganian et al. [21] showed that several problems that are $W[1]$ -hard with treewidth as parameter are fixed parameter tractable with tree-cut width as parameter.

We begin by defining the *tree-cut width* of a graph $G = (V, E)$. A tree-cut decomposition (T, \mathcal{X}) consists of a rooted tree T and a family \mathcal{X} of *bags* $(X_i)_{i \in V(T)}$ which form a near partition of $V(G)$ (i.e. some bags may be empty, but nonempty bags form a partition of $V(G)$). For $t \in V(T)$, we denote by $e(t)$ the edge of T incident to t and its parent. For $e \in E(T)$, let T_1, T_2 denote the two connected components of $T - e$. We denote by $\text{cut}(e)$ the set of edges with an endpoint in both of $\bigcup_{i \in V(T_1)} X_i$ and $\bigcup_{i \in V(T_2)} X_i$. The adhesion of $t \in V(T)$ is $\text{adh}(t) = |\text{cut}(e(t))|$, and its torso-size is $\text{tor}(t) = |X_t| + b_t$ where b_t is the number of edges $e \in E(T)$ incident to t such that $|\text{cut}(e)| \geq 3$. The width of the decomposition is then $\max_{t \in V(T)} \{\text{adh}(t), \text{tor}(t)\}$. Note that edges are allowed to go between vertices that are not in the same bag. The tree-cut width of a graph is the minimal width of tree-cut decomposition. When $|\text{cut}(e)| \geq 3$, the edge e is called *bold*, and otherwise, e is called *thin*. When $\text{adh}(t) \leq 2$, node t is called *thin*, otherwise it is called *bold*. In [21], it is shown that a tree-cut decomposition can be assumed to be *nice*, meaning that if $t \in V(T)$ is thin then $N(Y_t) \cap \left(\bigcup_{b \text{ sibling of } t} Y_b \right) = \emptyset$, where Y_i is the union of X_j for j in the subtree of i .

Wollan shows that having bounded tree-cut width is equivalent to only having wall immersions of bounded size.

► **Observation 18.** $K_{3,n-3}$ has tree-partition-width at most 3 but unbounded tree-cut width.

Indeed, note that any graph on $n - 3$ vertices with maximum degree 3 can be immersed in $K_{3,n-3}$. In particular, this works for any wall. The lower bound given by Wollan shows that the tree-cut width of $K_{3,n-3}$ is $\Omega(n^{\frac{1}{4}})$.

We denote by $\underline{\text{tpw}}(G)$ the minimum tree-partition-width over subdivisions of G , and by $\overline{\text{tpw}}(G)$ the maximum tree-partition-width of subdivisions of G . We will show that both are polynomially tied to the tree-partition-width of G , which proves useful in polynomially bounding tree-partition-width by tree-cut width due to the following lemma.

► **Lemma 19.** $\underline{\text{tpw}} = O(\text{tcw}^2)$.

Proof. Consider a nice tree-cut decomposition (T, \mathcal{X}) of a graph G of width k . We will construct a tree-partition for a subdivision of G . Note that the bags are already disjoint, but some edges are not between neighbouring bags of T .

Each edge uv of G is subdivided $d_T(u, v)$ times, which is the distance between the nodes containing u and v respectively in their bag (recall that the bags form a near partition). We then add the vertices of the subdivided edge in the bags on the path in the decomposition between the bags containing their endpoints. This is sufficient to make the decomposition a tree-partition T' of a subdivision of G .

We now argue that T' has a width of $O(k^2)$. A bag Y_t of T' contains at most:

- k initial vertices
- $\max(2, k)$ vertices from subdivisions of edges in $\text{cut}(e(t))$ accounting for edges going from a child of t to an ancestor of t
- $k^2/2$ vertices from edges that are between bold children of t . For u, v children of T , there are only edges between T_u and T_v if both are bold. There are also at most k such edges incident to T_u for any child u of T , and we may divide by 2 since each edge will be counted twice this way. We stress that thin children do not contribute because the tree-cut decomposition is nice.

Hence, $\underline{\text{tpw}}(G) \leq 2 + k(k + 2)/2 + k = O(\text{tcw}(G)^2)$ ◀

Next, we consider the parameters tpw and $\overline{\text{tpw}}$.

► **Lemma 20.** $\text{tpw} \leq \overline{\text{tpw}} \leq \text{tpw}(\text{tpw} + 1)$

Proof. The lower bound is immediate. We prove the upper bound.

Consider a graph G with a tree-partition (T, \mathcal{X}) of width k , and a subdivision G' of G . We construct a tree-partition (T', \mathcal{X}') of G' of width at most $k(k + 1)$.

We root the decomposition T arbitrarily.

Suppose that u, v are in the same bag of T and the edge uv was subdivided to form the path u, a_1, \dots, a_ℓ, v . We add the vertices a_i in new bags containing, $\{a_1, a_\ell\}$, $\{a_2, a_{\ell-1}, \dots$ which corresponds to a new branch of the decomposition of width at most 2.

Consider next the vertices obtained by subdividing an edge uv for u in the child bag of the bag of v . If a subdivided edge was between two vertices of adjacent bags, we order the vertices of the path obtained by subdividing the edge from the vertex in the child bag to the vertex in the parent bag. We add the penultimate vertex to the child bag, and fold the remaining vertices of the path in a fresh branch of the decomposition of width at most 2 similarly to the previous case.

This gives a tree partition T' . Bags of T' that are not in T have size at most 2, and, to bags of T' that are also in T , we added at most k^2 vertices (at most one per edge between the bag and its parent). We conclude that T' has width at most $k(k+1)$. ◀

A result of Ding and Oporowski [15] shows that tree-partition-width is tied to a parameter γ that is (by design) monotonic with respect to the topological minor relation. We adapt their proof to derive the following stronger result.

► **Theorem 21.** *There exists a parameter γ which is polynomially tied to the tree-partition-width, and is monotonic with respect to the topological minor relation. More precisely, $\mathbf{tpw} = \Omega(\gamma)$ and $\mathbf{tpw} = O(\gamma^{24})$.*

We deduce the following statement.

► **Corollary 22.** *\mathbf{tpw} , $\overline{\mathbf{tpw}}$, and \mathbf{tpw} are polynomially tied.*

Proof. Lemma 20 shows that \mathbf{tpw} and $\overline{\mathbf{tpw}}$ are polynomially tied. Note that, by definition, $\overline{\mathbf{tpw}} \leq \mathbf{tpw}$. Then, for a fixed graph G , consider H a subdivision of G achieving $\mathbf{tpw}(H) = \overline{\mathbf{tpw}}(G)$. $\mathbf{tpw}(G)^{O(1)} \leq \gamma(G) \leq \gamma(H) \leq \mathbf{tpw}(H)^{O(1)} = \overline{\mathbf{tpw}}(G)^{O(1)}$. The first and last inequalities come from the fact that γ and \mathbf{tpw} are polynomially tied. The middle inequality is because γ is monotonic with respect to the topological minor relation. ◀

From Lemma 19 and Corollary 22, we deduce the following theorem.

► **Theorem 23.** *The parameter tree-partition-width is polynomially bounded by the parameter tree-cut width. In other words, we show that there exist constants $C, c > 0$ such that for any graph G , $\mathbf{tpw}(G) \leq C \mathbf{tcw}(G)^c$.*

We now turn our focus to the technical proof of Theorem 21. We define the m -grid as the graph on the vertex set $[m] \times [m]$ with edges $(i, j)(i', j')$ when $|i - i'| + |j - j'| \leq 1$. We then define the m -wall as the graph obtained from the m -grid by removing edges $(i, j)(i+1, j)$ for $i + j$ even. The *wall number* of a graph G is then defined as the largest m such that G contains the m -wall as a (topological)² minor, and the *grid number* of G is the largest m such that G contains the m -grid as a minor. We denote them by $\mathbf{wn}(G)$ and $\mathbf{gn}(G)$ respectively.

► **Observation 24.** *The wall number and the grid number are linearly tied: $\mathbf{wn}(G) = \Theta(\mathbf{gn}(G))$.*

We use the following result of Chuzhoy and Tan [12] (the bound is weakened to have a lighter formula).

► **Lemma 25** (Chuzhoy and Tan [12]). *The treewidth is polynomially tied to the grid number: $\mathbf{tw} = \Omega(\mathbf{gn})$ and $\mathbf{tw} = O(\mathbf{gn}^{10})$.*

Hence, the treewidth is polynomially tied to the wall number: $\mathbf{tw} = \Omega(\mathbf{wn})$ and $\mathbf{tw} = O(\mathbf{wn}^{10})$.

We call m -fan the graph that consists of a path of order m with an additional vertex adjacent to all of the vertices of the path. We call m -branching-fans the graphs that consist of a tree T and a vertex v adjacent to a subset N of the vertices of T containing at least the leaves, such that m is the minimum size of a subset of vertices X of T such that each component of $T - X$ contains at most m vertices of N . In particular, the $(m+1)^2$ -fan is an

² The notions of minor and topological minor coincide for graphs of maximum degree at most 3.

$(m + 1)$ -branching-fan. We call m -multiple of a tree of order m a graph obtained from a tree of order m after replacing its edges by m parallel edges and then subdividing each edge once to keep the graph simple.

Let $\gamma_1(G)$ be the largest m such that G contains an m -branching-fan as a topological minor. Let $\gamma_2(G)$ be the largest m such that G contains an m -multiple of a tree of order m as a topological minor.

Let $\gamma(G)$ be the maximum of $\mathbf{wn}(G)$, $\gamma_1(G)$, and $\gamma_2(G)$.

▷ **Claim 26.** The parameter γ is monotonic with respect to the topological minor relation.

Proof. Let G be a graph and H be a topological minor of G . Any topological minor of H is also a topological minor of G , hence $\mathbf{wn}(G) \geq \mathbf{wn}(H)$, $\gamma_1(G) \geq \gamma_1(H)$, $\gamma_2(G) \geq \gamma_2(H)$. We conclude that $\gamma(G) \geq \gamma(H)$. ◁

► **Observation 27.** *The m -branching-fans, the m -multiples of trees of order m and the m -wall have tree-partition-width $\Omega(m)$. Hence, we have $\mathbf{tpw}(G) = \Omega(\gamma(G))$.*

We fix a graph G and let $m = \gamma(G)$. Note that $m \geq \gamma_2(G) > 0$.

We denote by G^b the graph on the vertex set of G , where xy is an edge if and only if there are at least b vertex disjoint paths from x to y . We now consider G^b for $b = \Omega(m^{10})$.

▷ **Claim 28.** The connected components of G^b have size at most m .

Proof. We proceed by contradiction, and assume there is a connected component C of size at least $m + 1$.

Since C is connected, it contains a spanning tree T . We number its edges e_1, \dots, e_ℓ such that every prefix induces a connected subtree of T . We construct a subgraph H of G that should be an $(m + 1)$ -multiple of a tree of order $m + 1$, contradicting the definition of m . For each edge uv , in order, we try to add to H $m + 1$ vertex disjoint paths from u to v that avoid vertices of C and the vertices already in H . If we manage to do this for at most m edges, then we have placed at most $m(m + 1)$ paths. Let uv be the first edge for which we could not find $m + 1$ vertex disjoint paths that do not intersect previous vertices (except for u or v). By definition of G^b , there are b vertex disjoint u, v -paths in G , we denote the set of such paths by π . At most m of the paths of π hit vertices of C already in H . Then, since at least $b - m$ are hit by previous paths and there are at most $m(m + 1)$ previous paths. By the pigeon hole principle, one of the previous paths P_0 must hit $\frac{b-m}{m(m+1)} \geq (m + 1)^2$ paths in π . By considering P_0 and the paths it hits in π , we easily obtain a subdivision of an $(m + 1)^2$ -fan. This is a contradiction with the definition of m . Hence, we must have been able to process m edges. Which means we obtained a subdivision of an $(m + 1)$ -multiple of a tree of order $m + 1$. This is a contradiction to the definition of m . We conclude that the connected component must have size at most m . ◁

Let H be the quotient of G by the connected components of G^b . We call it the b -reduction of G .

▷ **Claim 29.** The blocks of H have maximum degree at most bm^3 .

Proof. Assume by contradiction that the maximum degree is more than bm^3 . Let B be a block of H , and X be one of its vertices of maximum degree. X contains at most m vertices of G by Claim 28. The vertices of G in $B - X$ must be in the same connected component C of G since $b > m$. There are at least $bm^3 + 1$ edges between X and C . By the pigeon hole principle, one vertex v of X must have at least $bm^2 + 1$ neighbours in C . Consider a

7:20 On the Parameterized Complexity of Computing Tree-Partitions

spanning tree T of C . We iteratively remove leaves that are not neighbours of v , and then replace any vertex of degree 2 that is not a neighbour of v by an edge between its neighbours. We denote this reduced tree by T' .

First, note that the degree in T' is bounded by $b - 1$ because incident edges can be extended to vertex disjoint paths to leaves of T' which are neighbours of v by construction. We now use the fact that G contains no $(m + 1)$ -branching-fans as topological minors. In particular, there must be a set U of vertices of T' of size at most m such that components of $T' - U$ contain at most m neighbours of v . By removing at most m vertices of degree at most $(b - 1)$, we have at most $1 + (b - 2)m$ components in $T' - U$ meaning v has degree bounded by $(b - 1)m^2$. We found our contradiction. \triangleleft

\triangleright **Claim 30.** The treewidth of H is at most $O(b)$.

Proof. We first apply Lemma 25 to bound the treewidth of G by $O(b)$. Consider a tree decomposition of G of adequate width $\Theta(b)$, and replace each bag by the components of G^b that intersect it. By Claim 4, this is a decomposition of H . \triangleleft

Using Claim 29 and Claim 30 and the construction of Wood as we did in the approximation algorithm, we obtain a tree-partition of H of width $O(b^2m^3)$. We then replace components of G^b by their vertices, obtaining a tree-partition of G of width $O(b^2m^4)$ due to Claim 28.

We have obtained a tree-partition of width $O(b^2m^4) = O(m^{24})$. This concludes the proof of Theorem 21.

XNLP-Completeness for Parameterized Problems on Graphs with a Linear Structure

Hans L. Bodlaender  

Utrecht University, The Netherlands

Carla Groenland  

Utrecht University, The Netherlands

Hugo Jacob  

ENS Paris-Saclay, France

Lars Jaffke  

University of Bergen, Norway

Paloma T. Lima  

IT University of Copenhagen, Denmark

Abstract

In this paper, we showcase the class XNLP as a natural place for many hard problems parameterized by linear width measures. This strengthens existing $W[1]$ -hardness proofs for these problems, since XNLP-hardness implies $W[t]$ -hardness for all t . It also indicates, via a conjecture by Pilipczuk and Wrochna [ToCT 2018], that any XP algorithm for such problems is likely to require XP space.

In particular, we show XNLP-completeness for natural problems parameterized by pathwidth, linear clique-width, and linear mim-width. The problems we consider are INDEPENDENT SET, DOMINATING SET, ODD CYCLE TRANSVERSAL, $(q-)$ COLORING, MAX CUT, MAXIMUM REGULAR INDUCED SUBGRAPH, FEEDBACK VERTEX SET, CAPACITATED (RED-BLUE) DOMINATING SET, and BIPARTITE BANDWIDTH.

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

Keywords and phrases parameterized complexity, XNLP, linear clique-width, W-hierarchy, pathwidth, linear mim-width, bandwidth

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.8

Related Version *Full Version*: <https://arxiv.org/abs/2201.13119>

Funding *Carla Groenland*: Supported by the European Union's Horizon 2020 research and innovation programme under the ERC grant CRACKNP (number 853234) and the Marie Skłodowska-Curie grant GRAPHCOSY (number 101063180).

Lars Jaffke: Supported by the Norwegian Research Council (project number 274526) and the Meltzer Research Fund.

1 Introduction

Since the inception of parameterized complexity in the late 1980s and early 1990s, much research has been done on establishing the complexity of parameterized problems. Typically one is particularly interested in either designing FPT-algorithms for these problems, or to prove them $W[t]$ -hard, for some t , which provides evidence that such a problem is not likely to be fixed-parameter tractable. As opposed to the classical P versus NP-complete setting, the question of membership in some class of the W -hierarchy is often much less clear. While some natural problems such as INDEPENDENT SET and DOMINATING SET are known to be $W[1]$ -complete and $W[2]$ -complete, respectively, many other problems are unknown to



© Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Lars Jaffke, and Paloma T. Lima; licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 8; pp. 8:1–8:18

Leibniz International Proceedings in Informatics



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

be complete for a class of parameterized problems, and even conjectured not to be in the W -hierarchy. Recently, building upon work by Elberfeld et al. [12], Bodlaender et al. [4] introduced a complexity class called XNLP, which gives a way of addressing this question.

The class XNLP consists of the parameterized problems that can be solved with a non-deterministic algorithm that uses $f(k) \log n$ space and $f(k)n^c$ time, where f is a computable function, n is the input size, k is the parameter and c is a constant. In particular, XNLP-hardness implies $W[t]$ -hardness for all t . Therefore it is unlikely that any XNLP-hard problem is complete for some $W[t]$.

One success story within parameterized algorithms and complexity is the use of width measures of graphs as parameters (see, e.g., [9]). Typically, such width measures are defined in terms of a tree-like decomposition of a graph, and the width describes the complexity of the decomposition, and therefore, in turn, of the graph. Such width measures also have linear variants, where the decomposition resembles a path instead of a tree. In this work, we provide evidence that the class XNLP is the “natural home” for hard problems parameterized by linear width measures.

Let us give some intuitive explanation why this is the case. A typical dynamic programming algorithm that uses such a linear decomposition stores, at each node of the path, some partial solutions associated with it. The table entries associated with the nodes are then filled in the order in which they appear on the path. If one turns such an algorithm into a nondeterministic algorithm, it often suffices at the i -th node to nondeterministically determine the table index corresponding to the correct partial solution (if it exists) from the table entry that was previously determined for the $(i - 1)$ -th node. In such a case, membership in XNLP follows if each single table entry of such a DP algorithm can be represented by $f(k) \log n$ bits (where k is the width) and if the nondeterministic step does not require a computation that uses significantly more space. This is often the case. Now, such an approach fails for tree-like decompositions, since even a nondeterministic algorithm might have to keep too many table entries at some point during the computation. One common situation in which this occurs is when the algorithm needs to store one table entry for each level of the decomposition. This incurs a multiplicative factor in the memory usage that depends on the height of the tree, which can be prohibitively large.

In this direction, Bodlaender et al. [4] showed that LIST COLOURING parameterized by the pathwidth of the input graph, and BANDWIDTH are XNLP-complete. In this paper, we show XNLP-completeness of fundamental graph problems parameterized by linear variants of well-established width measures, such as pathwidth, linear clique-width and linear mim-width, as well as some of their logarithmic analogues.

Besides showing $W[t]$ -hardness for all t , XNLP-hardness also provides insight into the space complexity of parameterized problems. Pilipczuk and Wrochna [23] proposed the following conjecture.¹

► **Conjecture 1** (Slice-wise Polynomial Space Conjecture [23]). *XNLP-hard problems do not have an algorithm, that runs in $n^{f(k)}$ time and $f(k)n^c$ space, with f a computable function, k the parameter, n the input size, and c a constant.*

Typically, membership in XP for the problems studied in our paper follows from a dynamic programming approach that uses a significant amount of memory. XNLP-hardness indicates (via Conjecture 1) that dynamic programming is in some sense “optimal” (no XP algorithm can use “significantly less” memory).

¹ The statement of the conjecture here is equivalent to the conjecture on time and memory use for the LONGEST COMMON SUBSEQUENCE problem from [23]; the name of the conjecture is taken as analogue to the naming of XP as problems that use *slice-wise polynomial time* (see [9, Section 1.1]).

Linear width measures and logarithmic analogues. The width measures we consider in this work include linear variants of arguably the most prominent measures, and some of their generalizations. Pathwidth is a linear variant of the classic treewidth parameter, which, informally speaking, measures how close a connected graph is to being a tree. In this vein, pathwidth measures how close a connected graph is to being a path. Clique-width (or, equivalently, rank-width) generalizes treewidth to several simply structured dense graphs, and its linear counterpart is called linear clique-width (linear rank-width). Maximum induced matching width [24], or mim-width for short, in turn generalizes clique-width and remains bounded even on well-studied graph classes such as interval and permutation graphs, where the clique-width is known to be unbounded. In fact, for most of these classes the *linear* mim-width is bounded.

We also introduce a new parameter that we call logarithmic linear clique-width, analogous to the parameter logarithmic pathwidth that was introduced by Bodlaender et al. [4]. For an n -vertex graph of linear clique-width k , logarithmic linear clique-width takes the value $\lceil k/\log n \rceil$. We stress the fact that XNLP-hardness parameterized by a logarithmic parameter implies that there is no algorithm solving the problem in time $2^{O(k)}n^{O(1)}$ and space $k^{O(1)}n^{O(1)}$, where k is the original parameter², under Conjecture 1. Such results can complement existing (S)ETH lower bounds for single exponential FPT algorithms with lower bounds on the space requirements of such algorithms.

Bipartite bandwidth. Finally, we consider a bipartite variant of the notoriously difficult [2] problem of computing the bandwidth of a graph. Here, for a bipartite graph with vertex bipartition (A, B) and bandwidth target value w , we seek an ordering α of A and an ordering β of B , such that for each edge ab , $|\alpha(a) - \beta(b)| \leq w$. We consider this problem parameterized by w , and show that it is XNLP-complete, even when the input graph is a tree.

Our results. We summarize our results in the following theorem.

► **Theorem 2.** *The following problems are XNLP-complete.*

- (i) CAPACITATED RED-BLUE DOMINATING SET and CAPACITATED DOMINATING SET parameterized by pathwidth.
- (ii) COLORING, MAXIMUM REGULAR INDUCED SUBGRAPH, and MAX CUT parameterized by linear clique-width.
- (iii) q -COLORING and ODD CYCLE TRANSVERSAL parameterized by logarithmic pathwidth or logarithmic linear clique-width.
- (iv) INDEPENDENT SET, DOMINATING SET, FEEDBACK VERTEX SET, and q -COLORING for fixed $q \geq 5$ parameterized by linear mim-width.
- (v) BIPARTITE BANDWIDTH, even if the input graph is a tree.

Furthermore, FEEDBACK VERTEX SET parameterized by logarithmic pathwidth or logarithmic linear clique-width is XNLP-hard.

Note that Theorem 2(ii) and (iv) include the first XNLP-completeness results for graph problems with the linear clique-width and linear mim-width as parameter.

² Indeed, replacing k with $k' \log n$, this gives running time $2^{O(k' \log n)} = n^{O(k')}$ and space $k'^{O(1)}n^{O(1)}$, which is excluded by the conjecture.

Related Work. Guillemot [17] introduced the class WNL (which equals XNLP closed under fpt-reductions), and showed some problems to be complete for WNL, including a version of LONGEST COMMON SUBSEQUENCE. The class XNLP (under a different name) was introduced by Elberfeld et al. [12], who also showed a number of problems, including LINEAR CELLULAR AUTOMATON ACCEPTANCE, to be complete for the class. A large number of parameterized problems was shown to be XNLP-complete recently by Bodlaender et al. [4]. Very recently, in work that aims at separating the complexity of treewidth and pathwidth at one side, and stable gonality at another side, Bodlaender et al. [3] showed a number of flow problems parameterized by pathwidth to be complete for XNLP.

2 Overview of the results

In this section, we give a bird’s-eye view of the results proved in this paper, and discuss related work for the specific problems we consider. Due to space limitations, statements marked with ♣ had their proofs deferred to the full version of this work.

Parameterized by linear clique-width. We consider the MAX CUT, the COLORING, and the MAXIMUM REGULAR INDUCED SUBGRAPH problems parameterized by linear clique-width. Let $E(V_1, V_2)$ denote the set of edges with one endpoint in V_1 and one endpoint in V_2 .

MAX CUT

Input: A graph $G = (V, E)$ described by a given linear k -expression describing G and an integer W .

Parameter: k .

Question: Is there a bipartition of V into (V_1, V_2) such that $|E(V_1, V_2)| \geq W$?

In 1994, Wanke [25] showed that MAX CUT is in XP for graphs of bounded NLC-width, which directly implies XP-membership with clique-width as parameter, as NLC-width and clique-width are linearly related. In 2014, Fomin et al. [14] consider the fine grained complexity for MAX CUT for graphs of small clique-width, giving an algorithm with improved running time and showing asymptotic optimality (assuming the Exponential Time Hypothesis). From their results, it follows that MAX CUT is $W[1]$ -hard with clique-width as parameter. In Section 4.1, we prove the following theorem.

► **Theorem 3.** MAX CUT with linear clique-width as parameter is XNLP-complete.

Next, we consider the classical COLORING problem, which given a graph G and an integer k asks if G has a proper coloring with k colors. Similarly to the story of the MAX CUT problem, COLORING parameterized by clique-width was shown to be in XP by Wanke in 1994 [25], and a $W[1]$ -hardness proof only followed in 2010 by Fomin et al. [13]. The XP algorithm for coloring runs in time $n^{O(2^k)}$, where k is the clique-width, and Fomin et al. [15] even showed that this run time can probably not be substantially improved: an algorithm running in time $n^{2^{o(k)}}$ would refute the ETH. We prove the following.

► **Theorem 4 (♣).** COLORING parameterized by linear clique-width is XNLP-complete.

Lastly, we consider the MAXIMUM REGULAR INDUCED SUBGRAPH problem. The problem was studied by several authors, including Asahiro et al. [1], who show among others an algorithm that uses linear time for graphs of bounded treewidth, where the time depends single exponentially on the treewidth. Moser and Thilikos [22], and independently Mathieson and Szeider [21] show (amongst other results) that the problem is $W[1]$ -hard when the size

of the subgraph (parameter W in our description below) is used as parameter. Broersma et al. [7] give XP algorithms for several problems, including MAXIMUM REGULAR INDUCED SUBGRAPH for graphs of bounded clique-width.

MAXIMUM REGULAR INDUCED SUBGRAPH

Input: A graph G described by a given linear k -expression and two integers W and D .

Parameter: k .

Question: Is there a D -regular induced subgraph of G on at least W vertices?

We show the following.

► **Theorem 5 (♣).** *MAXIMUM REGULAR INDUCED SUBGRAPH parameterized by linear clique-width is XNLP-complete.*

Parameterized by pathwidth. We consider the CAPACITATED RED-BLUE DOMINATING SET and CAPACITATED DOMINATING SET problems. Below, we give the formal statement of the problems, where we have the width of the path decomposition as parameter. One of the reasons of interest in these problems is that they model facility location problems: the red vertices model possible facilities that can serve a bounded number of clients which are modelled by the blue vertices.

CAPACITATED RED-BLUE DOMINATING SET

Input: A bipartite graph $G = (R, B, E)$, a path decomposition of G of width ℓ , a capacity function $c : R \rightarrow \mathbb{N}$, and an integer k .

Parameter: ℓ .

Question: Is there a subset S of R , and an assignment of blue vertices $f : B \rightarrow S$ such that $\{w, f(w)\} \in E$ for all $w \in R$ and $|f^{-1}(v)| \leq c(v)$ for all v in S ?

CAPACITATED DOMINATING SET

Input: A graph $G = (V, E)$, a path decomposition of G of width ℓ , a capacity function $c : V \rightarrow \mathbb{N}$, and an integer k .

Parameter: ℓ .

Question: Is there a subset S of V , and an assignment of the vertices $f : V \rightarrow S$ such that $\{w, f(w)\} \in E$ or $w = f(w)$ for all $w \in V$ and $|f^{-1}(v)| \leq c(v)$ for all v in S ?

In 2008, Dom et al. [10] showed that CAPACITATED DOMINATING SET is $W[1]$ -hard, with the treewidth and solution size k as combined parameter. CAPACITATED DOMINATING SET was shown to be $W[1]$ -hard for planar graphs, with the solution size as parameter by Bodlaender et al. [5]. Fomin et al. [14] give bounds for the fine grained complexity of CAPACITATED RED-BLUE DOMINATING SET, for graphs with a small feedback vertex set; their results imply that the problem is $W[1]$ -hard with feedback vertex set as parameter. The proof of the following theorem can be found in Section 4.2.

► **Theorem 6.** *CAPACITATED RED-BLUE DOMINATING SET and CAPACITATED DOMINATING SET parameterized by pathwidth are XNLP-complete.*

Parameterized by logarithmic linear clique-width. Bodlaender et al. [4] introduced the parameter *logarithmic pathwidth* as $\mathbf{pw} / \log n$ for an n -vertex graph of pathwidth \mathbf{pw} . This allows the pathwidth to be linear in the logarithm of the number of vertices of the graph. Here we introduce the *logarithmic linear clique-width* as $\mathbf{lcw} / \log n$ for graphs on n vertices with linear clique-width \mathbf{lcw} .

We provide new XNLP-complete problems for the parameter logarithmic pathwidth, and show that these problems and the previously known XNLP-complete problems for this parameter [4] are also complete for the parameter logarithmic linear clique-width. Our results are summarised below.

The motivation to study the logarithmic linear clique-width or logarithmic pathwidth comes from the observation that many FPT algorithms with linear cliquewidth or pathwidth as parameter have a single exponential time dependency on the parameter. Thus, if linear cliquewidth or pathwidth is logarithmic in the size of the graph, these algorithms turn into XP algorithms.

► **Theorem 7 (♣).** *When parameterized by logarithmic pathwidth or logarithmic linear clique-width, INDEPENDENT SET, DOMINATING SET, q -LIST-COLORING for $q > 2$, and ODD CYCLE TRANSVERSAL are XNLP-complete, and FEEDBACK VERTEX SET is XNLP-hard.*

Lokshtanov et al. [20] established (tight) lower bounds for these problems for the parameter pathwidth under the Strong Exponential Time Hypothesis. Several of our gadgets are based on those used for these lower bounds by [20].

Parameterized by linear mim-width. We prove that several fundamental graph problems are XNLP-complete when parameterized by the mim-width of a given linear order of the input graph. $W[1]$ -hardness for INDEPENDENT SET and DOMINATING SET in this parameterization was shown by Fomin et al. [16], and for FEEDBACK VERTEX SET by Jaffke et al. [19]. For q -COLORING, $W[1]$ -hardness was not known before our work. We would like to point out that our XNLP-hardness proof uses a gadget that requires five colors to construct, and it would be interesting to see if this can be improved to three colors. In section Section 4.3 we prove the result below for q -COLORING. The proofs for the remaining problems are deferred to the full version.

► **Theorem 8 (♣).** *When parameterized by linear mim-width, INDEPENDENT SET, DOMINATING SET, q -COLORING for any fixed $q \geq 5$ and FEEDBACK VERTEX SET are XNLP-complete.*

Bipartite bandwidth. We consider the following bipartite variant of the BANDWIDTH problem.

BIPARTITE BANDWIDTH

Input: A bipartite graph $G = (X, Y, E)$ and an integer k .

Parameter: k .

Question: Are there orderings $\alpha : X \rightarrow [n]$ and $\beta : Y \rightarrow [m]$ such that for each $uv \in E$, $|\alpha(u) - \beta(v)| \leq k$?

A possible application of this problem is as follows. Let a matrix M be given. Create a vertex $x_i \in X$ for each row i and a vertex $y_j \in Y$ for each column j , and let x_i be adjacent to y_j if and only if $M_{i,j} \neq 0$. This graph has bipartite bandwidth at most k if and only if the rows and columns of M can be permuted (individually) in such a way that all non-zero entries are within k distance from the main diagonal. We show the following.

► **Theorem 9 (♣).** *BIPARTITE BANDWIDTH is XNLP-complete for trees.*

3 Preliminaries

The required background on the computational problems studied in this paper are given in their respective sections. The notions relevant to the entire paper are defined below.

We write $[n] = \{1, \dots, n\}$ and $[a, b]$ for the set of integers x with $a \leq x \leq b$. All logarithms in this paper have base 2. We use \mathbb{N} for the set of the natural numbers $\{0, 1, 2, \dots\}$, and \mathbb{Z}^+ denotes the set of the positive natural numbers $\{1, 2, \dots\}$. We write $N(S)$ and $N[S] = N(S) \cup S$ for the open and closed neighborhood of S .

3.1 Definition of the class XNLP

In this paper, we study parameterized decision problems, which are subsets of $\Sigma^* \times \mathbb{N}$, for a finite alphabet Σ . We assume the reader to be familiar with notions from parameterized complexity, such as XP, $W[1]$, $W[2]$, \dots , $W[P]$ (see e.g. [11]). The class XNLP (denoted $N[f \text{ poly}, f \log]$ by [12]) consists of the parameterized decision problems that can be solved by a non-deterministic algorithm that simultaneously uses at most $f(k)n^c$ time and at most $f(k) \log n$ space, on an input (x, k) , where x can be denoted with n bits, f a computable function, and c a constant. We assume that functions f of the parameter in time and resource bounds are computable – this is called *strongly uniform* by Downey and Fellows [11]. More information about the complexity class XNLP can be found in [4].

3.2 Reductions

In the remainder of the paper, unless stated otherwise, completeness for XNLP is with respect to pl-reductions, which are defined below. The definitions are based upon the formulations in [12].

- A *parameterized reduction* from a parameterized problem $Q_1 \subseteq \Sigma_1^* \times \mathbb{N}$ to a parameterized problem $Q_2 \subseteq \Sigma_2^* \times \mathbb{N}$ is a function $f : \Sigma_1^* \times \mathbb{N} \rightarrow \Sigma_2^* \times \mathbb{N}$, such that the following holds.
 1. For all $(x, k) \in \Sigma_1^* \times \mathbb{N}$, $(x, k) \in Q_1$ if and only if $f((x, k)) \in Q_2$.
 2. There is a computable function g , such that for all $(x, k) \in \Sigma_1^* \times \mathbb{N}$, if $f((x, k)) = (y, k')$, then $k' \leq g(k)$.
- A *parameterized logspace reduction* or *pl-reduction* is a parameterized reduction for which there is an algorithm that computes $f((x, k))$ in space $O(g(k) + \log n)$, with g a computable function and $n = |x|$ the number of bits to denote x .

3.3 Pathwidth, linear clique-width, and linear mim-width

A *path decomposition* of a graph $G = (V, E)$ is a sequence (X_1, X_2, \dots, X_r) of subsets of V with the following properties.

1. $\bigcup_{1 \leq i \leq r} X_i = V$.
2. For all $\{v, w\} \in E$, there is an $i \in I$ with $v, w \in X_i$.
3. For all $1 \leq i_0 < i_1 < i_2 \leq r$, $X_{i_0} \cap X_{i_2} \subseteq X_{i_1}$.

The *width* of a path decomposition (X_1, X_2, \dots, X_r) equals $\max_{1 \leq i \leq r} |X_i| - 1$, and the *pathwidth* pw of a graph G is the minimum width of a path decomposition of G .

A *k-labeled* graph is a graph $G = (V, E)$ together with a labeling function $\Gamma : V \rightarrow [k]$. A *k-expression* constructs a *k-labeled* graph by the means of the following operations:

1. *Vertex creation*: $i(v)$ is the *k-labeled* graph consisting of a single vertex v which is assigned label i .
2. *Disjoint union*: $H \oplus G$ is the disjoint union of *k-labeled* graphs H and G .

3. *Join*: $\eta_{i \times j}(G)$ is the k -labeled graph obtained by adding all possible edges between vertices with label i and vertices with label j to G .
4. *Renaming label*: $\rho_{i \rightarrow j}(G)$ is the k -labeled graph obtained by assigning label j to all vertices labelled i in G .

A *linear k -expression* is a k -expression with the additional condition that one of the arguments of the disjoint union operation needs to be a graph consisting of a single vertex. The *clique-width* $\mathbf{cw}(G)$ (resp. *linear clique-width* $\mathbf{lcw}(G)$) of a graph G is the minimal k such that G can be constructed by a k -expression (resp. linear k -expression) with any labeling.

For a graph $G = (V, E)$ and $A, B \subseteq V$ with $A \cap B = \emptyset$, we let $G[A, B]$ be the bipartite subgraph of G with vertices $A \cup B$ and edges $\{ab \mid ab \in E, a \in A, b \in B\}$. We let $\text{cutmim}_G(A, B)$ be the size of a maximum induced matching in $G[A, B]$ and $\text{mim}_G(A) = \text{cutmim}_G(A, V \setminus A)$. Here, an induced matching $M \subseteq E$ is a matching such that there are no additional edges between the endpoints of M in the graph in question. The *mim-width* of a linear order v_1, \dots, v_n of V is the maximum, over all i , of $\text{mim}_G(\{v_1, \dots, v_i\})$. The *linear mim-width* of G is the minimum mim-width over all linear orders of V .

3.4 Chained variants of Satisfiability and Multicolored Clique

In [4], the following problems were introduced, and shown to be XNLP-complete.

CHAINED POSITIVE CNF-SAT

Input: r sets of Boolean variables X_1, X_2, \dots, X_r , each of size q ; an integer $k \in \mathbb{N}$; Boolean formula ϕ , which is in conjunctive normal form and an expression on $2q$ variables, using only positive literals; for each i , a partition of X_i into $X_{i,1}, \dots, X_{i,k}$ such that $\forall j, j' \in [k], |X_{i,j}| = |X_{i,j'}|$.

Parameter: k .

Question: Is it possible to satisfy the formula

$$\bigwedge_{1 \leq i \leq r-1} \phi(X_i, X_{i+1})$$

by setting from each set $X_{i,j}$ exactly 1 variable to true and all others to false?

CHAINED MULTICOLORED CLIQUE

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

Parameter: k .

Question: Is there a set $W \subseteq V$ 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 $f(v) = j$?

The CHAINED MULTICOLORED INDEPENDENT SET problem is defined analogously, with the only difference that the solution W is required to be an independent set.

► **Theorem 10** (Bodlaender et al. [4]). *CHAINED POSITIVE CNF-SAT, CHAINED MULTICOLORED CLIQUE and CHAINED MULTICOLORED INDEPENDENT SET are XNLP-complete.*

4 Problems parameterized by linear width measures

In this section we prove XNLP-completeness for three of the problems mentioned in Section 2 parameterized by linear width measures. The full version of this work contains all the proofs of the results stated in Section 2.

4.1 Max Cut parameterized by linear clique-width

In this section, we consider the MAX CUT problem, with the linear clique-width as parameter, and show it to be XNLP-complete. Our result is based upon the XNLP-hardness result for a problem, called CIRCULATING ORIENTATION, with pathwidth as parameter. Borrowing from terminology from flows in graphs, we say that a directed graph $G = (V, A)$ with for each arc $a \in A$ a weight $w(a) \in \mathbb{N}$, is a *circulation*, if for each vertex v , the total weight of all incoming arcs at v equals the total weight of all arcs outgoing from v . We reduce from the following problem.

CIRCULATING ORIENTATION

Input: An undirected graph $G = (V, E)$ with a path decomposition of G of width ℓ , an edge weight function $w : E \rightarrow \mathbb{N}$, given in unary notation.

Parameter: ℓ .

Question: Is there an orientation of G that is a circulation?

► **Theorem 11** (Bodlaender et al. [3]). CIRCULATING ORIENTATION is XNLP-complete.

► **Theorem 3.** MAX CUT with linear clique-width as parameter is XNLP-complete.

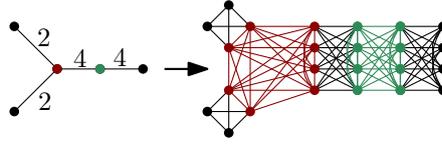
Proof. We first show membership in XNLP. The main idea is to turn the existing dynamic programming that solves the problem given a k -expression of an n -vertex graph of linear clique-width k into a non-deterministic algorithm, by guessing an element from a table instead of building full tables. For each vertex creation, we guess on which side of the partition the vertex is. We maintain the following certificate: for each label, the number of vertices on each side of the bipartition, and the number of edges of the current expression that were in the cut. Since there are at most k labels and the size of the cut is bounded by the number of edges, this certificate uses only $O(k \log n)$ bits.

To show hardness for XNLP, we reduce from CIRCULATING ORIENTATION with pathwidth as parameter. Suppose we have an instance for CIRCULATING ORIENTATION: an undirected graph G with edge weight function w . For each vertex v , write $D(v)$ as the total weight of all edges incident to v .

We build a new, undirected graph $H = (V_H, E_H)$ as follows. For each vertex $v \in V$, each edge e with v as one of its endpoints, and each integer $i \in [1, w(e)]$, we create a vertex $x_{v,e,i}$. Two distinct vertices $x_{v,e,i}$ and $x_{w,e',j}$ are adjacent if and only if $v = w$ or $e = e'$. In other words: for each vertex $v \in V$, we have a clique with $D(v)$ vertices, which consists of all vertices of the form $x_{v,\cdot,\cdot}$, that we call *the clique of v* . For each edge $e = \{v, w\} \in E$, we have a clique with $2w(e)$ vertices, namely all vertices of the form $x_{v,e,\cdot}$ and $x_{w,e,\cdot}$. See Figure 1 for a partial example.

▷ **Claim 12.** G has a circulating orientation if and only if H has a bipartition that cuts $\sum_{e \in E} w(e)^2 + \sum_{v \in V} D(v)^2/4$ edges.

Proof. Suppose G has a circulating orientation. For each edge $e = \{v, w\}$, if the orientation directs v to w , then add all vertices of the form $x_{v,e,i}$ to Z_1 and all vertices of the form $x_{w,e,i}$ to Z_2 ($i \in [1, w(e)]$); otherwise, add all vertices of the form $x_{v,e,i}$ to Z_2 and all vertices of the form $x_{w,e,i}$ to Z_1 ($i \in [1, w(e)]$).



■ **Figure 1** Example for the construction of the hardness proof of MAX CUT (fragment).

Since we started from a circulating orientation, for each vertex v there are $D(v)/2 \times D(v)/2$ edges of the form $\{x_{v,\cdot,\cdot}, x_{v,\cdot,\cdot}\}$ crossing the bipartition. Moreover, there are $w(e) \times w(e)$ edges of the form $\{x_{v,e,\cdot}, x_{w,e,\cdot}\}$ crossing the bipartition for each edge $e = \{v, w\}$. We conclude that the bipartition cuts the required number of edges.

Now, suppose we have a partition Z_1, Z_2 of V_H with $\sum_{e \in E} w(e)^2 + \sum_{v \in V} D(v)^2/4$ edges between Z_1 and Z_2 . We distinguish two types of edges in $E_H \cap (Z_1 \times Z_2)$. A Type 1 edge is an edge between two vertices $x_{v,e,i}$ and $x_{v,e',j}$ (i.e., it is in the clique of a vertex v). A Type 2 edge is an edge between two vertices $x_{v,e,i}$ and $x_{w,e,j}$ for some edge $e = \{v, w\}$. Note that each edge in H is of Type 1 or Type 2 and that H has precisely $\sum_{e \in E} w(e)^2$ Type 2 edges.

For each vertex $v \in V$, we consider how many Type 1 edges (those in the clique of v) are in $Z_1 \times Z_2$. If we have α vertices in the clique of v that belong to Z_1 , then $D(v) - \alpha$ vertices in the clique of v belong to Z_2 , and thus, in this clique, we cut $\alpha \cdot (D(v) - \alpha) \leq D(v)^2/4$ edges; the maximum possible is reached when $\alpha = D(v)/2$.

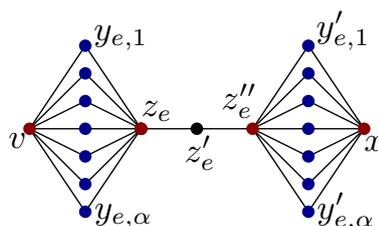
It follows that the number of Type 1 edges that are cut is at most $\sum_{v \in V} D(v)^2/4$. So, we must cut all Type 2 edges, i.e., for each edge $e = \{v, w\}$, all edges of the form $\{x_{v,e,i}, x_{w,e,j}\}$ are between a vertex in Z_1 and a vertex in Z_2 . It follows that we either have that all vertices of the form $x_{v,e,i}$ are in Z_1 and all vertices of the form $x_{w,e,i}$ are in Z_2 – in which case we direct the edge e from v to w ; or all vertices of the form $x_{v,e,i}$ are in Z_2 and all vertices of the form $x_{w,e,i}$ are in Z_1 , and now we direct the edge from w to v .

For each vertex $v \in V$, we must have exactly $D(v)/2$ vertices from the clique of v in Z_1 and equally many vertices in Z_2 ; otherwise, we cannot reach the required number of cut edges. Now, the total weight of all edges that we directed out of v precisely equals the number of vertices in the clique of v in Z_1 , and similarly, the total weight of all edges that we directed towards of v precisely equals the number of vertices in the clique of v in Z_2 . Both numbers equal $D(v)/2$. As this holds for each $v \in V$, the orientation defined above is a circulation. \triangleleft

Finally, we show that we can construct a linear clique expression for H given a path decomposition of G ; the number of colors we use for the clique width construction equals the width of the path decomposition plus 4. The construction uses ideas for constructing clique width constructions for line graphs of graphs of bounded treewidth; see [18].

Suppose we have a nice path decomposition, which uses introduce vertex, introduce edge, and forget nodes. We use $k + 1$ active colors – each active color will correspond to one vertex in the current bag. We also have an inactive color, which we will denote by the letter o . We also use two temporary colors, which we call α and β .

We sequentially visit the bags of the path decomposition. Bags correspond to a number of steps of the construction of H , as described next. If we introduce a vertex, we select a currently unused active color, and say this is the color of that vertex, and assume it to be used. If we introduce an edge $e = \{v, w\}$, we add the vertices $x_{v,e,i}$ one by one, each with the color α . Then, we add the vertices $x_{w,e,i}$ one by one, each with the color β . Now, we add all edges between vertices of color α and β . Now, recolor all vertices of color α by the color



■ **Figure 2** Edge gadget from the proof of Theorem 6.

of v . Then, recolor the vertices of color β by the color of w . If we forget a vertex v , we first add edges between all vertices of the color of v – at this point, these are all vertices in the clique of v , thus effectively ensuring that this set of vertices indeed is a clique. Then, recolor the vertices with the color of v with the inactive color o . Consider the color of v now unused.

One can verify that this indeed constructs precisely H , and that the construction can be done with $f(k) \log n$ additional space. ◀

4.2 Variants of Dominating Set parameterized by pathwidth

In this section we prove the following theorem.

► **Theorem 6.** CAPACITATED RED-BLUE DOMINATING SET and CAPACITATED DOMINATING SET parameterized by pathwidth are XNLP-complete.

Proof. We first show membership in XNLP for CAPACITATED RED-BLUE DOMINATING SET. For each red vertex, we guess if it is in the dominating set, and for each edge from a chosen red vertex to a blue neighbor, we guess if it is used for dominating. We do this while going through the path decomposition from left to right. We need to keep track which blue vertices are already dominated, which red vertices are in the dominating set plus their remaining capacity, and the total number of vertices in the dominating set so far. We may assume that the remaining capacities are never larger than the number of blue vertices; therefore, we only need to store $O(\log n)$ bits per vertex in the current bag. Membership in XNLP follows in a similar way for CAPACITATED DOMINATING SET.

Hardness follows by a reduction from CIRCULATING ORIENTATION (defined in Section 4.1). Suppose that we are given an input of CIRCULATING ORIENTATION, say a graph $G = (V, E)$ with weight function $w : E \rightarrow \mathbb{N}$. We assume that these weights are given in unary. Note that in a solution, the total weight of edges directed towards a vertex v and the total weight of the edges directed out of v should both equal $\sum_{\{v,x\} \in E} w(\{v,x\})/2$.

We build a graph as follows. For each vertex $v \in V$, we create a vertex v , colored red, in H . We give v a private blue neighbor v' . The capacity of v equals $1 + \sum_{\{v,x\} \in E} w(\{v,x\})/2$. We can assume this capacity is integral, otherwise there is no solution to the instance (G, w) . Each edge $e = \{v, x\} \in E$ is replaced by the following gadget. Suppose $w(\{v, x\}) = \alpha \in \mathbb{N}$. We create $2\alpha + 3$ vertices, called $y_{e,1}, y_{e,2}, \dots, y_{e,\alpha}, z_e, z'_e, z''_e, y'_{e,1}, \dots, y'_{e,\alpha}$. The edge e is replaced by the subgraph shown in Figure 2. The vertices z_e and z''_e are red, and all other new vertices are blue. We give the new red vertices z_e and z''_e a capacity that equals their degree.

Let $H = (V_H, E_H)$ be the resulting red-blue colored graph, with $c(v)$ the capacity of a red vertex $v \in V_H$. We claim that H has a dominating set of size $|V| + |E|$ for which each chosen red vertex dominates at most its capacity many blue vertices, if and only if G has a circulating orientation.

Suppose first that we have a set S of red vertices with $|S| \leq |V| + |E|$, and an assignment of blue vertices to neighbors in S , such that no red vertex has more than its capacity number of vertices assigned to it.

Each vertex that is a copy of a vertex from V must belong to S , as they have a private blue neighbor. For each edge e , either z_e or z_e'' must be in S , to dominate z_e' . This gives in total already $|V| + |E|$ vertices, so no edge can have both z_e and z_e'' in S . For each edge $e = \{v, x\}$, if $z_e \in S$, then orient the edge from v to x in G ; if $z_e'' \in S$, then orient the edge from x to v . Now, for each $v \in V$, the total weight of incoming edges of the orientation can be at most $c(v) - 1$, since v must also dominate its private neighbor. By definition, $c(v) - 1 = \sum_{\{v,x\} \in E} w(\{v,x\})/2$. This means that for each vertex, the total weight of incoming edges is at most half the total weight of incident edges; it follows that this total weight must be equal, because when there is a vertex for which this weight is smaller, then there must be another vertex for which it is larger. So, we have an orientation that is a circulation.

Suppose now that we have a circulation that is an orientation. Add each original vertex $v \in V$ to S , and for each edge $e = \{v, x\}$, place z_e in S when the edge is oriented from v to x and otherwise place z_e'' in S . Red vertices on edge gadgets dominate all their neighbors; red original vertices dominate their private neighbor and all not yet dominated blue neighbors. This gives a dominating set where each red vertex in S dominates precisely its capacity many neighbors, as desired.

Finally, we show that we can build a log-space transducer that transforms a path decomposition of G of width ℓ to one of H with width at most $\ell + 2$. We first ensure that the path decomposition of G is nice (which can be done via a log-space transducer). We pass through the bags from left to right. For a forget bag in the path decomposition of G , we take the same bag for H . For an introduce bag $X_i = X_{i-1} \cup \{v\}$, we loop through the vertices in X_{i-1} one-by-one, say these are x_1, \dots, x_r . For each $j \in [r]$, if $\{v, x_j\} \in E$, then we add the following bags (in order):

$$X_i \cup \{z_e, y_{e,1}\}, X_i \cup \{z_e, y_{e,2}\}, \dots, X_i \cup \{z_e, y_{e,w(e)}\}, X_i \cup \{z_e, z_e'\}, X_i \cup \{z_e', z_e''\}, \\ X_i \cup \{z_e'', y_{e',1}\}, X_i \cup \{z_e'', y_{e',2}\}, \dots, X_i \cup \{z_e'', y_{e',w(e)}\}.$$

One can verify that this gives a path decomposition of H . The width has increased by at most 2.

A standard transformation now shows that CAPACITATED DOMINATING SET is also XNLP-hard with pathwidth as parameter. Given an instance (G, w) of CAPACITATED RED-BLUE DOMINATING SET, we build an equivalent instance of CAPACITATED DOMINATING SET. We give each blue vertex capacity zero. We add two new vertices x and x' , with x' of degree one and x adjacent to all red vertices and to x' . The capacity of x is equal to the number of red vertices plus 2. We increase the target size of the solution by one (and remove all colors). The pathwidth has gone up by at most one. ◀

We remark that a similar reduction can be used to show XNLP-hardness of CAPACITATED VERTEX COVER (by removing the vertex z_e' , having parallel paths of length 3 instead of 2 in the gadget of Figure 2 and giving each original vertex a new neighbor of degree one).

4.3 q -Coloring parameterized by linear mim-width

In this section we show that for each fixed $q \geq 5$, q -COLORING is XNLP-complete when parameterized by the linear mim-width of the input graph.

q -COLORING

Input: A graph $G = (V, E)$ and a linear order of V of mim-width w .

Parameter: w .

Question: Does G have a proper vertex-coloring with q colors?

XNLP-membership for this problem will be shown via the corresponding known dynamic programming XP-algorithm [8] which is based on the following.

► **Definition 13** (Neighborhood Equivalence). *Let $G = (V, E)$ be a graph and $A \subseteq V$. For all $X, Y \subseteq A$: $X \equiv_A Y \Leftrightarrow N(X) \cap (V \setminus A) = N(Y) \cap (V \setminus A)$.*

► **Lemma 14.** *q -COLORING parameterized by the mim-width of a linear order of the vertices of the input graph is in XNLP.*

Proof. Let n be the number of vertices of the input graph $G = (V, E)$ and w the mim-width of the given linear order v_1, \dots, v_n of V . Membership in XNLP is shown by adapting the XP-algorithm [8] to a nondeterministic polynomial-time algorithm that also uses at most $O(w \log n)$ space.

With i going from 1 to n , at step i we store partial solutions associated with the subgraph of G induced by the vertices $V_i = \{v_1, \dots, v_i\}$. (For convenience, we let $\overline{V}_i = V \setminus V_i$.) In the XP algorithm of [8], partial solutions are proper colorings of $G[V_i]$ and a table index consists of representatives of equivalence classes $\mathcal{Q}_1, \dots, \mathcal{Q}_q$ of \equiv_{V_i} such that for all $i \in [q]$, color class i in the coloring is contained in \mathcal{Q}_i .

To prove that each such coloring can be represented using $O(w \log n)$ bits, we use the following claim shown in [8]. We reprove it here to clarify that it leads to an algorithm satisfying the time and space requirements.

▷ **Claim 15.** For each $i \in [n - 1]$, and each $S_i \subseteq V_i$, there is a set $R_i \subseteq V_i$ with $R_i \equiv_{V_i} S_i$ and $|R_i| \leq w$. Furthermore, there is a polynomial-time algorithm using at most $O(w \log n)$ space that determines R_i from R_{i-1} , where $R_{i-1} \equiv_{V_{i-1}} S_i \cap V_{i-1}$ and $|R_{i-1}| \leq w$.

Proof. For $i \leq w$, we can simply let $R_i = S_i$, so suppose that $i > w \geq 1$, and that $|S_i| > w$. By induction, we can assume that we have $R_{i-1} \subseteq V_{i-1}$ of size at most w such that $R_{i-1} \equiv_{V_{i-1}} S_i \cap V_{i-1}$. Let $R'_i = R_{i-1} \cup \{v\}$. If $|R'_i| \leq w$, then we let $R_i = R'_i$ and we are done. We may assume that $|R'_i| = w + 1$. If there is some $x \in R'_i$ such that $N(R'_i \setminus \{x\}) \cap \overline{V}_i = N(R'_i) \cap \overline{V}_i$, then we let $R_i = R'_i \setminus \{x\}$ and we are done. Otherwise, we know that each vertex x in R'_i has a neighbor y in \overline{V}_i such that y is non-adjacent to all vertices in $R'_i \setminus \{x\}$. This means that these xy -edges form an induced matching in $G[V_i, \overline{V}_i]$, a contradiction. ◁

The previous claim immediately shows that each table index can be encoded using $O(qw \log n)$ bits, which is $O(w \log n)$ since q is a constant. The algorithm works as follows. Upon arrival of the next vertex v_{i+1} , we nondeterministically guess which of the q colors v_{i+1} receives. We then nondeterministically guess the table index corresponding to the updated solution. By Claim 15 we can conclude that the nondeterministic step can be implemented in polynomial time, and using only $O(w \log n)$ space. ◀

For two disjoint sets $A, B \subseteq V$, the *bipartite complement* replaces each edge with one endpoint in A and the other in B with a non-edge and vice versa. The following construction is due to Fomin et al. [16].

► **Definition 16** (Subdivision-complement, Fomin et al. [16]). Let $G = (V, E)$ be a graph, and $A, B \subseteq V$ with $A \cap B = \emptyset$. The subdivision-complement between A and B is the following operation:

1. Subdivide each edge uv with $u \in A$ and $v \in B$; call the resulting set of vertices R .
2. Take the bipartite complement between A and R and the bipartite complement between B and R .

The reason why this operation is useful for reductions for problems parameterized by mim-width are the following bounds on the maximum induced matching size of cuts resulting from this construction. This can also be derived from [16], but we include a simple direct proof here for completeness.

► **Lemma 17.** Let $G = (V, E)$ be a graph, and $A, B \subseteq V$ with $A \cap B = \emptyset$. Let G' be the graph obtained from G by applying the subdivision-complement between A and B ; let R denote the set of vertices created in the construction. Then, for all $C \in \{A, B\}$, $\text{cutmim}_{G'}(C, R) \leq 2$.

Proof. Suppose for a contradiction that there is an induced matching of size three in $G'[A, R]$, say $M = \{a_i r_i \mid i \in [3], a_i \in A, r_i \in R\}$. For all $i \in [3]$, let e_i denote the edge in G whose subdivision created vertex r_i . Since M is an induced matching and by construction, a_1 is the endpoint of e_2 and e_3 . But this implies that a_2 is not the endpoint of e_3 , and therefore that the edge $a_2 r_3$ exists in $G'[A, R]$. ◀

To prove the bound on the mim-width of linear orders constructed in the hardness proofs in this section, we need the following additional lemma which can be seen as a variation of a lemma in [6], but for linear mim-width. Recall that for a graph $G = (V, E)$ and a partition \mathcal{P} of V , the *quotient graph* G/\mathcal{P} is the graph obtained from G by contracting each part of \mathcal{P} into a single vertex. The *cutwidth* of a linear order $\Lambda = v_1, \dots, v_n$ of V , denoted by $\text{cutw}(\Lambda)$ is the maximum, over all i , of the number of edges with one endpoint in $\{v_1, \dots, v_i\}$ and the other in $\{v_{i+1}, \dots, v_n\}$.

► **Lemma 18** (♣). Let $G = (V, E)$ be a graph, let $\mathcal{P} = (P_1, \dots, P_r)$ be a partition of V , and let $G' = G/\mathcal{P}$. For all $i \in [r]$ let Λ_i be a linear order of P_i such that $\text{mimw}_{G[P_i]}(\Lambda_i) \leq c$, and suppose that for all distinct $i, j \in [r]$, $\text{cutmim}_G(P_i, P_j) \leq d$. Let $\Lambda = \Lambda_1, \Lambda_2, \dots, \Lambda_r$, and let $\Lambda' = P_1, \dots, P_r$ be the corresponding linear order of G/\mathcal{P} . Then, $\text{mimw}(\Lambda) \leq 2d \cdot \text{cutw}(\Lambda') + c$.

► **Definition 19** (Frame graph). Let $(G = (V, E), V_1, \dots, V_r, f)$ be an instance of CHAINED MULTICOLORED CLIQUE; for each $i \in [r]$, let $V(i, 1), \dots, V(i, k)$ denote the partition of V_i according to f .

The frame graph $G' = (V', E')$ is obtained from G by applying, for each $h \in [r-1]$ and each pair $(i_1, j_1), (i_2, j_2) \in \{h, h+1\} \times [k]$, where $(i_1, j_1) <_{LEX} (i_2, j_2)$, the subdivision-complement between $V(i_1, j_1)$ and $V(i_2, j_2)$.³ We denote the set of new vertices by $R(i_1, j_1, i_2, j_2)$.

For convenience, we let \mathcal{P} denote the partition of V' into $V(1, 1), \dots, V(r, k), R(1, 1, 1, 1), \dots, R(r, k, r, k)$, and we define the following auxiliary partial function $\phi: E \rightarrow V'$: For all (i_1, j_1) and (i_2, j_2) as above, for each $v_1 \in V(i_1, j_1)$ and $v_2 \in V(i_2, j_2)$ with $v_1 v_2 \in E$, we let $\phi(v_1 v_2) \in R(i_1, j_1, i_2, j_2)$ be the vertex created when subdividing $v_1 v_2$.

³ Here, $<_{LEX}$ denotes the lexicographic ordering. We have $(i_1, j_1) <_{LEX} (i_2, j_2)$ if either $i_1 < i_2$ or if $i_1 = i_2$ and $j_1 < j_2$.

► **Lemma 20.** *Let $(G = (V, E), V_1, \dots, V_r, f)$ be an instance of CHAINED MULTICOLORED CLIQUE, and let $G' = (V', E')$ be its frame graph; adapt the notation from Definition 19. Then, G' has an independent set S with $|S \cap P| = 1$ for all $P \in \mathcal{P}$ if and only if G has a chained multicolored clique.*

Proof. Suppose G' has an independent set S with $|S \cap P| = 1$ for all $P \in \mathcal{P}$. Let $v_{i,j} \in S \cap V(i, j)$ for all $i \in [r]$, $j \in [k]$. We claim that this implies that for all $h \in [r-1]$, and all $(i_1, j_1), (i_2, j_2) \in \{h, h+1\} \times [k]$ with $(i_1, j_1) <_{LEX} (i_2, j_2)$, we have that $\phi(v_{i_1, j_1} v_{i_2, j_2}) \in S \cap R(i_1, j_1, i_2, j_2)$, which implies that $v_{i_1, j_1} v_{i_2, j_2} \in E$ and in particular that $S \cap V$ is a chained multicolored clique in G . Let $r \in S \cap R(i_1, j_1, i_2, j_2)$ and suppose $r \neq \phi(v_{i_1, j_1} v_{i_2, j_2})$. We may assume that $r = \phi(v, w)$ where $v \in V(i_1, j_1) \setminus \{v_{i_1, j_1}\}$. But then, $v_{i_1, j_1} r$ is an edge in G' , a contradiction.

For the other direction, let $W \subseteq V$ be the chained multicolored clique in G . Let $S = \emptyset$. For each $i \in [r]$ and $j \in [k]$, we add the vertex $v_{i,j} \in W \cap V(i, j)$ to S . Next, for each $h \in [r-1]$, and each pair $(i_1, j_1), (i_2, j_2) \in \{h, h+1\} \times [k]$ where $(i_1, j_1) <_{LEX} (i_2, j_2)$, we add $\phi(v_{i_1, j_1} v_{i_2, j_2})$ to S . Note that since W is a chained multicolored clique, the edge $v_{i_1, j_1} v_{i_2, j_2}$ always exists in G . It follows immediately from the construction that S is an independent set in G' , and that for all $P \in \mathcal{P}$, $|S \cap P| = 1$. ◀

► **Lemma 21.** *For fixed $q \geq 5$, q -COLORING parameterized by the mim-width of a given linear order of the vertices of the input graph is XNLP-hard.*

Proof. We give a parameterized logspace reduction from CHAINED MULTICOLORED CLIQUE to 5-LIST-COLORING. Let $\mathcal{I} = (G = (V, E), V_1, \dots, V_r, f)$ be the instance of CHAINED MULTICOLORED CLIQUE. We create the graph $G'' = (V'', E'')$ of the 5-LIST-COLORING instance as follows: Let $G' = (V', E')$ be the frame graph of \mathcal{I} and adapt the notation of Definition 19. We obtain G'' and the lists $L'' = \{L(v) \mid v \in V''\}$ as follows:

- For each $P \in \mathcal{P}$, we add two vertices $a(P)$ and $b(P)$, and make $P'' = P \cup \{a(P), b(P)\}$ a path from $a(P)$ to $b(P)$. We let $\mathcal{P}'' = \{P'' \mid P \in \mathcal{P}\}$.
- For each $(i, j) \in [r] \times [k]$, each list of a vertex in $P = V(i, j)$ is $\{3\}$. If $|P|$ is even, the lists of both $a(P)$ and $b(P)$ are $\{1\}$; and if $|P|$ is odd, the list of $a(P)$ is $\{1\}$, and the list of $b(P)$ is $\{2\}$.
- For each $h \in [r-1]$ and $(i_1, j_1), (i_2, j_2) \in \{h, h+1\} \times [k]$ with $(i_1, j_1) <_{LEX} (i_2, j_2)$, each list of a vertex in $R = R(i_1, j_1, i_2, j_2)$ is $\{3, 4, 5\}$. If $|R|$ is even, the lists of both $a(R)$ and $b(R)$ are $\{5\}$; and if $|R|$ is odd, the list of $a(R)$ is $\{5\}$, and the list of $b(R)$ is $\{4\}$.

The following observation is immediate from the above construction.

► **Observation 22.** *In each proper list coloring of (G'', L'') and each $P \in \mathcal{P}$, there is a vertex in P that received color 3. Conversely, if some vertex $v \in P$ received color 3 in a proper list coloring of (G'', L'') , then the vertices in $P'' \setminus \{v\}$ can be properly list-colored with colors $\{1, 2\}$, if $P = V(i, j)$ for some i, j or with colors $\{4, 5\}$, if $P = R(i_1, j_1, i_2, j_2)$, for some i_1, i_2, j_1, j_2 .*

Now suppose that G has a chained multicolored clique W . Then by Lemma 20, there is an independent set S in G' such that $|S \cap P| = 1$ for all $P \in \mathcal{P}$. Note that S is also an independent set in G'' . We can therefore let S be color class 3, and by Observation 22, the remaining vertices of each path P can be properly list colored without using color 3. It suffices to check the edges between $V(i_1, j_1)$ and $R(i_1, j_1, i_2, j_2)$, for any valid choice of i_1, i_2, j_1, j_2 . Since S is an independent set, the vertices $v \in S \cap V(i_1, j_1)$ and $w \in S \cap R(i_1, j_1, i_2, j_2)$ are non-adjacent. Furthermore, v has a different color from all vertices in $R(i_1, j_1, i_2, j_2) \setminus S$. Finally, the sets of colors appearing on $V(i_1, j_1) \setminus S$ and $R(i_1, j_1, i_2, j_2) \setminus S$ are disjoint.

Conversely, suppose that (G'', L'') has a proper list-coloring. Then we can combine Observation 22 and Lemma 20 to conclude that G has a chained multicolored clique (with color class 3 being the independent set required by Lemma 20).

We conclude with the following claim.

▷ **Claim 23.** One can in polynomial time and logarithmic space construct a linear order Λ of V'' such that $\text{mimw}(\Lambda) = O(k^2)$.

Proof. Each part $P \in \mathcal{P}$, together with $a(P)$ and $b(P)$ forms a path. Therefore we can trivially obtain a linear order of $P \cup \{a(P), b(P)\}$ by following the path from $a(P)$ to $b(P)$ whose mim-width is 1. For all $i \in [r]$ and $j \in [k]$, we let $\Lambda(i, j)$ be such a linear order where P corresponds to $V(i, j)$. For all $h \in [r - 1]$, and all $(i_1, j_1), (i_2, j_2) \in \{h, h + 1\} \times [k]$ with $(i_1, j_1) <_{LEX} (i_2, j_2)$, we let $\Gamma(i_1, j_1, i_2, j_2)$ be such a linear order where P corresponds to $R(i_1, j_1, i_2, j_2)$. The desired linear order Λ traverses V'' as follows: Consider $(i, j) \in [r] \times [k]$ in lexicographically increasing order. First, we follow $\Lambda(i, j)$, and then $\Gamma(i, j, i, j + 1), \dots, \Gamma(i, j, i, k)$, and if $i < r$, then $\Gamma(i, j, i + 1, 1), \dots, \Gamma(i, j, i + 1, k)$. This linear order of G'' can be created using $O(\log n)$ bits of memory, where $n = |V|$.

As pointed out above, each $\Lambda(i, j)$ and each $\Gamma(i_1, j_1, i_2, j_2)$ has mim-width at most 1. The only edges in G' between different parts of \mathcal{P} are between $V(i_1, j_1)$ and $R(i_1, j_1, i_2, j_2)$, and between $V(i_2, j_2)$ and $R(i_1, j_1, i_2, j_2)$, where $(i_1, j_1) <_{LEX} (i_2, j_2)$. By construction it therefore follows from Lemma 17 that for each pair of distinct parts $P_1, P_2 \in \mathcal{P}$, $\text{cutmim}_{G'}(P_1, P_2) \leq 2$. Let Λ' be the linear order of the vertices of G''/\mathcal{P} such Λ can be obtained by traversing the parts of \mathcal{P} in the order of Λ' , and then following the above described order on each part of \mathcal{P} . We can observe that $\text{cutw}(\Lambda') = O(k^2)$, and therefore the claim follows from Lemma 18. ◁

We have shown that 5-LIST-COLORING parameterized by the mim-width of a given linear order of the input graph is XNLP-hard. To derive XNLP-hardness of 5-COLORING in the same parameterization, observe that we can use the standard trick of adding a clique on vertices $\{1, \dots, 5\}$, and for each $i \in [5]$, connecting i and v if $i \notin L(v)$. Since adding c vertices can only increase the mim-width of a given linear order by at most c , no matter where the new vertices are placed, this does not prohibitively increase the linear mim-width either.

To obtain hardness for any $q > 5$, we simply add $q - 5$ universal vertices to the 5-COLORING instance obtained in the previous paragraph. Adding universal vertices cannot increase the mim-width w of any linear order, regardless of where they are placed, unless $w = 0$. ◀

Combining Lemmas 14 and 21, we obtain that q -COLORING is XNLP-complete parameterized by linear mim-width, for each fixed $q \geq 5$.

5 Conclusion

In this paper, we gave a number of XNLP-completeness proofs for graph problems parameterized by linear width measures. Such results are interesting for a number of reasons: they pinpoint the “right” complexity class for parameterized problems, they imply hardness for all classes $W[t]$, and they tell that it is unlikely that there is an algorithm that uses ‘XP time and FPT space’ by Conjecture 1.

This paper gives among others the first examples of XNLP-complete problems when the linear clique-width or linear mim-width is taken as parameter. Our hardness results give new starting points for future hardness and completeness proofs, in particular for problems with width measures like pathwidth, (linear) clique-width, or (linear) mim-width as parameter.

Other interesting directions for future research in this line are, for instance, to consider the parameterization by *cutwidth*; a promising candidate problem to show XNLP-completeness parameterized by cutwidth is LIST EDGE COLORING. Another interesting parameter to consider in this context is the *degeneracy* of a graph. It is also interesting to explore the concept of XNLP-completeness for width measures of other objects than graphs. One could for instance consider (linear) width measures of digraphs or hypergraphs.

We also leave open what the correct parameterized complexity class is for FEEDBACK VERTEX SET parameterized by logarithmic pathwidth or logarithmic linear cliquewidth – we showed XNLP-hardness, but did not prove containment in XNLP.

References

- 1 Yuichi Asahiro, Hiroshi Eto, Takehiro Ito, and Eiji Miyano. Complexity of finding maximum regular induced subgraphs with prescribed degree. *Theor. Comput. Sci.*, 550:21–35, 2014. doi:10.1016/j.tcs.2014.07.008.
- 2 Hans L. Bodlaender. Parameterized complexity of bandwidth of caterpillars and weighted path emulation. In Lukasz Kowalik, Michal Pilipczuk, and Pawel Rzazewski, editors, *Proceedings of the 47th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2021)*, volume 12911 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2021. doi:10.1007/978-3-030-86838-3_2.
- 3 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, 2022. doi:10.1007/978-3-031-15914-5_7.
- 4 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, 2021. doi:10.1109/FOCS52979.2021.00027.
- 5 Hans L. Bodlaender, Daniel Lokshtanov, and Eelko Penninkx. Planar capacitated dominating set is $W[1]$ -hard. In Jianer Chen and Fedor V. Fomin, editors, *Proceedings 4th International Workshop on Parameterized and Exact Computation, IWPEC 2009*, volume 5917 of *Lecture Notes in Computer Science*, pages 50–60. Springer, 2009. doi:10.1007/978-3-642-11269-0_4.
- 6 Nick Brettell, Jake Horsfield, Andrea Munaro, and Daniël Paulusma. List k -colouring P_t -free graphs: A mim-width perspective. *Inf. Process. Lett.*, 173:106168, 2022. doi:10.1016/j.ipl.2021.106168.
- 7 Hajo Broersma, Petr A. Golovach, and Viresh Patel. Tight complexity bounds for FPT subgraph problems parameterized by the clique-width. *Theor. Comput. Sci.*, 485:69–84, 2013. doi:10.1016/j.tcs.2013.03.008.
- 8 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theor. Comput. Sci.*, 511:66–76, 2013. doi:10.1016/j.tcs.2013.01.009.
- 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 Michael Dom, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. Capacitated domination and covering: A parameterized perspective. In Martin Grohe and Rolf Niedermeier, editors, *Proceedings 3rd International Workshop on Parameterized and Exact Computation, IWPEC 2008*, volume 5018 of *Lecture Notes in Computer Science*, pages 78–90. Springer, 2008. doi:10.1007/978-3-540-79723-4_9.
- 11 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.

- 12 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.
- 13 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010. doi:10.1137/080742270.
- 14 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014. doi:10.1137/130910932.
- 15 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: Hamiltonian cycle and the odd case of graph coloring. *ACM Trans. Algorithms*, 15(1):9:1–9:27, 2019. doi:10.1145/3280824.
- 16 Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the tractability of optimization problems on H-graphs. *Algorithmica*, 82(9):2432–2473, 2020. doi:10.1007/s00453-020-00692-9.
- 17 Sylvain Guillemot. Parameterized complexity and approximability of the Longest Compatible Sequence problem. *Discret. Optim.*, 8(1):50–60, 2011. doi:10.1016/j.disopt.2010.08.003.
- 18 Frank Gurski and Egon Wanke. Line graphs of bounded clique-width. *Discret. Math.*, 307(22):2734–2754, 2007. doi:10.1016/j.disc.2007.01.020.
- 19 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width II. The feedback vertex set problem. *Algorithmica*, 82:118–145, 2020.
- 20 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.
- 21 Luke Mathieson and Stefan Szeider. The parameterized complexity of regular subgraph problems and generalizations. In James Harland and Prabhu Manyem, editors, *Proceedings 14th Computing: The Australasian Theory Symposium, CATS 2008*, volume 77 of *CRPIT*, pages 79–86. Australian Computer Society, 2008. URL: <http://crpit.scem.westernsydney.edu.au/abstracts/CRPITV77Mathieson.html>.
- 22 Hannes Moser and Dimitrios M. Thilikos. Parameterized complexity of finding regular induced subgraphs. *J. Discrete Algorithms*, 7(2):181–190, 2009. doi:10.1016/j.jda.2008.09.005.
- 23 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.
- 24 Martin Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, Norway, 2012.
- 25 Egon Wanke. k -NLC graphs and polynomial algorithms. *Discret. Appl. Math.*, 54(2-3):251–266, 1994. doi:10.1016/0166-218X(94)90026-4.

Twin-Width VIII: Delineation and Win-Wins

Édouard Bonnet   

Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Dibyayan Chakraborty 

Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

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

Raul Lopes  

Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Stéphan Thomassé 

Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Abstract

We introduce the notion of delineation. A graph class \mathcal{C} is said *delineated by twin-width* (or simply, *delineated*) if for every hereditary closure \mathcal{D} of a subclass of \mathcal{C} , it holds that \mathcal{D} has bounded twin-width if and only if \mathcal{D} is monadically dependent. An effective strengthening of *delineation* for a class \mathcal{C} implies that tractable FO model checking on \mathcal{C} is perfectly understood: On hereditary closures of subclasses \mathcal{D} of \mathcal{C} , FO model checking on \mathcal{D} is fixed-parameter tractable (FPT) exactly when \mathcal{D} has bounded twin-width. Ordered graphs [BGODMSTT, STOC '22] and permutation graphs [BKTW, JACM '22] are effectively delineated, while subcubic graphs are not. On the one hand, we prove that interval graphs, and even, rooted directed path graphs are delineated. On the other hand, we observe or show that segment graphs, directed path graphs (with arbitrarily many roots), and visibility graphs of simple polygons are not delineated.

In an effort to draw the delineation frontier between interval graphs (that are delineated) and axis-parallel two-lengthed segment graphs (that are not), we investigate the twin-width of restricted segment intersection classes. It was known that (triangle-free) pure axis-parallel unit segment graphs have unbounded twin-width [BGKTW, SODA '21]. We show that $K_{t,t}$ -free segment graphs, and axis-parallel H_t -free unit segment graphs have bounded twin-width, where H_t is the half-graph or ladder of height t . In contrast, axis-parallel H_4 -free two-lengthed segment graphs have unbounded twin-width. We leave as an open question whether unit segment graphs are delineated.

More broadly, we explore which structures (large bicliques, half-graphs, or independent sets) are responsible for making the twin-width large on the main classes of intersection and visibility graphs. Our new results, combined with the FPT algorithm for first-order model checking on graphs given with $O(1)$ -sequences [BKTW, JACM '22], give rise to a variety of algorithmic win-win arguments. They all fall in the same framework: If p is an FO definable graph parameter that effectively functionally upperbounds twin-width on a class \mathcal{C} , then $p(G) \geq k$ can be decided in FPT time $f(k) \cdot |V(G)|^{O(1)}$. For instance, we readily derive FPT algorithms for k -LADDER on visibility graphs of 1.5D terrains, and k -INDEPENDENT SET on visibility graphs of simple polygons. This showcases that the theory of twin-width can serve outside of classes of bounded twin-width.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis; Theory of computation \rightarrow Fixed parameter tractability

Keywords and phrases Twin-width, intersection graphs, visibility graphs, monadic dependence and stability, first-order model checking

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.9

Related Version *Full Version:* <https://arxiv.org/abs/2204.00722>



© Édouard Bonnet, Dibyayan Chakraborty, Eun Jung Kim, Noleen Köhler, Raul Lopes, and Stéphan Thomassé;

licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 9; pp. 9:1–9:18



Leibniz International Proceedings in Informatics

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

1 Introduction

A *trigraph* G has a vertex set $V(G)$, and two disjoint edge sets, the black edge set $E(G)$ and the red edge set $R(G)$. A (vertex) *contraction* consists of merging two (non-necessarily adjacent) vertices, say, u, v into a vertex w , and keeping every existing edge wz black if and only if uz and vz were previously black edges. The other edges incident to w turn red (if not already), while the rest of the trigraph remains the same. A *contraction sequence* of an n -vertex (tri)graph G is a sequence of trigraphs $G = G_n, \dots, G_1 = K_1$ such that G_i is obtained from G_{i+1} by performing one contraction. A *d-sequence* is a contraction sequence wherein all trigraphs have red degree at most d . The *twin-width* of G , denoted by $\text{tw}(G)$, is the minimum integer d such that G admits a d -sequence. A graph class \mathcal{C} has then *bounded twin-width* if there is a constant t such that every graph $G \in \mathcal{C}$ satisfies $\text{tw}(G) \leq t$. See Figure 1 for an illustration of a 2-sequence of a graph.

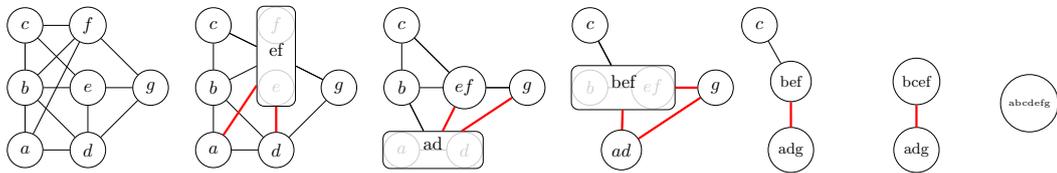


Figure 1 A 2-sequence witnesses that the initial 7-vertex graph has twin-width at most 2.

The main algorithmic application of twin-width is that first-order (FO) model checking, that is, deciding if a first-order sentence φ holds in a graph G , can be decided in fixed-parameter time (FPT) $f(|\varphi|, d) \cdot |V(G)|$ for some computable function f , when given a d -sequence of G [6]. We recall that there is an ample list of graph classes (or more generally of binary structures, since the definition of twin-width extends to them) of bounded twin-width, including bounded clique-width graphs, H -minor free graphs, posets with antichains of bounded size, strict subclasses of permutation graphs, map graphs, bounded-degree string graphs [6], as well as $\Omega(\log n)$ -subdivisions of n -vertex graphs, and some particular classes of cubic expanders [4]. In contrast, (sub)cubic graphs, interval graphs, triangle-free unit segment graphs, unit disk graphs have unbounded twin-width [4].

The missing element for an FPT FO model-checking algorithm on any class of bounded twin-width is a polynomial-time algorithm and a computable function f , that given a constant integer bound c and a graph G , either finds an $f(c)$ -sequence for G , or correctly reports that $\text{tw}(G) > c$. The runtime of the algorithm could be $n^{g(c)}$, for some function g . However to get an FPT algorithm in the combined parameter *size of the sentence + bound on the twin-width*, one would further require that the approximation algorithm takes FPT time in c (now thought of as a parameter), i.e., $g(c)n^{O(1)}$. Such an algorithm exists on ordered graphs (more generally, ordered binary structures) [5], graphs of bounded clique-width, proper minor-closed classes [6], but not on general graphs. Let us observe that exactly computing the twin-width, and even distinguishing between $\text{tw}(G) = 4$ and $\text{tw}(G) = 5$, is NP-complete [3].

Motivation. We aim to get around the two main caveats of using twin-width for algorithm design. Namely:

- an FPT (or XP) approximation of twin-width is still missing, and
- a priori *only* classes of bounded twin-width are concerned.

The central theme of this paper is to showcase how to bypass the above caveats using twin-width and to provide a necessary toolbox. First we show that on certain graph classes, bounded twin-width is precisely what renders FO model-checking FPT, and the notion of *delineation* is introduced to this end. Second, we demonstrate how to summon a win-win strategy on important graph classes by means of twin-width, which is reminiscent of the well-known win-win argument based on treewidth.

The main obstacle for computing the twin-width is to get a good vertex ordering. Geometric graph classes of *unbounded* twin-width constitute a diverse and intriguing pool for testing these two avenues: interval graphs, (rooted) directed path graphs, segment graphs, visibility graphs of polygons and terrains. For all these classes, a vertex-ordering procedure either comes naturally or can be worked out and efficiently computed.

Global strategy. For our purpose, the following characterization of bounded twin-width will be pivotal.

► **Theorem 1** ([5]). *A class \mathcal{C} has bounded twin-width if and only if there is an integer k such that every graph of \mathcal{C} admits an adjacency matrix without rank- k division, i.e., k -division such that every cell has combinatorial rank at least k .*

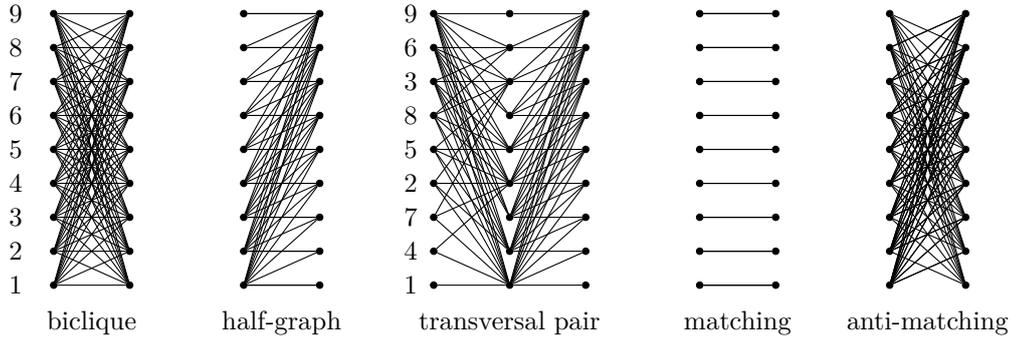
Here, a k -division of a matrix is a partition of its column (resp. row) set into k intervals, called *column* (resp. *row*) *parts*, of consecutive columns (resp. rows). A k -division naturally defines k^2 *cells* (contiguous submatrices) made by the entries at the intersection of a column part with a row part. A *rank- k division* of M is a k -division \mathcal{D} such that each of the k^2 cells has at least k distinct rows or at least k distinct columns (that is, combinatorial rank at least k). The maximum integer k such that M admits a rank- k division is called *grid rank*, and is denoted by $\text{gr}(M)$. Theorem 1 is effective: There is a computable function f , such that, given a vertex ordering along which the adjacency matrix of a graph G has no rank- k division, one can efficiently find an $f(k)$ -sequence for G , witnessing that $\text{tw}(G) \leq f(k)$.

Suppose that for a graph class \mathcal{C} , a canonical vertex ordering can be obtained. Either the consequential adjacency matrix has no rank- k division – and we get a favorable contraction sequence by Theorem 1 – or it does have such a division. In the latter case, a large structured object of variable complexity may be found, such as a biclique, a half-graph (or ladder), or even an obstacle to an FPT FO model checking in the form of a *transversal pair of half-graphs* (or *transversal pair*, for short) or some variant of it; see the middle figure in Figure 2, Section 3 for a formal definition, and for why transversal pairs indeed are such obstacles.

Delineation. For monotone (i.e., closed under removing vertices and edges) classes, the FPT algorithm of Grohe, Kreutzer, and Siebertz [20] for FO model checking on nowhere dense classes, is complemented by W[1]-hardness on classes that are somewhere dense (i.e., *not* nowhere dense) [13], and even AW[*]-hardness on classes that are *effectively* somewhere dense [23]. The latter two results imply that, for monotone classes, FO model checking is unlikely to be FPT beyond nowhere dense classes. Thus the classification of monotone classes admitting an FPT FO model checking is complete. However such a classification remains an active line of work for the more general hereditary classes of graphs and binary structures [15, 18, 19]. It is conjectured (see for instance [18, Conjecture 8.2]) that:

► **Conjecture 2.** *For every hereditary class \mathcal{C} of structures, FO model checking is FPT on \mathcal{C} if and only if \mathcal{C} is monadically dependent.¹*

¹ A model-theoretic notion which roughly says that not every graph G can be built from a nondeterministic $O(1)$ -coloring of some $\mathcal{S} \in \mathcal{C}$ by means of a first-order formula $\varphi(x, y)$, in the relations of \mathcal{S} and the added colors, imposing the edge set of G ; see Section 2 for a definition.



■ **Figure 2** Biclique, half-graph (or ladder), transversal pair of half-graphs, matching, anti-matching, all of height 9. Bicliques and half-graphs are semi-induced by default. The number next to each leftmost vertex v of the transversal pair indicates the height of the neighbor of v in the central column which is not also a neighbor of the vertex just below v .

If for every hereditary closure \mathcal{D} of a subclass² of \mathcal{C} , \mathcal{D} has bounded twin-width if and only if \mathcal{D} is monadically dependent, we say that \mathcal{C} is *delineated by twin-width* (or simply, *delineated*). Although not stated in those terms, permutation graphs were already proven to be delineated [6], as well as ordered graphs [5]. We add interval graphs and rooted directed path graphs (see Section 3 for a definition) to the list of delineated classes. Therefore, for every hereditary subclass of these classes the classification of FPT FO model checking, Conjecture 2, is now provably settled.³ In contrast, we rule out delineation for directed path graphs (with multiple roots), intersection graphs of pure axis-parallel segments with two distinct lengths, and visibility graphs of simple polygons.

► **Theorem 3.** *Interval graphs, and more generally rooted directed path graphs, are delineated.*

A (variant of a) transversal pair plays the key role to establish Theorem 3. We show that on a class \mathcal{C} , if a (variant of a) transversal pair can systematically be found as a result of unbounded twin-width, then the classification of FPT FO model checking for hereditary subclasses of \mathcal{C} is entirely settled by the algorithm on graphs of bounded twin-width [6].

Twin-width win-wins. If segment graphs and visibility graphs of simple polygons do not yield in their subfamilies of unbounded twin-width complex enough structures to settle Conjecture 2, unbounded twin-width still imply in those classes that some other graph parameters are unbounded. This gives rise to a win-win approach to compute these parameters. To give a context, we draw a parallel with what happens with treewidth.

The algorithmic use of a parameter like treewidth extends beyond classes wherein treewidth is bounded. Any problem admitting an FPT algorithm parameterized by treewidth (like MSO definable problems [9]), and a trivial answer (such as a systematic YES or a systematic NO) when the treewidth is large, subjects itself to a straightforward win-win argument. This is at the basis of the so-called *bidimensionality* theory [17]. Since a problem like k -VERTEX COVER admits a $2^{\text{tw}(G)}n^{O(1)}$ -time algorithm [10] and a systematic NO answer in presence of a, say, $(2\sqrt{k} + 1) \times (2\sqrt{k} + 1)$ grid minor, one then derives for this problem an FPT algorithm running in time $2^{O(\sqrt{k})}n^{O(1)}$ in planar graphs.

² The reason we do not simply quantify over hereditary subclasses of \mathcal{C} is to have a notion that is also meaningful when \mathcal{C} is not hereditary.

³ We actually need an effective strengthening of delineation that also holds for these classes and will be defined in Section 2.

Let us forget one moment the intermediary role of the grid minor. Efficiently computing a parameter $p(G)$ – like the *vertex cover number* $\tau(G)$ – can boil down to establishing an upperbound of the form $\text{tw}(G) \leq f(p(G))$.

We explore such upper bounds, and resultant win-wins, with twin-width in place of treewidth. Given two graph parameters p, q , and a graph class \mathcal{C} , we will write $p \sqsubseteq q$ on \mathcal{C} to signify that there is a computable function f such that $\forall G \in \mathcal{C}, p(G) \leq f(q(G))$. By a similar argument to what was presented in the previous paragraphs, one gets the following.

► **Theorem 4** (informal). *Let \mathcal{C} be a graph class and p be a graph invariant such that*

1. *computing p is FPT in the combined parameter $p + \text{tw}$ on \mathcal{C} , and*
2. *$\text{tw} \sqsubseteq p$ on \mathcal{C} .*

Then, computing p is FPT on \mathcal{C} .

First-order logic yields a natural pool of invariants p that are fixed-parameter tractable with respect to $p + \text{tw}$ [6]. As a first example of Item 2, we show the following.

► **Theorem 5.** *Biclique-free segment graphs have bounded twin-width. Furthermore, if a geometric representation is given, an $O(1)$ -sequence of the graph is found in polynomial time.*

A reformulation is that, in segment graphs, twin-width is upperbounded by a function of the largest biclique; or, denoting by $\beta(G)$ the largest integer t such that G admits a biclique $K_{t,t}$ as a subgraph, it holds that $\text{tw} \sqsubseteq \beta$ on segment graphs. The corresponding problem k -BICLIQUE was famously shown W[1]-hard by Lin [25], after its parameterized complexity has been open for over a decade [11]. From Theorems 5 and 19 one rederives⁴ that k -BICLIQUE is FPT on segment graphs given with a geometric representation.

The counterpart of the large grid minor (in treewidth win-wins) is a large rank division in *every* adjacency matrix of the graph (recall Theorem 1). Large twin-width in a class \mathcal{C} in particular implies a large rank division in the adjacency matrix along a vertex ordering that, at least partially, respects the structure of \mathcal{C} . In turn, this complex structure – despite being along a canonical order – may help lowerbounding other parameters (like the grid minor was lowerbounding the vertex cover number in our example). We give two such examples, both on classes of visibility graphs.

A *simple polygon* is a polygon without holes. Two vertices (more generally, points) of a polygon *see* each other if the line segment defined by these vertices (or points) is entirely contained in the polygon. The following problem is sometimes advertised as *hiding (people) in polygons*, and its solution is called a *hidden set*. It is NP-complete [27], even APX-hard [16], and can be equivalently defined as k -INDEPENDENT SET in visibility graphs of simple polygons given with a representation.

► **Theorem 6.** *Given a simple polygon \mathcal{P} and an integer k , finding k vertices of \mathcal{P} pairwise not seeing each other is FPT.*

A key step for proving Theorem 6 is to turn a large rank division in the adjacency matrix along a Hamiltonian path describing the boundary of the polygon into a large independent set. In conclusion: we establish $\text{tw} \sqsubseteq \alpha$ in visibility graphs of simple polygons (where $\alpha(G)$ is the independence number of G), which immediately implies Theorem 6 thanks to Theorem 4.

⁴ This fact can alternatively be obtained via the algorithmic theory of Sparsity [13, 14], and the existence of truly sublinear balanced separators in $K_{t,t}$ -free segment graphs [24].

In contrast, k -DOMINATING SET remains $W[1]$ -hard on visibility graphs of simple polygons [7], thus likely does *not* admit an FPT algorithm. We remark that Hliněný et al. [22] conjectured that FO model checking is FPT on weak visibility graphs of simple polygons additionally parameterized by the independence number. Our proof that $\text{tw} \sqsubseteq \alpha$ on visibility graphs of simple polygons confirms this conjecture, even for the more general (non-weak) visibility graphs. We observe that the approach would not work with a classic width measure, since none of the three items hold replacing *twin-width* by *clique-width*; this mainly because grids and long paths of consistently ordered half-graphs have bounded twin-width but unbounded clique-width.

A *1.5D terrain* (or here, *terrain* for short) is an x -monotone polygonal chain in the plane. Two vertices of a terrain *see* each other if the line segment they define entirely lies above the terrain. Let $\lambda(G)$, the *ladder index* of G , be the greatest height of a semi-induced half-graph in G . A folklore structural property of terrains, often called *Order Claim*, imposes the existence of large half-graphs in a large rank division along the left-right ordering. Thus $\text{tw} \sqsubseteq \lambda$ in visibility graphs of 1.5D terrains. We conclude:

► **Theorem 7.** *k -LADDER and k -BICLIQUE are FPT on visibility graphs of 1.5D terrains given with a geometric representation.*

The full version of this paper is available on arXiv, where all missing proofs can be found.

2 Preliminaries

We may denote the set of integers between i and j by $[i, j]$, and $[k]$ may be used as a short-hand for $[1, k]$.

2.1 Graph theory

We use the standard graph-theoretic definitions and notations. We denote by $V(G)$, and $E(G)$, the vertex set, and the edge set, of a graph G , and by $G[S]$ the subgraph of G induced by $S \subseteq V(G)$. When $A, B \subseteq V(G)$ are two disjoint sets, we denote by $G[A, B]$ the bipartite graph $(A, B, \{ab : a \in A, b \in B, ab \in E(G)\})$. We denote by $\text{Adj}_{\prec}(G)$ the adjacency matrix of G along the total order \prec of $V(G)$.

A *biclique* and *half-graph* (or *ladder*) of height t play a central role in this paper. The formal definition can be found in the long version, and See Figure 2 for illustrations. A bipartite graph H is *semi-induced* in G if there are two disjoint $A, B \subseteq V(G)$ such that H is isomorphic to $G[A, B]$. A graph is $K_{t,t}$ -free (resp. H_t -free) if it does not contain $K_{t,t}$ (resp. H_t) as a semi-induced subgraph.

2.2 Model checking, interpretations, transductions, and dependence

A relational *signature* σ is a finite set of relation symbols R , each having a specified arity $r \in \mathbb{N}$. A σ -*structure* \mathbf{A} is defined by a set A (the *domain* of \mathbf{A}) and a relation $R^{\mathbf{A}} \subseteq A^r$ for each relation symbol $R \in \sigma$ with arity r .

A *binary structure* is a relational structure with symbols of arity at most 2. The syntax and semantics of first-order formulas over σ (or σ -*formulas* for short), are defined as usual. We recall that a *sentence* is a formula without free variable. Most of the time we will consider σ -structures with σ consisting of a single binary relation symbol E , and identify them to graphs. But we will also deal with binary structures that are graphs augmented with a total order (called *totally ordered graphs*, or *ordered graphs* for short) and/or some unary relations.

Interpretations, transductions, and monadic dependence. Let σ, τ be relational signatures. A *simple FO interpretation* (here, *FO interpretation* for short) I of τ -structures in σ -structures consists of the following σ -formulas: a *domain* formula $\nu(x)$, and for each relation symbol $R \in \tau$ of arity r , a formula $\varphi_R(x_1, \dots, x_r)$. If \mathbf{A} is a σ -structure, the τ -structure $\mathsf{I}(\mathbf{A})$ has domain $\nu(\mathbf{A}) = \{v \in A : \mathbf{A} \models \nu(v)\}$ and the interpretation of a relation symbol $R \in \sigma$ of arity r is $R^{\mathsf{I}(\mathbf{A})} = \{(v_1, \dots, v_r) \in \nu(\mathbf{A})^r : \mathbf{A} \models \varphi_R(v_1, \dots, v_r)\}$. If \mathcal{C} is a class of σ -structures, we set $\mathsf{I}(\mathcal{C}) = \{\mathsf{I}(\mathbf{A}) : \mathbf{A} \in \mathcal{C}\}$.

Let $\sigma \subseteq \sigma^+$ be relational signatures. The σ -*reduct* of a σ^+ -structure \mathbf{A} , denoted by $\text{reduct}_{\sigma^+ \rightarrow \sigma}(\mathbf{A})$, is the structure obtained from \mathbf{A} by deleting all the relations not in σ . A *monadic h -lift* of a σ -structure \mathbf{A} is a σ^+ -structure \mathbf{A}^+ , where σ^+ is the union of σ and a set of h unary relation symbols, and $\text{reduct}_{\sigma^+ \rightarrow \sigma}(\mathbf{A}^+) = \mathbf{A}$.

A *simple non-copying FO transduction* (here, *FO transduction* for short) T of τ -structures in σ -structures is an interpretation of τ -structures in σ^+ -structures, where the σ^+ -structures are *monadic h -lifts* of σ -structures for some fixed integer h . As there are many ways of interpreting the extra unary relations, a transduction (contrary to an interpretation) builds on a given input structure several output structures. If \mathcal{C} is a class of σ -structures, $\mathsf{T}(\mathcal{C})$ denotes the class of all the τ -structures output on any σ -structure $\mathbf{A} \in \mathcal{C}$.

We say that a class \mathcal{C} *interprets* a class \mathcal{D} (or that \mathcal{D} *interprets in* \mathcal{C}) if there is an interpretation I such that $\mathcal{D} \subseteq \mathsf{I}(\mathcal{C})$. Further, a class \mathcal{C} *efficiently interprets* \mathcal{D} if additionally a polytime algorithm inputs $\mathbf{A} \in \mathcal{D}$, and outputs a structure $\mathbf{B} \in \mathcal{C}$ such that $\mathsf{I}(\mathbf{B})$ is isomorphic to \mathbf{A} . Similarly, we say that a class \mathcal{C} *transduces* a class \mathcal{D} (or that \mathcal{D} *transduces in* \mathcal{C}) if there is a transduction T such that $\mathcal{D} \subseteq \mathsf{T}(\mathcal{C})$. Two classes \mathcal{C} and \mathcal{D} are *transduction equivalent* if \mathcal{C} transduces \mathcal{D} , and \mathcal{D} transduces \mathcal{C} . We will frequently use the fact that one can compose transductions: If \mathcal{C} transduces \mathcal{D} , and \mathcal{D} transduces \mathcal{E} , then \mathcal{C} transduces \mathcal{E} .

The following is a particularly useful fact to bound the twin-width of a class.

► **Theorem 8** ([6]). *Every FO transduction of a class with bounded twin-width has bounded twin-width.*

Furthermore, given an FO transduction T and a class \mathcal{C} on which $O(1)$ -sequences can be computed in polynomial time, one can also compute $O(1)$ -sequences for graphs of $\mathsf{T}(\mathcal{C})$ in polynomial time.

We will not need the original definition of monadic dependence; solely the following characterization:

► **Theorem 9** (Baldwin and Shelah [1]). *\mathcal{C} is monadically dependent if and only if \mathcal{C} does not transduce the class \mathcal{G} of all finite graphs.*

Since FO model checking on the class of all graphs is AW[*]-hard [12], one notices that if \mathcal{C} efficiently interprets the class of all graphs then FO model checking on \mathcal{C} is AW[*]-hard[1, 12]. Conjecture 2 anticipates that every hereditary class of structures not transducing the class of all graphs admits an FPT FO model checking, and no other hereditary class does.

2.3 Rank divisions, universal patterns and twin-width

A *division* \mathcal{D} of a matrix M is a pair $(\mathcal{D}^R, \mathcal{D}^C)$, where \mathcal{D}^R (resp. \mathcal{D}^C) is a partition of the rows (resp. columns) of M into intervals of consecutive rows (resp. columns). Each element of \mathcal{D}^R (resp. \mathcal{D}^C) is called a *row part* (resp. *column part*). A *k -division* is a division satisfying $|\mathcal{D}^R| = |\mathcal{D}^C| = k$. We often list the row (resp. column) parts of \mathcal{D}^R (resp. \mathcal{D}^C) R_1, R_2, \dots, R_k (resp. C_1, C_2, \dots, C_k) when R_i is just below R_{i+1} (resp. C_j is just to the left of C_{j+1}). For every pair $R_i \in \mathcal{D}^R, C_j \in \mathcal{D}^C$, the (contiguous) submatrix $R_i \cap C_j$ is called *cell*

or *zone* of \mathcal{D} , or more precisely, the (i, j) -cell of \mathcal{D} . Note that a k -division defines k^2 zones. We say that a cell, or more generally a matrix, is *empty* or *full 0* if all its entries are 0. The *dividing lines* of $\mathcal{D}^R = R_1, R_2, \dots$ (resp. $\mathcal{D}^C = C_1, C_2, \dots$) are the strips (of width 2) made by the last row of R_i and the first row of R_{i+1} (resp. last column of C_j and the first column of C_{j+1}). A dividing line of \mathcal{D}^R (resp. \mathcal{D}^C) *stabs* a set of rows (resp. of columns) if it intersects it. We may call *regular k -division* a k -division where every row part and column part have the same size.

A *rank- k division* of M is a k -division \mathcal{D} such that for every $R_i \in \mathcal{D}^R$ and $C_j \in \mathcal{D}^C$ the cell $R_i \cap C_j$ has at least k distinct rows or at least k distinct columns (that is, combinatorial rank at least k). By *large rank division*, we informally mean a rank- k division for arbitrarily large values of k . The maximum integer k such that M admits a rank- k division is called *grid rank*, and is denoted by $\text{gr}(M)$.

An adjacency matrix M of a binary structure encodes in any bijective fashion the *atomic type* of every pair of vertices (u, v) (i.e., the set of atomic propositions the pair (u, v) satisfies) at position (u, v) in M . We denote by $\text{Adj}_{\prec}(\mathbf{A})$ the adjacency matrix of \mathbf{A} *along* \prec , a total order on A . The *grid rank* of a binary structure \mathbf{A} , denoted by $\text{gr}(\mathbf{A})$, is the least integer k such that there is a total order \prec of A satisfying $\text{gr}(\text{Adj}_{\prec}(\mathbf{A})) = k$.

We will not need the original definition of twin-width (presented in the introduction) generalized to binary structures.⁵ So we do *not* reproduce it here. Instead we recall that the twin-width and the grid rank of a binary structure are functionally equivalent, and we encourage the reader to think of the twin-width of \mathbf{A} , $\text{tw}(\mathbf{A})$, simply as its grid rank $\text{gr}(\mathbf{A})$.

Instead we give the following useful characterization of bounded twin-width, readily generalizable to classes of other binary structures than graphs. The twin-width of the binary structure is then defined as the twin-width of the unordered matrix M , denoted by $\text{tw}(M)$. The precise value of $\text{tw}(M)$ is also defined by contraction

► **Theorem 10** ([5]). *There is a computable function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that for every binary structure \mathbf{A} , the following two implications hold:*

- *If $\text{tw}(\mathbf{A}) \leq k$, then there is a total order \prec of A such that $\text{gr}(\text{Adj}_{\prec}(\mathbf{A})) \leq f(k)$, and*
- *If there is a total order \prec of A such that $\text{gr}(\text{Adj}_{\prec}(\mathbf{A})) \leq k$, then $\text{tw}(\mathbf{A}) \leq f(k)$.*

Furthermore there are computable functions $g, h : \mathbb{N} \rightarrow \mathbb{N}$ and an algorithm running in time $h(k) \cdot |A|^{O(1)}$ which inputs an adjacency matrix $\text{Adj}_{\prec}(\mathbf{A})$ without rank- k division and outputs a $g(k)$ -sequence of \mathbf{A} .

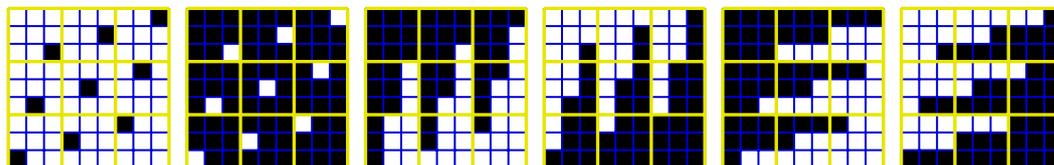
► **Theorem 11** (informal version, see [5]). *Twin-width and grid rank are effectively tied.*

It was shown in a previous paper of the series [5] that highly-structured rank divisions can always be found in large rank divisions. We now make that statement precise. Let $\mathcal{M}_k(0)$ be the $k^2 \times k^2$ permutation matrix such that if $\mathcal{M}_k(0)$ is divided in k row parts and k column parts, each of size k , there is exactly one 1 entry in each cell of the division, and this 1 entry is at position (i, j) of the (j, i) -cell; see leftmost matrix in Figure 3. For every positive integer k and $s \in \{1, \uparrow, \downarrow, \leftarrow, \rightarrow\}$, let $\mathcal{M}_k(s)$ be the $k^2 \times k^2$ 0, 1-matrix defined from $\mathcal{M}_k(0)$ by doing one of the following operations:

- switching 1 entries and 0 entries, if $s = 1$,
- turning 0 entries into 1 entries if there is a 1 entry somewhere below them, if $s = \uparrow$,
- turning 0 entries into 1 entries if there is a 1 entry somewhere above them, if $s = \downarrow$,
- turning 0 entries into 1 entries if there is a 1 entry somewhere to their right, if $s = \leftarrow$,
- turning 0 entries into 1 entries if there is a 1 entry somewhere to their left, if $s = \rightarrow$.

⁵ The definition is similar with red edges appearing between the contraction of u and v , and vertex z whenever (u, z) and (v, z) have different atomic types. We refer the curious reader to [6].

We call $\mathcal{M}_k(s)$ a *universal pattern* and $\{\mathcal{M}_k(s) : k \in \mathbb{N}\}$ a *permutation-universal family*; see Figure 3.



■ **Figure 3** The six universal patterns with $k = 3$. The black cells always represent 1 entries, and white cells, 0 entries. From left to right, $\mathcal{M}_3(0)$, $\mathcal{M}_3(1)$, $\mathcal{M}_3(\uparrow)$, $\mathcal{M}_3(\downarrow)$, $\mathcal{M}_3(\leftarrow)$, and $\mathcal{M}_3(\rightarrow)$. We always adopt the convention that the matrix entry at position $(1, 1)$ is the bottom-left one.

It was shown that, taking the adjacency matrix of a graph G along some order, either yields a matrix with bounded grid rank, and Theorem 1 effectively gives a favorable contraction sequence of G , or yields a matrix with huge grid rank, wherein a large universal pattern can be extracted:

► **Theorem 12** ([5]). *Given M an adjacency matrix of an n -vertex graph G , and an integer k , there is an $f(k)n^{O(1)}$ -time algorithm which either returns*

- $\mathcal{M}_k(s)$ as an off-diagonal submatrix of M , for some $s \in \{0, 1, \uparrow, \downarrow, \leftarrow, \rightarrow\}$,
- or a contraction sequence certifying that $\text{tw}(G) \leq g(k)$.

where f and g are computable functions.

Here, an *off-diagonal* submatrix of a square matrix is entirely contained strictly above the diagonal, or entirely contained strictly below it. In particular, its row indices and column indices are disjoint.

3 Delineation: intersection graphs of trees and paths

In this section we present a tool for showing that a class \mathcal{D} is delineated, and explore the delineation of intersection graphs of trees and paths, i.e., certain (subclasses of) chordal graphs. Our proofs of (effective) delineation will follow the same path. Either an $O(1)$ -sequence of the graph is found (bounded twin-width) or an arbitrarily large semi-induced generalized transversal pair is detected. We shall see that the latter implies monadic independence (hence, in particular, unbounded twin-width).

A *generalized transversal pair of half-graphs* consists of $3 + \ell$ sets $A = \{a_{i,j} : i, j \in [t]\}$, $B_0 = \{b_{i,j}^0 : i, j \in [t]\}, \dots, B_\ell = \{b_{i,j}^\ell : i, j \in [t]\}$, and $C = \{c_{i,j} : i, j \in [t]\}$ such that there is an edge between $a_{i,j}$ and $b_{i',j'}^k$ if and only if $(i, j) \leq_{\text{lex}} (i', j')$, for $k \in [\ell]$ there is an edge between $b_{i,j}^{k-1}$ and $b_{i',j'}^k$ if and only if $(i, j) = (i', j')$ and there is an edge between $b_{i,j}^\ell$ and $c_{i',j'}$ if and only if $(j, i) \leq_{\text{lex}} (j', i')$, where \leq_{lex} denotes the lexicographic (left-right) order. We denote this graph by $T_{t,\ell}$, and a *semi-induced* $T_{t,\ell}$ is such a graph with possibly some extra edges between two sets $X, Y \in \{A, B_0, \dots, B_\ell, C\}$ with no predefined edges. Note that $A \cup B_0$ and $B_\ell \cup C$ both induce a half-graph, but the “order” these two half-graphs put on the endpoints of the paths $(b_{i,j}^0, \dots, b_{i,j}^\ell)$ is different. We define $T_k := T_{k,0}$ and we call T_k a transversal pair (of half-graphs); see middle of Figure 2.

► **Lemma 13.** *Let ℓ be a fixed non-negative integer. Let \mathcal{C} be a hereditary class containing a semi-induced generalized transversal pair of half-graphs $T_{n,\ell}$, for every positive integer n . Then \mathcal{C} is monadically independent.*

Proof. It is folklore that the class \mathcal{M}_b of all totally ordered bipartite matchings is monadically independent (see for instance [5, 8]). By *totally ordered bipartite matching*, we mean two sets X, Y of same cardinality, with a total order over $X \cup Y$ such that X and Y are each an interval along that order, and a matching between X and Y . We shall just argue that \mathcal{M}_b transduces in \mathcal{C} . We first show the lemma when $\ell = 0$, that is, \mathcal{C} contains a semi-induced $T_{n,0} = T_n$ for every n .

Let $(G = (X, Y, E(G)), \prec)$ be any member of \mathcal{M}_b . Let $x_1 \prec x_2 \prec \dots \prec x_n$ be the elements of X , and $y_1 \prec y_2 \prec \dots \prec y_n$, the elements of Y . Finally let π be the permutation such that, for every $i \in [n]$, $x_i y_j \in E(G)$ if and only if $j = \pi(i)$.

Let (A, B, C) be the tripartition of a semi-induced T_n in \mathcal{C} . The transduction T guesses the tripartition (A, B, C) with 3 corresponding unary relations. Eventually (X, Y) will be a subset of (A, B) . We interpret a total order on $A \cup B$ by

$$x \prec y \equiv (A(x) \wedge B(y)) \vee (x \neq y \wedge A(x) \wedge A(y) \wedge \forall z (B(z) \wedge E(x, z)) \rightarrow E(y, z)) \\ \vee (x \neq y \wedge B(x) \wedge B(y) \wedge \forall z (C(z) \wedge E(x, z)) \rightarrow E(y, z)).$$

We then interpret a matching between A and B by $\varphi(x, y) \equiv A(x) \wedge B(y) \wedge E(x, y) \wedge \forall z (z \prec x \rightarrow \neg E(z, y))$. Observe that φ and \prec define a universal structure for totally ordered bipartite matchings on $2n$ vertices.

In particular, a fourth unary relation can guess the domain ($X \subseteq A, Y \subseteq B$), by picking the rows and columns of the biadjacency matrix $\text{Adj}_{\prec}(A, B, \{ab : T_n \models \varphi(a, b)\})$ corresponding to the 1 entries, which, in the regular n -division falls in the $(i, \pi(i))$ -cells with $i \in [n]$. Thus $\mathsf{T}(T_n)$ outputs (G, \prec) .

We now deal with the general case by reducing it to $\ell = 0$. For that, we transduce a semi-induced T_n in a semi-induced $T_{n,\ell}$. The transduction is imply based on the definition of generalized transversal pairs. It uses $3 + \ell$ unary relations A, B_0, \dots, B_ℓ, C , redefines the domain as $A \cup B_0 \cup C$, keeps the edges between A and B_0 , and adds an edge between $x \in B_0$ and $y \in C$ if and only if there is a path from x to y going through B_1, B_2, \dots, B_ℓ , in this order. All of this is easily expressible in first-order logic. ◀

From Lemma 13, one can easily deduce the following.

► **Lemma 14.** *Let ℓ be a fixed non-negative integer. Let $f : \mathbb{N} \rightarrow \mathbb{N}$ be any computable function, and \mathcal{C} be a graph class. If for every natural k and $G \in \mathcal{C}$, either G admits an $f(k)$ -sequence or G has a semi-induced generalized transversal pair $T_{k,\ell}$, then \mathcal{C} is delineated.*

Furthermore, if the contraction sequence can be found in time $g(k) \cdot |V(G)|^{O(1)}$ for some computable function g , then \mathcal{C} is effectively delineated.

By Theorem 1, the $f(k)$ -sequence of G in Lemma 14 can be replaced by an adjacency matrix of G of grid rank at most $f(k)$.

Showing that a class \mathcal{D} is effectively delineated establishes that, as far as efficient (that is, FPT) FO model checking is concerned, twin-width gives a complete picture of what happens on \mathcal{D} . Indeed it is unlikely that a monadically independent class admits an FPT algorithm for FO model checking (see Section 2.2). Trivially, every class with bounded twin-width is delineated, and every class where $O(1)$ -sequences can be found in polynomial time (see [4]) is effectively delineated. We now list some non-trivial examples of (effectively) delineated classes.

► **Theorem 15** ([5, 6, 21] + this paper). *The following classes of binary structures are effectively delineated: permutation graphs [6], and even, circle graphs [21], ordered graphs [5], interval graphs, and even, rooted directed path graphs.*

The proof of Theorem 15 relies on finding a good vertex ordering \prec for interval graphs or rooted directed path graphs so that for any graph G which is an interval or a rooted directed path graph, $\text{Adj}_\prec(G)$ already has small grid rank or G contains a semi-induced generalized transversal pair $T_{k,\ell}$. Then Lemma 14 is applicable, especially for the last two classes, thus implies Theorem 15.

On the contrary, the class of subcubic graphs is *not* delineated. Indeed the whole class is monadically dependent (see for instance [26]), even monadically stable, but has unbounded twin-width [4]. We will see that the classes of segment graphs (even with some further restrictions) and visibility graphs of simple polygons are also *not* delineated. In some sense, what we do is to reduce to the easy case of subcubic graphs.

It is known [4, 8] that classes of bounded twin-width have exponential growth. Thus by the contrapositive, classes of super-exponential growth, like the following ones, have unbounded twin-width.

► **Theorem 16** ([4]). *The following classes have unbounded twin-width:*

- *the class $\mathcal{G}_{\leq 3}$ of every subcubic graph;*
- *the class $\mathcal{B}_{\leq 3}$ of every bipartite subcubic graph;*
- *the 2-subdivision of every biclique $K_{n,n}$.*

► **Lemma 17.** *If \mathcal{C} admits a subclass which is transduction equivalent to $\mathcal{G}_{\leq 3}$ or to $\mathcal{B}_{\leq 3}$, then \mathcal{C} is not delineated.*

In what follows, we sketch the key ideas for settling the last piece toward Theorem 15, stated below.

► **Proposition 18.** *There exist a computable function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that the following holds. For any interval graph, or rooted directed path graph G , there exists a vertex ordering \prec on $V(G)$ such that for every natural k , either $\text{Adj}_\prec(G)$ had grid rank at most $f(k)$ or G has a semi-induced generalized transversal pair $T_{k,\ell}$ for some ℓ .*

The class of interval graphs is delineated, proof idea. Let G be an interval graph and $\mathcal{I}_G = \{I_v : v \in V(G)\}$ be an interval representation of G , where the interval I_v is of the form $[\ell_v, r_v]$ for some integers $1 \leq \ell_v \leq r_v$. We further assume some minimality on the representation \mathcal{I} , i.e., if $\ell_u < \ell_w$ for vertices $u, w \in V(G)$, there exists a vertex $v \in V(G)$ such that $\ell_u \leq r_v < \ell_w$.

Let \mathcal{C} be a hereditary class of interval graphs of unbounded twin-width. For each graph $G \in \mathcal{C}$ with an interval representation \mathcal{I} , we associate a total order \prec , following a lexicographic order on \mathcal{I} . For any integer t , and any interval graph $G \in \mathcal{C}$ of sufficiently large twin-width we use a large rank division of the adjacency matrix $\text{Adj}_\prec(G)$ to find a semi-induced transversal pair T_t and obtain delineation of interval graphs by Lemma 13. To find T_t using the rank division we extract two groups $\{A_1, \dots, A_{f(t)}\}$ and $\{B_1, \dots, B_{f(t)}\}$ of vertex disjoint blocks from the rank division of $A_\prec(G)$ such that $A_1 \prec \dots \prec A_{f(t)} \prec B_1 \prec \dots \prec B_{f(t)}$. We can now assign an interval I_i to each block A_i and an interval J_i to each block B_i containing all respective start points. After some cleaning we can assume that these intervals are disjoint. By picking out appropriate selections a_1, \dots, a_{t^2} , $a_i \in A_i$ and b_1, \dots, b_{t^2} , $b_i \in B_i$ we can force the a_i 's and b_i 's to form a half-graph which induce any order we wish on the a_i 's using the large rank of each cell. Furthermore, using the minimality assumption, and the order and disjointness of intervals I_i we can find c_1, \dots, c_ℓ forming a half-graph with the a_i 's inducing the natural order on the a_i 's. Appropriate selections of a_i, b_i and c_i will therefore yield a transversal pair.

The class of rooted directed paths is delineated, proof idea. *Directed path graphs* are the intersection graphs of directed paths of an oriented tree. In other words, there is a collection $\{P_v : v \in V(G)\}$ consisting directed paths of an oriented tree T such that $(u, w) \in E(G)$ if and only if $V(P_u) \cap V(P_w) \neq \emptyset$. If T in a tree model of G is an out-tree, we say that G is a *rooted directed path graph*. For $v \in V(G)$ we denote by $\text{high}(v)$ and $\text{low}(v)$ the nodes of P_v that are closest and furthest from the root, respectively. Notice they are not necessarily distinct. We extend this notation to sets of vertices by defining $\text{high}(X) = \{\text{high}(v) : v \in X\}$ and $\text{low}(X) = \{\text{low}(v) : v \in X\}$ for $X \subseteq V(G)$.

Since interval graphs can be visualized as the intersection graph of subpath of a directed path, they form a subclass of rooted directed path graphs.

In general, directed path graphs are not delineated. This can be observed by subdividing the edges of a subcubic bipartite graph G and making a clique of the newly added vertices to generate a directed path graph G' that encodes G , and then applying Lemma 17 to the family of all directed path graphs constructed this way. Since this class is chordal, this also implies that chordal graphs, and even split graphs, are not delineated. On the positive side, we show that rooted directed path graphs are delineated.

We start by extracting the vertices of a rooted directed path graph G which consist of two collections associated with row and column parts of a rank- $f(t)$ division of $\text{Adj}_{\prec}(G)$ as an off-diagonal submatrix: we may assume $\mathcal{A} = \{A_i : i \in [f(t)/2]\}$ be the first $f(t)/2$ parts of the row division and $\mathcal{B} = \{B_i : i \in [f(t)/2]\}$ to be the last $f(t)/2$ parts of the column division. Then, for each A_i and B_i we take vertices a_i, b_i to represent the sets, respectively, define A° to contain all a_i and B° to contain all b_i . The goal is to use A° and B° to distinguish between two cases in the proof.

We first observe that there is a directed path P of T containing all $\text{high}(u)$ where u is a vertex defining some adjacency between sets of \mathcal{A} and \mathcal{B} and that P defines an order $<_P$ on both A° and B° . We denote by $p(u)$ the node in $V(P_u) \cap V(P)$ that is closer to $\text{low}(u)$ and say that $u \leq_P v$ if and only if $p(u) \leq_T p(v)$. From this point, we prove a series of claims to show that, to organize large parts of \mathcal{A} and \mathcal{B} in a desirable way, we can focus on organizing large parts of A° and B° .

The easier case is when both A° and B° contain sufficiently large strictly increasing chains with respect to $<_P$. Since $<_P$ may not agree with \prec , we apply the Erdős-Szekeres theorem to extract a large monotone sequence of both chains, and keep only the vertices appearing in those sequences in A° and B° . We then use those sequences to define, for each A_i associated with a vertex in the new A° , an exclusive subpath I_i of P that contains $p(a)$ for every $a \in A_i$, and do the same for each B_i . This is done by observing that no $p(a)$ can be “very far away” from $p(a_i)$ with respect to the monotone sequence. With these subpaths, we construct an interval graph and then solve this case as in the proof of delineation for this class.

If only A° , for instance, contains a large strictly increasing chain with respect to $<_P$ then there must be a node $p \in P$ on which a large subset of $\{p(b_i) : b_i \in B^\circ\}$ is concentrated. Although we can, to some extent, predict the behavior of the paths associated with vertices in B° after they leave P through p , we cannot use a minimality assumption on the tree model to find in G vertices distinguishing each of the parts of \mathcal{B} associated with vertices of B° . This is the crucial difference that makes finding a semi-induced $T_{t,2}$ in this configuration much harder than in the first one.

4 Win-wins via twin-width: segment graphs and visibility graphs

A graph parameter p is said *FO definable* if there is a function that inputs a positive integer k and outputs a first-order sentence φ_k such that for every graph G , $p(G) = k$ if and only if $G \models \varphi_k$. It is further *effectively FO definable* if an algorithm realizes that function and takes time $f(k)$ for some computable function f .

We say that a parameter q is *p -bounded on class \mathcal{C}* , denoted by $q \sqsubseteq p$ on \mathcal{C} or $q \sqsubseteq^{\mathcal{C}} p$, if there is a non-decreasing function f such that for every graph $G \in \mathcal{C}$, $q(G) \leq f(p(G))$. We say that twin-width is *effectively p -bounded on \mathcal{C}* , denoted by $\text{tw} \sqsubseteq_{\text{eff}} p$ on \mathcal{C} or $\text{tw} \sqsubseteq_{\text{eff}}^{\mathcal{C}} p$, if further there is an algorithm that outputs a $g(p(G))$ -sequence for every graph $G \in \mathcal{C}$ in time $h(p(G)) \cdot |V(G)|^{O(1)}$ for some computable functions g, h .

The following reduces the task of showing that an FO definable parameter p is FPT on \mathcal{C} to showing that $\text{tw} \sqsubseteq_{\text{eff}}^{\mathcal{C}} p$ holds.

► **Theorem 19.** *Let p be an effectively FO definable parameter, and \mathcal{C} a class such that $\text{tw} \sqsubseteq_{\text{eff}}^{\mathcal{C}} p$. Then $p(G) \geq k$ for $G \in \mathcal{C}$ can be decided in FPT time $f(p(G)) \cdot |V(G)|^{O(1)}$ for some computable function f .*

4.1 Segment graphs

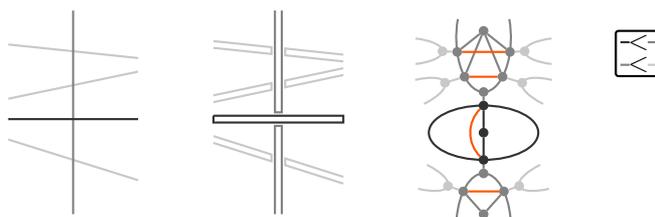
Pure (hence triangle-free) axis-parallel unit segment graphs were shown to have unbounded twin-width [4], by constructing a family of such graphs with super-exponential growth. This family contains arbitrarily large bicliques (see [4, Figure 4]). We will show that bicliques are necessary to make the twin-width large, even when we lift the requirements that the segments are axis-parallel and unit.

Techniques to show Theorem 5. We first FO transduce $K_{t,t}$ -free segment graphs from a class \mathcal{F} of 2-edge-colored graphs that, we next show, has bounded twin-width. Once this is done, it follows from Theorem 8 that $K_{t,t}$ -free segment graphs have twin-width at most $h(t)$ and $h(t)$ -sequence can be obtained from Theorem 8.

To capture a suitable graph class \mathcal{F} , imagine a representation of $K_{t,t}$ -free segment graph G which uses thin rectangles in place of segments. Using bounded degeneracy of $K_{t,t}$ -free segment graphs [24], each rectangle can fragment into at most $d + 1$ sub-rectangles at places *stabbed* by a rectangle preceding it in the d -degeneracy ordering. This fragmented representation preserves the adjacency of G in the form of local adjacency between sub-rectangles, and the former can be restored by FO transduction from the latter. It can be naturally translated to a superposition of two graphs over the same vertex set, namely a plane graph $P = (V, E)$ and a matching graph $H = (V, M)$ which is *nicely aligned* with respect to a packing \mathcal{C} of facial cycles partitioning $V(P)$, see Figure 4.

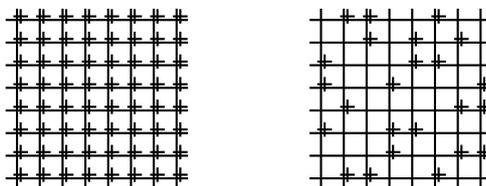
Our task then boils down to showing that a plane graph P admits a vertex ordering \prec which circularly orders each the facial cycle of \mathcal{C} so that (P, \prec) has bounded twin-width. It turns out that a variant of BFS discovery order works, with two important features: The edges are considered in the cyclic order during the exploration phase of a vertex v , and the vertices of a facial cycle $C \in \mathcal{C}$ are processed in batch, when ordered in the cyclic order around the face.

In the previous theorem, one cannot relax the $K_{t,t}$ -freeness assumption to H_t -freeness. Let B_n be the graph obtained from the 2-subdivision of a biclique $K_{n,n}$ by adding back the edges of the original biclique. The left part of Figure 5 shows that, for every $n \in \mathbb{N}$, the graph B_n is realizable with axis-parallel segments of two different lengths. Note however that B_n has no semi-induced H_4 , and that $\lim_{n \rightarrow \infty} \text{tw}(B_n) = \infty$.



■ **Figure 4** Left: Original segment representation. Center: Thin rectangle stabbed by its predecessor in the degeneracy ordering. Right: Plane graph with a nicely aligned matching, preserving the information of the original segment graph.

To establish the latter claim, one can for instance “remove” the edges of the biclique by means of an FO transduction, and invoke Theorem 8 and the third item of Theorem 16. The transduction first marks the long horizontal segments by unary relation U_1 (color 1), and the long vertical segments, by unary relation U_2 (color 2), and interpret the new edges as $\varphi(x, y) \equiv E(x, y) \wedge \neg(U_1(x) \wedge U_2(y)) \wedge \neg(U_1(y) \wedge U_2(x))$.



■ **Figure 5** Left: An axis-parallel H_4 -free two-lengthed segment graph realizing B_n (here drawn with $n = 8$), whose twin-width grows with n . Right: Axis-parallel segment graphs are transduction equivalent to $\mathcal{B}_{\leq 3}$, thus not delineated.

Similarly the 2-subdivision of any subcubic bipartite graph, augmented with the biclique between its two partite sets, is realizable with axis-parallel segments of two different lengths (see right-hand side of Figure 5). Indeed those graphs – let us denote by \mathcal{C} the class they form – are induced subgraphs of some B_n . We claim (see right of Figure 5 and long version, for a proof) that \mathcal{C} and $\mathcal{B}_{\leq 3}$ are transduction equivalent, and by Lemma 17, two-lengthed axis-parallel segments are not delineated.

In the construction of Figure 5, we use two different lengths for the segments. We show that with a unique length (unit segments), axis-parallel H_t -freeness implies bounded twin-width.

▶ **Theorem 20.** *Axis-parallel H_t -free unit segment graphs have bounded twin-width.*

Actually we prove a stronger statement than the previous theorem, where segments are not necessarily unit, but the ratio between the largest and the smallest lengths is bounded. Again it shows that the fact that this ratio is unbounded in Figure 5 (left) is unavoidable.

Techniques to show Theorem 20. We face again the challenging task of finding a “good” linear order on objects from a two-dimensional space. We place a virtual grid whose cells are of size 1×1 , and cut the segments along this grid, adding some junction vertices in between the cut pieces corresponding to the same segment. We first prove by FO transduction that if this new graph has bounded twin-width, then the original segment graph has bounded twin-width. For the newly built graph, a natural order consists of locally enumerating the

segments counter-clockwise according to where they cross the grid, and globally enumerating the cells of the grid row by row. Note that the dimension of the grid cells imposes that every segment crosses the grid.

The crux is then to argue that the circular order along the boundary of a cell yields adjacency matrices with bounded grid rank. Somewhat surprisingly this part leverages the same argument as we will later use for H_t -free visibility graphs of terrains; a forbidden pattern like the Order Claim.

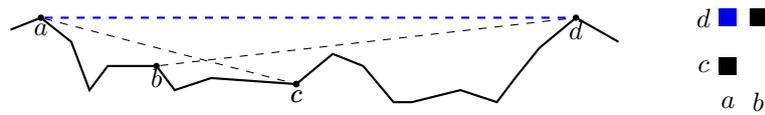
4.2 Visibility graphs

We first show that visibility graphs of terrains without arbitrarily large ladders have bounded twin-width.

► **Theorem 21.** *H_t -free visibility graphs of 1.5D terrains have bounded twin-width.*

Rather naturally, we choose the order \prec along the boundary of the terrain. Due to the Order Claim (see Lemma 22 and Figure 6) the obtained adjacency matrices exclude a pattern (right of Figure 6) that, combined with H_t -freeness, prevents large universal patterns. Hence we conclude by Theorem 12.

► **Lemma 22** (Order Claim [2]). *If $a \prec b \prec c \prec d$, a see c , and b see d , then a and d also see each other.*



■ **Figure 6** Left: The Order Claim. The dashed black edges imply the dashed blue edge. Right: In the thus ordered adjacency matrix, the 1 entries at (a, c) and (b, d) implies the 1 entry at (a, d) .

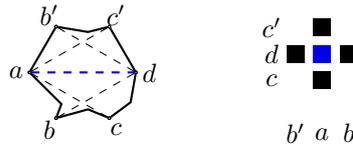
In stark contrast, we can exhibit a subclass of visibility graphs of simple polygons whose hereditary closure has unbounded twin-width but is monadically dependent, and even monadically stable. This transduction is more involved than the previous ones, so we give full details (in the long version). Finally, we show that the twin-width of simple polygons is bounded by a function of their independence number α .

► **Theorem 23.** *Twin-width is α -bounded in visibility graphs of simple polygons, and effectively α -bounded if a geometric representation is given.*

Proof (Sketch). Let \mathcal{P} be a simple polygon, and G its visibility graph. We identify a vertex of G with its corresponding geometric vertex of \mathcal{P} . Let \prec be the total order whose successor relation is a Hamiltonian path of the boundary of \mathcal{P} . Visibility graphs of simple polygons satisfy the double-X property: If $b' \prec a \prec b \prec c \prec d \prec c'$, and ac, bd, ac', db' are all in $E(G)$, then ad is also an edge of G (see Figure 7).

This excludes that the complement of a(n arbitrary) permutation is realized by the adjacency matrix of G ordered along \prec . That is, the second universal pattern in Figure 3 would not appear in $\text{Adj}_{\prec}(G)$.

We now upperbound the size of a universal pattern in $\text{Adj}_{\prec}(G)$ (among the other five patterns) in terms of $\alpha(G)$, and conclude by Theorem 12. We will actually not need the universal pattern in its whole, but simply a decreasing subsequence of it. (This is made



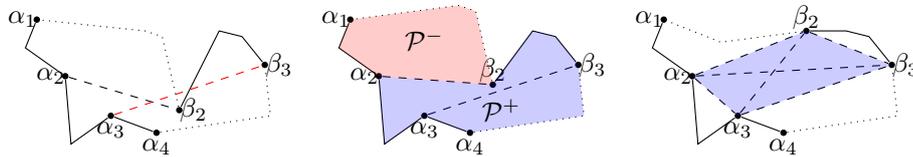
■ **Figure 7** Left: The double-X property. Right: What it implies in the adjacency matrix ordered along the boundary of the polygon; the four 1 entries in black force the central one in blue.

formal in the next paragraph, where we extract a large anti-diagonal induced matching or half-graph.) This is convenient since we can thus apply Ramsey’s theorem while keeping the “complexity” of the initial structure.

Let $p = \text{Ram}(\text{Ram}(4, \alpha(G)), \alpha(G))$, where $\text{Ram}(s, t)$ is the function of Ramsey’s theorem which enforces a monochromatic clique on s or t vertices in a 2-edge-colored complete graph on $\text{Ram}(s, t)$ vertices. Note that if the twin-width of G is larger than a certain function of p , we can find in each of the five allowed universal patterns $2p$ vertices of G : $a_1 \prec a_2 \prec \dots \prec a_{p-1} \prec a_p \prec b_p \prec b_{p-1} \prec \dots \prec b_2 \prec b_1$ such that $a_i b_j \in E(G)$ if and only if $i = j$ (resp. $i \leq j$, resp. $i \geq j$). We denote $\{a_1, \dots, a_p\}$ (resp. $\{b_1, \dots, b_p\}$) by A (resp. B). We now work toward finding a contradiction.

Let $A' \subseteq A$ induce a clique in G with $|A'| = \text{Ram}(\alpha(G), 4)$. Let B' be the vertices of B with the same index as a vertex of A' , and let $B'' \subseteq B'$ induce a clique in G of size 4. Finally let A'' be the vertices in A' (or A for that matter) with same index as a vertex in B'' . We relabel the eight vertices of $A'' \cup B''$ by $\alpha_1 \prec \alpha_2 \prec \alpha_3 \prec \alpha_4 \prec \beta_4 \prec \beta_3 \prec \beta_2 \prec \beta_1$.

First observe that, since they form a clique, $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ are in convex position. For $\alpha_2 \beta_2$ and $\alpha_3 \beta_3$ to be in $E(G)$, the vertices β_2 and β_3 have to be in the convex (possibly infinite) region delimited by the line segment $\alpha_2 \alpha_3$, the ray starting at α_2 and passing through α_1 , and the ray starting at α_3 and passing through α_4 . Since β_2 comes after β_3 in the boundary order, the quadrangle $\alpha_2 \alpha_3 \beta_3 \beta_2$ has to be non self-intersecting (otherwise $\alpha_2 \beta_2$ and $\alpha_3 \beta_3$ cannot both be edges, see left of Figure 8). We now claim that $\alpha_2 \alpha_3 \beta_3 \beta_2$ is a convex quadrangle. Assume for the sake of contradiction that β_2 is in the interior of the triangle $\alpha_2 \alpha_3 \beta_3$ (this is without loss of generality). As $\alpha_2 \beta_2$ is an edge of G , the line segment $\alpha_2 \beta_2$ cuts \mathcal{P} into two simple polygons: \mathcal{P}^- containing α_1 , and \mathcal{P}^+ containing α_3 . Observe that no line segment starting at β_3 and fully contained in \mathcal{P} can intersect $\mathcal{P}^- \setminus \{\beta_2\}$. Indeed, since β_2 is in the interior of $\alpha_2 \alpha_3 \beta_3$, the ray starting at β_3 and passing through β_2 remains entirely within \mathcal{P}^+ . However α_1 is in \mathcal{P}^- . Therefore β_3 and β_1 cannot see each other; a contradiction (see middle of Figure 8).



■ **Figure 8** Left: If $\alpha_2 \alpha_3 \beta_3 \beta_2$ is self-intersecting, at least one edge of $\alpha_2 \beta_2, \alpha_3 \beta_3$ (here $\alpha_3 \beta_3$) is missing from G . Center: If β_2 lies in the interior of $\alpha_2 \alpha_3 \beta_3$, vertices β_1 (in \mathcal{P}^+) and β_3 cannot see each other without blocking the edge $\alpha_2 \beta_2$. Right: If $\alpha_2 \alpha_3 \beta_3 \beta_2$ is in convex position (in this order), then both $\alpha_2 \beta_3$ and $\alpha_3 \beta_2$ are edges; another contradiction.

Since the four sides of the convex, non self-intersecting quadrangle $\alpha_2 \alpha_3 \beta_3 \beta_2$ are edges of G , the two diagonals $\alpha_2 \beta_3$ and $\alpha_3 \beta_2$ are also edges (since \mathcal{P} cannot intersect the interior of $\alpha_2 \alpha_3 \beta_3 \beta_2$, see right of Figure 8); a contradiction to the induced matching or half-graph in between A and B . ◀

Combined with the FO model checking algorithm in [6], generalizes a conjecture of Hliněný, Pokrývka, and Roy [22], and shows in particular that k -INDEPENDENT SET is FPT on visibility graphs of simple polygons.

References

- 1 John T. Baldwin and Saharon Shelah. Second-order quantifiers and the complexity of theories. *Notre Dame Journal of Formal Logic*, 26(3):229–303, 1985.
- 2 Boaz Ben-Moshe, Matthew J. Katz, and Joseph S. B. Mitchell. A constant-factor approximation algorithm for optimal 1.5D terrain guarding. *SIAM J. Comput.*, 36(6):1631–1647, 2007. doi:10.1137/S0097539704446384.
- 3 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is NP-complete. *CoRR*, abs/2112.08953, 2021. arXiv:2112.08953.
- 4 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1977–1996, 2021. doi:10.1137/1.9781611976465.118.
- 5 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Toruńczyk. Twin-width IV: ordered graphs and matrices. *CoRR*, abs/2102.03117, Accepted at STOC 2022. arXiv:2102.03117.
- 6 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: Tractable FO Model Checking. *J. ACM*, 69(1):1–46, November 2022. doi:10.1145/3486655.
- 7 Édouard Bonnet and Tillmann Miltzow. Parameterized hardness of art gallery problems. *ACM Trans. Algorithms*, 16(4):42:1–42:23, 2020. doi:10.1145/3398684.
- 8 Édouard Bonnet, Jaroslav Nesetril, Patrice Ossona de Mendez, Sebastian Siebertz, and Stéphan Thomassé. Twin-width and permutations. *CoRR*, abs/2102.06880, 2021. arXiv:2102.06880.
- 9 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.
- 10 Marek Cygan, Fedor V Fomin, Lukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4. Springer, 2015.
- 11 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 12 Rodney G. Downey, Michael R. Fellows, and Udayan Taylor. The parameterized complexity of relational database queries and an improved characterization of W[1]. In Douglas S. Bridges, Cristian S. Calude, Jeremy Gibbons, Steve Reeves, and Ian H. Witten, editors, *First Conference of the Centre for Discrete Mathematics and Theoretical Computer Science, DMTCS 1996, Auckland, New Zealand, December, 9-13, 1996*, pages 194–213. Springer-Verlag, Singapore, 1996.
- 13 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.
- 14 Zdenek Dvorák and Sergey Norin. Strongly sublinear separators and polynomial expansion. *SIAM J. Discrete Math.*, 30(2):1095–1101, 2016. doi:10.1137/15M1017569.
- 15 Kord Eickmeyer, Jan van den Heuvel, Ken-ichi Kawarabayashi, Stephan Kreutzer, Patrice Ossona de Mendez, Michal Pilipczuk, Daniel A. Quiroz, Roman Rabinovich, and Sebastian Siebertz. Model-checking on ordered structures. *ACM Trans. Comput. Log.*, 21(2):11:1–11:28, 2020. doi:10.1145/3360011.
- 16 Stephan J. Eidenbenz. Inapproximability of finding maximum hidden sets on polygons and terrains. *Comput. Geom.*, 21(3):139–153, 2002. doi:10.1016/S0925-7721(01)00029-3.
- 17 Fedor V. Fomin, Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Bidimensionality. In *Encyclopedia of Algorithms*, pages 203–207. Springer, 2016. doi:10.1007/978-1-4939-2864-4_47.

- 18 Jakub Gajarský, Petr Hliněný, Jan Obdržálek, Daniel Lokshtanov, and M. S. Ramanujan. A new perspective on FO model checking of dense graph classes. *ACM Trans. Comput. Log.*, 21(4):28:1–28:23, 2020. doi:10.1145/3383206.
- 19 Jakub Gajarský, Stephan Kreutzer, Jaroslav Nesetril, Patrice Ossona de Mendez, Michal Pilipczuk, Sebastian Siebertz, and Szymon Torunczyk. First-order interpretations of bounded expansion classes. *ACM Trans. Comput. Log.*, 21(4):29:1–29:41, 2020. doi:10.1145/3382093.
- 20 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.
- 21 Petr Hliněný and Filip Pokrývka, 2022. Personal communication.
- 22 Petr Hliněný, Filip Pokrývka, and Bodhayan Roy. FO model checking on geometric graphs. *Computational Geometry*, 78:1–19, 2019.
- 23 Stephan Kreutzer and Anuj Dawar. Parameterized complexity of first-order logic. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:131, 2009. URL: <http://ecc.eccc.hpi-web.de/report/2009/131>, arXiv:TR09-131.
- 24 James R. Lee. Separators in region intersection graphs. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, volume 67 of *LIPICs*, pages 1:1–1:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ITCS.2017.1.
- 25 Bingkai Lin. The parameterized complexity of the k -biclique problem. *J. ACM*, 65(5):34:1–34:23, 2018. doi:10.1145/3212622.
- 26 Klaus-Peter Podewski and Martin Ziegler. Stable graphs. *Fundamenta Mathematicae*, 100(2):101–107, 1978.
- 27 Thomas C. Shermer. Hiding people in polygons. *Computing*, 42(2-3):109–131, 1989. doi:10.1007/BF02239742.

Obstructions to Faster Diameter Computation: Asteroidal Sets

Guillaume Ducoffe  

National Institute of Research and Development in Informatics, Bucharest, Romania
University of Bucharest, Romania

Abstract

An *extremity* is a vertex such that the removal of its closed neighbourhood does not increase the number of connected components. Let Ext_α be the class of all connected graphs whose *quotient graph* obtained from modular decomposition contains no more than α pairwise nonadjacent extremities. Our main contributions are as follows. First, we prove that the diameter of every m -edge graph in Ext_α can be computed in deterministic $\mathcal{O}(\alpha^3 m^{3/2})$ time. We then improve the runtime to $\mathcal{O}(\alpha^3 m)$ for bipartite graphs, to $\mathcal{O}(\alpha^5 m)$ for triangle-free graphs, $\mathcal{O}(\alpha^3 \Delta m)$ for graphs with maximum degree Δ , and more generally to linear for all graphs with bounded clique-number. Furthermore, we can compute an additive $+1$ -approximation of all vertex eccentricities in deterministic $\mathcal{O}(\alpha^2 m)$ time. This is in sharp contrast with general m -edge graphs for which, under the Strong Exponential Time Hypothesis (SETH), one cannot compute the diameter in $\mathcal{O}(m^{2-\epsilon})$ time for any $\epsilon > 0$.

As important special cases of our main result, we derive an $\mathcal{O}(m^{3/2})$ -time algorithm for exact diameter computation within *dominating pair* graphs of diameter at least six, and an $\mathcal{O}(k^3 m^{3/2})$ -time algorithm for this problem on graphs of *asteroidal number* at most k . Both results extend prior works on exact and approximate diameter computation within AT-free graphs. To the best of our knowledge, this is also the first deterministic subquadratic-time algorithm for computing the diameter within the subclasses of: chordal graphs of bounded leafage (generalizing the interval graphs), k -moplex graphs and k -polygon graphs (generalizing the permutation graphs) for any fixed k . We end up presenting an improved algorithm for chordal graphs of bounded asteroidal number, and a partial extension of our results to the larger class of all graphs with a *dominating target* of bounded cardinality. Our time upper bounds in the paper are shown to be essentially optimal under plausible complexity assumptions.

Our approach is purely combinatorial, that differs from most prior recent works in this area which have relied on geometric primitives such as Voronoi diagrams or range queries. On our way, we uncover interesting connections between the diameter problem, Lexicographic Breadth-First Search, graph extremities and the asteroidal number.

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

Keywords and phrases Diameter computation, Asteroidal number, LexBFS

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.10

Related Version *Full Version*: <https://arxiv.org/abs/2209.12438>

Funding This work was supported by project PN-19-37-04-01 “New solutions for complex problems in current ICT research fields based on modelling and optimization”, funded by the Romanian Core Program of the Ministry of Research and Innovation (MCI) 2019-2022. This work was also supported by a grant of the Ministry of Research, Innovation and Digitalization, CCCDI - UEFISCDI, project number PN-III-P2-2.1-PED-2021-2142, within PNCDI III.

1 Introduction

For any undefined graph terminology, see [9]. All graphs considered in this paper are undirected, simple (i.e., without loops nor multiple edges) and connected, unless stated otherwise. Given a graph $G = (V, E)$, let $n = |V|$ and $m = |E|$. For every vertices



© Guillaume Ducoffe;

licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 10; pp. 10:1–10:24

Leibniz International Proceedings in Informatics



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

u and v , let $d_G(u, v)$ be their distance (minimum number of edges on a uv -path) in G . Let $e_G(u)$ denote the eccentricity of vertex u (maximum distance to any other vertex). We sometimes omit the subscript if the graph G is clear from the context. Finally, let $\text{diam}(G) = \max_{u, v \in V} d(u, v) = \max_{u \in V} e(u)$ be the diameter of G .

Computing the diameter is important in a variety of network applications such as in order to estimate the maximum latency in communication systems [32] or to identify the peripheral nodes in some complex networks with a core/periphery structure [60]. On n -vertex m -edge graphs, this can be done in $\mathcal{O}(nm)$ time by running a BFS from every vertex. This runtime is quadratic in the number m of edges, even for sparse graphs (with $m \leq c \cdot n$ edges for some c), and therefore it is too prohibitive for large graphs with millions of vertices and sometimes billions of edges. Using Seidel’s algorithm [71], the diameter of any n -vertex graph can be also computed in $\mathcal{O}(n^{\omega+o(1)})$ time, with ω the square matrix multiplication exponent. If $\omega = 2$, then this is almost linear-time for dense graphs, with $m \geq c \cdot n^2$ edges for some constant c (currently, it is only known that $\omega < 2.37286$ [2]). However, for sparse graphs, this is still in $\Omega(m^2)$. In [69], Roditty and Vassilevska Williams proved that assuming the Strong Exponential-Time Hypothesis of Impagliazzo, Paturi and Zane [57], the diameter of n -vertex graphs with $n^{1+o(1)}$ edges *cannot* be computed in $\mathcal{O}(n^{2-\epsilon})$ time, for any $\epsilon > 0$. Therefore, breaking the quadratic barrier for diameter computation is likely to require additional graph structure, even for sparse graphs. In this paper, we make progress in this direction.

Let us call an algorithm truly subquadratic if it runs in $\mathcal{O}(m^{2-\epsilon})$ time on m -edge graphs, for some fixed positive ϵ . Over the last decades, the existence of a truly subquadratic (often linear-time) algorithm for the diameter problem was proved for many important graph classes [1, 12, 14, 16, 18, 23, 29, 33, 34, 36, 37, 42, 38, 39, 40, 41, 44, 45, 49, 66]. This has culminated in some interesting connections between faster diameter computation algorithms and Computational Geometry, e.g., see the use of Voronoi diagrams for computing the diameter of planar graphs [16, 49], and of data structures for range queries in order to compute all eccentricities within bounded treewidth graphs [1, 14, 17], bounded clique-width graphs [41], or even proper minor-closed graph classes [44]. However, this type of geometric approach usually works only if certain Helly-type properties hold for the graph classes considered [3, 4, 11, 22, 39, 43, 44]. Beyond that, the finer-grained complexity of the diameter problem is much less understood, with only a few graph classes for which truly subquadratic algorithms are known [10]. The premise of this paper is that the *asteroidal number* could help in finding several new positive cases for diameter computation.

Related work

Recall that an independent set in a graph G is a set of pairwise non-adjacent vertices. An asteroidal set in a graph G is an independent set A with the additional property that, for every vertex $a \in A$, there exists a path between any two remaining vertices of $A \setminus \{a\}$ that does not contain a nor any of its neighbours in G . Let the asteroidal number of G be the largest cardinality of its asteroidal sets. The graphs of asteroidal number at most two are sometimes called *AT-free graphs*, and they generalize interval graphs, permutation graphs and co-comparability graphs amongst other subclasses [27]. It is worth mentioning here that all the aforementioned subclasses have unbounded treewidth and clique-width. The properties of AT-free graphs have been thoroughly studied in the literature [5, 8, 13, 15, 24, 26, 27, 46, 50, 51, 53, 56, 61, 62, 72], and some of these properties were generalized to the graphs of bounded asteroidal number [28, 58, 59]. In particular, as far as we are concerned here, there is a simple linear-time algorithm for computing a vertex in any AT-free graph whose eccentricity is within one of the diameter [23]. However, it

has been only recently that a truly subquadratic algorithm for *exact* diameter computation within this class was presented [42]. This algorithm runs in deterministic $\mathcal{O}(m^{3/2})$ time on m -edge AT-free graphs, and it is combinatorial – that means, roughly, it does not rely on fast matrix multiplication algorithms or other algebraic techniques. In fact both algorithms from [23] and [42] are based on specific properties of *LexBFS orderings* for the AT-free graphs¹. Roughly, the algorithm from [42] starts computing a dominating shortest path. In doing so, the search for a diametral vertex can be restricted to the closed neighbourhood of any one end of this path. However, in general this neighbourhood might be very large. The key procedure of the algorithm consists in further pruning out the neighbourhood so that it reduces to a clique. Then, we are done executing a BFS from every vertex in this clique. Both the computation of a dominating shortest path and the pruning procedure of the algorithm are taking advantage of the existence of a linear structure for AT-free graphs, that can be efficiently uncovered by using a double-sweep LexBFS [24]. Unfortunately, this linear structure no more exists for graphs of asteroidal number ≥ 3 .

Contributions

The structure of graphs of bounded asteroidal number, and its relation to LexBFS, is much less understood than for AT-free graphs. Therefore, extending the known results for the diameter problem on AT-free graphs to the more general case of graphs of bounded asteroidal number is quite challenging. Doing just that is our main contribution in the paper. In fact, we prove even more strongly that only some types of large asteroidal sets need to be excluded in order to obtain a faster diameter computation algorithm.

More specifically, an *extremity* is a vertex such that the removal of its closed neighbourhood leaves the graph connected, see [59, 63]. Note that every subset of pairwise nonadjacent extremities forms an asteroidal set. A module in a graph $G = (V, E)$ is a subset of vertices X such that every vertex of $V \setminus X$ is either adjacent to every of X or nonadjacent to every of X . It is a strong module if it does not overlap any other module of G . Finally, the *quotient graph* of G is the induced subgraph obtained by keeping one vertex in every inclusionwise maximal strict subset of V which is a strong module of G (see also Sec. 2 for a more detailed discussion about the modular decomposition of a graph). It is known that except in a few degenerate cases, the diameter of G always equals the diameter of its quotient graph [29]. **We are interested in the maximum number of pairwise nonadjacent extremities in the quotient graph**, that according to the above is always a lower bound for the asteroidal number. See [59, Fig. 1] for an example where it is smaller than the asteroidal number.

Throughout the paper, let Ext_α denote the class of all graphs whose quotient graph contains no more than α pairwise nonadjacent extremities.

► **Theorem 1.** *For every graph $G = (V, E) \in Ext_\alpha$, we can compute estimates $\bar{e}(u)$, $u \in V$, in deterministic $\mathcal{O}(\alpha^2 m)$ time so that $e(u) \geq \bar{e}(u) \geq e(u) - 1$ for every vertex u . In particular, we can compute a vertex whose eccentricity is within one of the diameter. Moreover, the exact diameter of G can be computed in deterministic $\mathcal{O}(\alpha^3 m^{3/2})$ time.*

Let us now sketch the main lines of our approach toward proving Theorem 1. First, we replace the input graph by its quotient graph, that can be done in linear time [74]. Then, we compute $\mathcal{O}(\alpha)$ shortest paths with one common end-vertex c , the union of which is a

¹ It was also shown in [23] that there exist AT-free graphs such that a multi-sweep LexBFS fails in computing their diameter. Therefore, we need to further process the LexBFS orderings of AT-free graphs, resp. of graphs of bounded asteroidal number, to output their diameter.

dominating set (to be compared with the dominating shortest path computed in [42] for the AT-free graphs). For that, we prove interesting new relations between graph extremities and LexBFS, but only for graphs that are prime for modular decomposition (this is why we need to consider the quotient graph). Roughly, our algorithm computes $\mathcal{O}(\alpha)$ pairwise nonadjacent extremities, i.e., the other end-vertices of the shortest-paths than c , by repeatedly executing a modified LexBFS. We stress that our procedure is more complicated than a multi-sweep LexBFS due to the need to avoid getting stuck between two mutually distant extremities. In doing so, the search for a diametral vertex can be now restricted to the closed neighbourhoods of only $\mathcal{O}(\alpha^2)$ vertices (namely, to the $\mathcal{O}(\alpha)$ furthest vertices from c on every shortest path). However, unlike what has been done in [42] for AT-free graphs, we failed in further pruning out each neighbourhood to a clique. Instead, we present a new procedure which given a vertex u outputs a vertex in its closed neighbourhood of maximum eccentricity. This is done by iterating on some extremities at maximum distance from vertex u . Therefore, a key to our analyses in this paper is the number of extremities in a graph. We provide several bounds on this number. In doing so, our runtime for exact diameter computation can be improved to $\mathcal{O}(\alpha^3 m)$ time for the bipartite graphs, and more generally to linear time for every graph in Ext_α of constant clique number. We present some more alternative time bounds for our Theorem 1 in Sec. 5.2.

Matching (Conditional) Lower bounds

The algorithm of Theorem 1 is combinatorial. In [42], the classic problem of detecting a simplicial vertex within an n -vertex graph is reduced in $\mathcal{O}(n^2)$ time to the diameter problem on $\mathcal{O}(n)$ -vertex AT-free graphs. The best known combinatorial algorithm for detecting a simplicial vertex in an n -vertex graph runs in $\mathcal{O}(n^3)$ time. In the same way, in [23], Corneil et al. proved an equivalence between the problem of deciding whether an AT-free graph has diameter at most two and a disjoint sets problem which has been recently studied under the name of high-dimensional OV [30]. The high-dimensional OV problem can be reduced to Boolean Matrix Multiplication, which is conjectured not to be solvable in $\mathcal{O}(n^{3-\epsilon})$ time, for any $\epsilon > 0$, using a combinatorial algorithm [75]. It is open whether high-dimensional OV can be solved faster than Boolean Matrix Multiplication. Therefore, due to both reductions from [42] and [22], the existence of an $\mathcal{O}(f(\alpha)m^{3/2-\epsilon})$ -time combinatorial algorithm for diameter computation within Ext_α , for some function f and for some $\epsilon > 0$, would be a significant algorithmic breakthrough.

Applications to some graph classes

Let us review some interesting subclasses of graphs of Ext_α , for some constant α , for which to the best of our knowledge the best-known deterministic algorithm for diameter computation until this paper has been the brute-force $\mathcal{O}(nm)$ -time algorithm.

A circle graph is the intersection graph of chords in a cycle. For every $k \geq 2$, a k -polygon graph is the intersection graph of chords in a convex k -polygon where the ends of each chord lie on two different sides. Note that the k -polygon graphs form an increasing hierarchy of all the circle graphs, and that the 2-polygon graphs are exactly the permutation graphs. Recently [40], an almost linear-time algorithm was proposed which computes a +2-approximation of the diameter of any k -polygon graph, for any fixed k . By [73], every k -polygon graph has asteroidal number at most k . Therefore, for the **k -polygon graphs**, we obtain an improved +1-approximation in linear time, and the first truly subquadratic algorithm for exact diameter computation.

A chordal graph is a graph with no induced cycle of length more than three. Chordal graphs are exactly the intersection graphs of a collection of subtrees of a host tree [48]. We call such a representation a tree model. The leafage of a chordal graph is the smallest number of leaves amongst its tree models. In particular, the chordal graphs of leafage at most two are exactly the interval graphs, which are exactly the AT-free chordal graphs. More generally, every chordal graph of leafage at most k also has asteroidal number at most k [65]. In [42], a randomized $\mathcal{O}(km \log^2 n)$ -time algorithm was presented in order to compute the diameter of chordal graphs of asteroidal number at most k . Our Theorem 1 provides a deterministic alternative, but at the price of a higher runtime. Even more strongly, by combining the ideas of Theorem 1 with some special properties of chordal graphs, we were able to improve the runtime to $\mathcal{O}(km)$ – see our Theorem 34. Previously, such a result was only known for interval graphs [66].

A *moplex* in a graph is a module inducing a clique and whose neighbourhood is a minimal separator (the notions of module and minimal separator are recalled in Sec. 2). Moplexes are strongly related to LexBFS; indeed, every vertex last visited during a LexBFS is in a moplex [6]. This has motivated some recent studies on k -moplex graphs, a.k.a., the graphs with at most k moplexes. In particular, every k -moplex graph has asteroidal number at most k [8, 31]. Hence, our results in this paper can be applied to the **k -moplex graphs**.

Finally, a *dominating pair* consists of two vertices x and y such that every xy -path is a dominating set. Note that every AT-free graph contains a dominating pair [27]. A dominating pair graph (for short, DP graph) is one such that every connected induced subgraph contains a dominating pair. The family $(K_n^+)_{n \geq 4}$ of DP graphs in [68, Sec. 4.1] shows that for every $\alpha \geq 2$, there exists a DP graph which is not in Ext_α . However, we here prove that every DP graph with diameter at least six is in Ext_2 – see our Lemma 16. In doing so, we obtain a deterministic $\mathcal{O}(m^{3/2})$ -time algorithm which, given an m -edge DP graph, either computes its diameter or asserts that its diameter is ≤ 5 . We left open whether the diameter of DP graphs can be computed in truly subquadratic time.

Organization of the paper

We give the necessary graph terminology for this paper in Sec. 2. Then, in Sec. 3 we present some properties of graph extremities which, to our knowledge, have not been noticed before our work. In particular if a graph is prime for modular decomposition, then there always exists a diametral path whose both ends are extremities of the graph. We think these results could be helpful in future studies on the diameter problem (for other graph classes), and in order to better understand the relevant graph structure to be considered for fast diameter computation. We complete Sec. 3 with additional properties of extremities for graphs of bounded asteroidal number and for DP graphs. In Sec. 4, we relate extremities to the properties of Lexicographic Breadth-First Search. Doing so, we design a general framework in order to compute extremities under various constraints. We prove Theorem 1 in Sec. 5, then we discuss some of its extensions in Sec. 6. We conclude this paper and propose some open questions in Sec. 7.

Due to lack of space, most proofs are either omitted or postponed to the appendix.

2 Preliminaries

We introduce in this section the necessary graph terminology for our proofs. Let $G = (V, E)$ be a graph. For any vertex $v \in V$, let $N_G(v) = \{u \in V \mid uv \in E\}$ be its (open) neighbourhood and let $N_G[v] = N_G(v) \cup \{v\}$ be its closed neighbourhood. Similarly, for any vertex-subset

$S \subseteq V$, let $N_G[S] = \bigcup_{v \in S} N_G[v]$ and let $N_G(S) = N_G[S] \setminus S$. For any vertices u and v , we call a subset $S \subseteq V$ a uv -separator if u and v are in separate connected components of $G \setminus S$. A minimal uv -separator is an inclusion-wise minimal uv -separator. We call a subset S a (minimal) separator if it is a (minimal) uv -separator for some vertices u and v . Alternatively, a full component for S is a connected component C of $G \setminus S$ such that $N_G(C) = S$. It is known [52] that S is a minimal separator if there exist at least two full components for S .

Distances

Recall that the distance $d_G(u, v)$ between two vertices u and v equals the minimum number of edges on a uv -path. Let the interval $I_G(u, v) = \{w \in V \mid d_G(u, v) = d_G(u, w) + d_G(w, v)\}$ contain all vertices on a shortest uv -path. Furthermore, for every $\ell \geq 0$, let $N_G^\ell[u] = \{v \in V \mid d_G(u, v) \leq \ell\}$ be the ball of center u and radius ℓ in G . We recall that the eccentricity of a vertex $v \in V$ is defined as $e_G(v) = \max_{u \in V} d_G(u, v)$. We sometimes omit the subscript if the graph G is clear from the context. Let $F(v) = \{u \in V \mid d(u, v) = e(v)\}$ be the set of vertices most distant to vertex v . The diameter and the radius of G are defined as $\text{diam}(G) = \max_{v \in V} e(v)$ and $\text{rad}(G) = \min_{v \in V} e(v)$, respectively. We call (x, y) a diametral pair if $d(x, y) = \text{diam}(G)$.

Modular decomposition

Two vertices $u, v \in V$ are twins if we have $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. A twin class is a maximal vertex-subset of pairwise twins. More generally, a module is a vertex-subset $M \subseteq V$ such that $N(x) \setminus M = N(y) \setminus M$ for any $x, y \in M$. Note that any twin class is also a module. We call G *prime* if its only modules are: \emptyset, V and $\{v\}$ for every $v \in V$ (trivial modules). A module M is *strong* if it does not overlap any other module, i.e., for any module M' of G , either one of M or M' is contained in the other or M and M' do not intersect. We denote by $\mathcal{M}(G)$ the family of all inclusion wise maximal strong modules of G that do not contain all the vertices of G . Finally, the *quotient graph* of G is the graph G' with vertex-set $\mathcal{M}(G)$ and an edge between every two $M, M' \in \mathcal{M}(G)$ such that every vertex of M is adjacent to every vertex of M' . The following well-known result is due to Gallai:

► **Theorem 2** ([47]). *For an arbitrary graph G exactly one of the following conditions is satisfied.*

1. G is disconnected;
2. its complement \overline{G} is disconnected;
3. or its quotient graph G' is prime for modular decomposition.

For general graphs, there is a tree representation of all the modules in a graph, sometimes called the modular decomposition, that can be computed in linear time [74]. Note that since we only consider connected graphs, only the two last items of Theorem 2 are relevant to our study. Moreover, it is easy to prove that if the complement of a graph G is disconnected, then we have $\text{diam}(G) \leq 2$. Therefore, the following result is an easy byproduct of Gallai's theorem for modular decomposition [47]:

► **Lemma 3** (cf. Theorem 14 in [29]). *Computing the diameter (resp., all eccentricities) of any graph G can be reduced in linear time to computing the diameter (resp., all eccentricities) of its quotient graph G' .*

An important observation for what follows is that, if a graph G belongs to Ext_α for some α , then so does its quotient graph G' . Hence, we may only consider *prime* graphs in Ext_α .

Hyperbolicity

The hyperbolicity of a graph G [54] is the smallest half-integer $\delta \geq 0$ such that, for any four vertices u, v, w, x , the two largest of the three distance sums $d(u, v) + d(w, x)$, $d(u, w) + d(v, x)$, $d(u, x) + d(v, w)$ differ by at most 2δ . In this case we say that G is δ -hyperbolic. To quote [34]: “As the tree-width of a graph measures its combinatorial tree-likeness, so does the hyperbolicity of a graph measure its metric tree-likeness. In other words, the smaller the hyperbolicity δ of G is, the closer G is to a tree metrically.” We will use in what follows the following “tree-likeness” properties of hyperbolic graphs:

► **Lemma 4** ([20]). *If G is δ -hyperbolic, then $\text{diam}(G) \geq 2\text{rad}(G) - 4\delta - 1$.*

► **Lemma 5** (Proposition 3(c) in [21]). *Let G be a δ -hyperbolic graph and let u, v be a pair of vertices of G such that $v \in F(u)$. We have $e(v) \geq \text{diam}(G) - 8\delta \geq 2\text{rad}(G) - 12\delta - 1$.*

► **Lemma 6** ([20]). *Let u be an arbitrary vertex of a δ -hyperbolic graph G . If $v \in F(u)$ and $w \in F(v)$, then let $c \in I(v, w)$ be satisfying $d(c, v) = \lfloor d(v, w)/2 \rfloor$. We have $e(c) \leq \text{rad}(G) + 5\delta$.*

3 Properties of graph extremities

We present several simple properties of graph extremities in what follows. In Sec. 3.1, we give bounds on the number of extremities in a graph. Then, we show in Sec. 3.2 that, for computing the vertex eccentricities of a prime graph (and so, its diameter), it is sufficient to only consider its so-called extremities. In Sec. 3.3, we relate the location of the extremities in a graph to the one of an arbitrary dominating target. We state in Sec. 3.4 a relationship between extremities and the hyperbolicity of a graph (this result easily follows from [59]). In Sec. 3.5, additional properties of extremities in some graph classes are discussed.

3.1 Bounds on the Number of Graph extremities

Every non-complete prime graph has at least two extremities [59]. The remainder of this section is devoted to proving an upper bound on the number of extremities in a prime graph. Unfortunately, there may be up to $\Theta(n)$ extremities in an n -vertex graph, even if it is AT-free. See the construction of [23, Fig. 2] for an example. It is worth mentioning this example also has clique-number equal to $\Theta(n)$. Our general upper bounds in what follows show that only dense prime graphs may have $\Omega(n)$ extremities.

► **Lemma 7.** *If $G \in \text{Ext}_\alpha$ is prime, then the number of its extremities is at most:*

- $\alpha \cdot \chi(G)$, where $\chi(G)$ denotes the chromatic number of G ;
- $R(\alpha + 1, \omega(G) + 1) - 1$, where $\omega(G)$ is the clique number of G , and $R(\cdot, \cdot)$ is a Ramsey number.

In particular, it is in $\mathcal{O}(\alpha\sqrt{m})$.

Proof. Let us denote by q the number of extremities of G , and let H be induced by all the extremities. Note that H is not necessarily connected. Since we assume that $G \in \text{Ext}_\alpha$, the independence number of H is at most α . In this situation, $q < R(\alpha + 1, \omega(G) + 1)$ (otherwise, either H would contain an independent set of size $\alpha + 1$, or H and so, G , would contain a clique of size $\omega(G) + 1$). Since the chromatic number of H is at most $\chi(G)$, we also have that H can be partitioned in at most $\chi(G)$ independent sets, and so, $q \leq \alpha \cdot \chi(G)$. In particular, $q = \mathcal{O}(\alpha\sqrt{m})$ because $\chi(G) = \mathcal{O}(\sqrt{m})$ for any graph G . ◀

Let $G \in Ext_\alpha$ be prime, with q extremities. Note that, using $R(s, t) = \mathcal{O}(t^{s-1})$ for any fixed s , we get that $q = \mathcal{O}(\alpha^{\omega(G)+1})$. That is in $\mathcal{O}(\alpha^3)$ for triangle-free graphs. For graphs of constant chromatic number, the bound of Lemma 7 is linear in α . In particular, $q = \mathcal{O}(\alpha\Delta)$ for the graphs of maximum degree Δ , and $q = \mathcal{O}(\alpha)$ for bipartite graphs.

3.2 Relationships with the diameter

To the best of our knowledge, the following relation between extremities and vertex eccentricities has not been noticed before:

► **Lemma 8.** *If x is a vertex of a prime graph $G = (V, E)$ with $|V| \geq 3$, then there exists an extremity y of G such that $d(x, y) = e(x)$.*

In particular, for every $y' \in F(x)$, there is an extremity $y \in F(x)$ so that $d(y, y') \leq 2$.

The proof of Lemma 8 is postponed to Appendix A. We observe that a slightly weaker version of Lemma 8 could be also deduced from Lemma 19 (proved in the next section).

► **Corollary 9.** *If $G = (V, E)$ is prime and $\text{diam}(G) \geq 2$, then there exist extremities x, y such that $d(x, y) = \text{diam}(G)$.*

Overall if we were given the q extremities of a prime graph G , then by Lemma 8, we could compute all eccentricities (and so, the diameter) in $\mathcal{O}(qm)$ time. By Lemma 7, this runtime is in $\mathcal{O}(\alpha m^{3/2})$ for the graphs within Ext_α , which is subquadratic for any fixed α . This bound can be improved to linear time for any graph of Ext_α with constant clique number. However, the best-known algorithms for computing the extremities run in $\mathcal{O}(nm)$ time and in $\mathcal{O}(n^{2.79})$ time [63], respectively. Furthermore, computing the extremities is at least as hard as triangle detection, even for AT-free graphs [63]. We leave as an open problem whether there exists a truly subquadratic algorithm for computing all extremities in a graph.

3.3 Relationships with Dominating targets

A dominating target in a graph G is a vertex-subset D with the property that any connected subgraph of G containing all of D must be a dominating set. Dominating targets of cardinality two have been studied under the different name of dominating pairs. In particular, every AT-free graph contains a dominating pair [27].

► **Lemma 10** (special case of Theorems 6 and 7 in [59]). *If G is prime, then every inclusion-wise maximal subset of pairwise nonadjacent extremities in G is a dominating target.*

We have the following relation between extremities and dominating targets:

► **Lemma 11.** *If D is a dominating target of a graph G (not necessarily prime), then every extremity of G is contained in $N[D]$. In particular, there are at most $(\Delta + 1) \cdot |D|$ extremities, where Δ denotes the maximum degree of G .*

Proof. Suppose by contradiction the existence of some extremity $v \notin N[D]$. Then, $H = G \setminus N[v]$ is a connected subgraph of G that contains all of D but such that v has no neighbour in $V(H)$. The latter contradicts that D is a dominating target of G . ◀

It follows from both Lemma 8 and Lemma 11 that, for any dominating target D in a prime graph, there is a diametral vertex in $N[D]$. We slightly strengthen this result, as follows. The following simple lemmas also generalize prior results on AT-free graphs [23] and graphs with a dominating pair [42].

- **Lemma 12.** *If D is a dominating target, then $F(x) \cap N[D] \neq \emptyset$ for any vertex x .
In particular if $F(x) \cap D = \emptyset$, then $F(x) \subseteq N(D)$.*

Proof. We may assume that $F(x) \cap D = \emptyset$ (else, we are done). In this situation, let $y \in F(x)$ be arbitrary. Then, let H be the union of shortest xu -paths, for every $u \in D$. Since H is a connected subgraph, we have $y \in N[H]$. In particular, there is a shortest xu -path P , for some fixed $u \in D$, such that $y \in N[P]$. Observe that $y \notin V(P)$ (otherwise, $d(x, y) \leq d(x, u)$, and therefore $u \in F(x)$). So, let $y^* \in V(P) \cap N(y)$. If $y^* \neq u$, then $d(x, y) \leq d(x, y^*) + 1 \leq (d(x, u) - 1) + 1 = d(x, u)$, and therefore, $u \in F(x)$. A contradiction. As a result, $y \in N(u) \setminus D \subseteq N(D)$. ◀

- **Corollary 13.** *If D is a dominating target of a graph G , and no vertex of D is in a diametral pair, then $x, y \in N(D)$ for every diametral pair (x, y) .*

This above Corollary 13 suggests the following strategy in order to compute the diameter of a prime graph G . First, we compute a small dominating target D . Then, we search for a diametral vertex within the neighbourhood of each of its $|D|$ vertices. If $G \in Ext_\alpha$, then according to Lemma 10 there always exists such a D with $\mathcal{O}(\alpha)$ vertices. However, we are not aware of any truly subquadratic algorithm for computing this dominating target. By Lemma 10, it is sufficient to compute a maximal independent set of extremities, but then we circle back to the aforementioned problem of computing all extremities in a graph. In Sec. 5 in the paper, we prove that we needn't compute a dominating target in full in order to determine what the diameter is. Specifically, we may only compute a strict subset $D' \subset D$ of a dominating target (for that, we use the techniques presented in Sec. 4). However, the price to pay is that while doing so, we also need to consider a bounded number of vertices outside of $N[D']$ and their respective neighbourhoods. Hence, the number of neighbourhoods to be considered grows to $\mathcal{O}(\alpha^2)$. This will be our starting approach for proving Theorem 1.

3.4 Relationships with Hyperbolicity

Recall the definition of δ -hyperbolic graphs in Sec. 2. In a δ -hyperbolic graph G , an “almost central” vertex of eccentricity $\leq rad(G) + c\delta$, for some $c > 0$, can be computed in linear time, using a double-sweep BFS (see Lemma 6). Then, according to Lemma 4, any diametral vertex must at a distance $\geq rad(G) - c'\delta$, for some $c' > 0$, to this almost central vertex. Roughly, we wish to combine these properties with the computation of some subset D' of a small dominating target (see Sec. 3.3) in order to properly locate some neighbourhood that contains a diametral vertex. For that, we need to prove here that graphs in Ext_α are δ -hyperbolic for some δ depending on α . Namely:

- **Lemma 14.** *Every graph $G \in Ext_\alpha$ is $(3\alpha - 1)$ -hyperbolic.*

3.5 Extremities in some Graph classes

We complete this section with the following inclusions between graph classes.

- **Lemma 15.** *Every graph G of asteroidal number k belongs to Ext_k .*

While this above Lemma 15 trivially follows from the respective definitions of Ext_k and the asteroidal number, the following result is less immediate:

- **Lemma 16.** *Every DP graph $G = (V, E)$ of diameter at least six belongs to Ext_2 .*

The proof of Lemma 16 is postponed to Appendix B. Lemma 16 does not hold for diameter-five DP graphs, as it can be shown from the example in [68, Fig. 6], that has three pairwise nonadjacent extremities. Moreover, for every $n \geq 4$, there exists a diameter-two DP graph K_n^+ with n pairwise nonadjacent extremities [68]. We left open whether, for any $d \in \{3, 4, 5\}$, there exists some constant $\alpha(d) \geq 3$ such that all diameter- d DP graphs belong to $Ext_{\alpha(d)}$.

4 A framework for computing extremities

We identify sufficient conditions for computing an independent set of extremities (not necessarily a maximal one). To the best of our knowledge, before this work there was no faster known algorithm for computing *one* extremity than for computing all such vertices. We present a simple linear-time algorithm for this problem on prime graphs – see Sec. 4.2. Then, we refine our strategy in Sec. 4.3 so as to compute one extremity avoided by some fixed connected subset. This procedure is key to our proof of Theorem 1, for which we need to iteratively compute extremities, and connect those to some pre-defined vertex c using shortest paths, until we obtain a connected dominating set. In fact, our approach in Sec. 4.3 works under more general conditions which we properly state in Def. 21. Our main algorithmic tool here is LexBFS, of which we first recall basic properties in Sec. 4.1.

4.1 LexBFS

The Lexicographic Breadth-First Search (LexBFS) is a standard algorithmic procedure, that runs in linear time [70]. We give a pseudo-code in Algorithm 1. Note that we can always enforce a start vertex u by assigning to it an initial non empty label. Then, for a given graph $G = (V, E)$ and a start vertex u , $LexBFS(u)$ denotes the corresponding execution of LexBFS. Its output is a numbering σ over the vertex-set (namely, the reverse of the ordering in which vertices are visited during the search). In particular, if $\sigma(i) = x$, then $\sigma^{-1}(x) = i$.

Algorithm 1 LexBFS [70].

Require: A graph $G = (V, E)$.

```

1: assign the label  $\emptyset$  to each vertex;
2: for  $i = n$  to 1 do
3:   pick an unnumbered vertex  $x$  with the largest label in the lexicographic order;
4:   for all unnumbered neighbours  $y$  of  $x$  do
5:     add  $i$  to  $label(y)$ ;
6:    $\sigma(i) \leftarrow x$  /* number  $x$  by  $i$  */;
```

We use some notations from [24]. Fix some LexBFS ordering σ . Then, for any vertices u and v , $u \prec v$ if and only if $\sigma^{-1}(u) < \sigma^{-1}(v)$. Similarly, $u \preceq v$ if either $u = v$ or $u \prec v$. Let us define $N_{\prec}(v) = \{u \in N(v) \mid u \prec v\}$ and $N_{\succ}(v) = \{u \in N(v) \mid v \prec u\}$. Let also \triangleleft denote the lexicographic total order over the sets of LexBFS labels. For every vertices u and v , $u \preceq v$, let $\lambda(u, v)$ be the label of vertex u when vertex v was about to be numbered. We stress that $\lambda(u, v) \preceq \lambda(v, v)$ (i.e., the vertex selected to be numbered at any step has maximum label for the lexicographic order). Furthermore, a useful observation is that $\lambda(u, v)$ is just the list of all neighbours of u which got numbered before v , ordered by decreasing LexBFS number. In particular, for any $u \preceq v$ we have $\lambda(u, v) = \lambda(v, v)$ if and only if $N_{\succ}(v) \subseteq N(u)$. We often use this latter property in our proofs.

► **Lemma 17** (monotonicity property [24]). *Let a, b, c and d be vertices of a graph G such that: $a \preceq c$, $b \preceq c$ and $c \prec d$. If $\lambda(a, d) \triangleleft \lambda(b, d)$, then $\lambda(a, c) \triangleleft \lambda(b, c)$.*

► **Corollary 18.** *Let x, y, z be vertices of a graph G such that: $x \preceq y \preceq z$, and $\lambda(x, z) = \lambda(z, z)$. Then, $\lambda(y, z) = \lambda(z, z)$.*

4.2 Finding one extremity

It turns out that finding *one* extremity is simple, namely:

► **Lemma 19.** *If $G = (V, E)$ is a prime graph with $|V| \geq 3$, and σ is any LexBFS order, then $v = \sigma(1)$ is an extremity of G .*

Proof. Suppose by contradiction $G \setminus N[v]$ to be disconnected. Let $u = \sigma(n)$ be the start vertex of the LexBFS ordering, and let C be any component of $G \setminus N[v]$ which does not contain vertex u . We denote by $z = \sigma(i)$, $n > i > 1$ the vertex of C with maximum LexBFS number. By maximality of z , we obtain that $N_{\succ}(z) \subseteq N(v)$. In particular, $\lambda(v, z) = \lambda(z, z)$. Then, let $M = \{w \in V \mid v \preceq w \preceq z\}$. Since we have $\lambda(v, z) = \lambda(z, z)$, by Corollary 18 we also get $\lambda(w, z) = \lambda(z, z)$ for every $w \in M$. But this implies $N(w) \setminus M = N_{\succ}(z)$ for each $w \in M$, therefore M is a nontrivial module of G . A contradiction. ◀

An alternative proof of Lemma 19 could be deduced from the work of Berry and Bordat on the relations between mplexes and LexBFS [7]. However, to the best of our knowledge, Lemma 19 has not been proved before.

4.3 Generalization

Before generalizing Lemma 19, we need to introduce a few more notions and terminology.

► **Definition 20.** *Let $G = (V, E)$ be a graph and let $u, v, w \in V$ be pairwise independent. We write $u \perp_w v$ if and only if u, v are in separate connected components of $G \setminus N[w]$.*

A vertex w intercepts a path P if $N[w] \cap V(P) \neq \emptyset$, and it misses P otherwise. We can check that if $u \perp_w v$, then w intercepts all uv -paths, and conversely if $u \not\perp_w v$ and $uw, vw \notin E$, then w misses a uv -path.

► **Definition 21.** *Let u and S be, respectively, a vertex and a vertex-subset of some graph $G = (V, E)$. We call S a u -transitive set if, for every $x \in S$ and $y \in V$ nonadjacent, $x \perp_y u \implies y \in S$.*

Next, we give examples of u -transitive sets.

► **Lemma 22.** *If H is a connected subgraph of a graph G , then its closed neighbourhood $N[H]$ is u -transitive for every $u \in V(H)$. In particular, every ball centered at u and of arbitrary radius is u -transitive.*

Proof. Let $x \in N[H]$ and y satisfy $x \perp_y u$. Since $x \in N[H]$ and $u \in V(H)$, there exists a xu -path P of which all vertices except maybe x are in H . Furthermore, since we assume that $x \perp_y u$, and so x and y are nonadjacent, we obtain $y \in N[P \setminus \{x\}] \subseteq N[H]$. ◀

For an example of non-connected u -transitive set, we may simply consider three pairwise nonadjacent vertices u, v, w in a cycle. Then, $S = \{v, w\}$ is u -transitive.

We are now ready to state the following key lemma:

► **Lemma 23.** *Let $G = (V, E)$ be a prime graph with $|V| \geq 3$, let $u \in V$ be arbitrary and let $S \subseteq V$ be u -transitive. If $V \neq S \cup N[u]$, then we can compute in linear time an extremity $v \notin S \cup N[u]$ such that $d(u, v)$ is maximized.*

The proof of Lemma 23 is postponed to Appendix C.

5 Proof of Theorem 1

In Sec. 5.1, we present a linear-time algorithm for computing a vertex whose eccentricity is within one of the true diameter. This part of the proof is simpler, and it gives some intuition for our exact diameter computation algorithm, which we next present in Sec. 5.2.

5.1 Approximation algorithm

► **Theorem 24.** *For every graph $G = (V, E) \in Ext_\alpha$, we can compute in deterministic $\mathcal{O}(\alpha^2 m)$ time estimates $e(v) \geq \bar{e}(v) \geq e(v) - 1$ for every vertex v .*

Proof. We may assume G to be prime by Lemma 3. Furthermore, let us assume that $|V| \geq 3$. We subdivide the algorithm in three main phases.

- First, we compute some shortest path by using a double-sweep LexBFS. More specifically, let x_1 be the last vertex numbered in a LexBFS. Let x_2 be the last vertex numbered in a LexBFS(x_1). We compute a vertex $c \in I(x_1, x_2)$ so that $d(c, x_1) = \lfloor d(x_1, x_2)/2 \rfloor$. Then, let P_1 (resp., P_2) be an arbitrary shortest cx_1 -path (resp., cx_2 -path). – Note that for AT-free graphs (but not necessarily in our case), the shortest x_1x_2 -path $P_1 \cup P_2$ is dominating [24]. –
- Second, we set $H := P_1 \cup P_2$. While H is not a dominating set of G , we compute an extremity $x_i \notin N[H]$ and we add an arbitrary shortest cx_i -path to H . – We stress that such an extremity x_i always exists due to H being connected and therefore $N[H]$ being c -transitive (see Lemma 22) and by Lemma 23. – Let x_3, x_4, \dots, x_t denote all the extremities computed. By construction, H is the union of t shortest paths P_1, P_2, \dots, P_t with one common end-vertex c .
- Finally, for every $1 \leq i \leq t$, let the subset U_i be composed of the $\min\{d(x_i, c) + 1, 66\alpha - 19\}$ closest vertices to x_i in P_i (including x_i itself). Let $U = \bigcup_{i=1}^t U_i$. For every vertex $v \in V$, we set $\bar{e}(v) := \max\{d(u, v) \mid u \in U\}$.

Correctness. Let $v \in V$ be arbitrary. Since H is a dominating set of G , some vertex $u \in H$, in the closed neighbourhood of any vertex of $F(v)$, must satisfy $d(u, v) \geq e(v) - 1$. In order to prove correctness of our algorithm, it suffices to prove the existence of one such vertex in U . For that, let δ be chosen such that G is δ -hyperbolic. By Lemma 6, $e(c) \leq rad(G) + 5\delta$. Furthermore if $u \in H$ satisfies $N[u] \cap F(v) \neq \emptyset$, then by Lemma 5, $e(u) \geq 2rad(G) - 12\delta - 2$. In particular, $d(u, c) \geq rad(G) - 17\delta - 2$ (else, $e(u) \leq d(u, c) + e(c) \leq 2rad(G) - 12\delta - 3$). For $1 \leq i \leq t$ such that $u \in V(P_i)$, since P_i is a shortest $x_i c$ -path of length at most $e(c) \leq rad(G) + 5\delta$, we get that u must be one of the $(rad(G) + 5\delta) - (rad(G) - 17\delta - 2) + 1 = 22\delta + 3$ closest vertices to x_i . By Lemma 14, $\delta \leq 3\alpha - 1$, therefore $22\delta + 3 \leq 66\alpha - 19$.

Complexity. Recall that the first phase of the algorithm consists in a double-sweep LexBFS. Hence, it can be done in linear time. Then, at every step of the second phase we must decide whether H is a dominating set of G , that can be done in linear time. If H is not a dominating set, then we compute an extremity $x_i \notin N[H]$, that can also be done in linear time by Lemma 23. We further compute an arbitrary shortest $x_i c$ -path P_i , that can be done in linear time using BFS. Overall, the second phase takes $\mathcal{O}(tm)$ time, with t the number of extremities computed. Finally, in the third phase, we need to execute $\mathcal{O}(\alpha)$ BFS for every shortest path P_1, P_2, \dots, P_t , that takes $\mathcal{O}(\alpha tm)$ time.

By Lemma 19, both x_1 and x_2 are also extremities. We observe that $x_1x_2 \notin E$ (else, since $x_2 \in F(x_1)$, x_1 would be universal, thus contradicting either that G is prime or $|V| \geq 3$). By construction, x_3, x_4, \dots, x_t are pairwise nonadjacent, and they are also nonadjacent to both x_1 and x_2 . Altogether combined, we obtain that x_1, x_2, \dots, x_t are pairwise nonadjacent extremities. As a result, $t \leq \alpha$. It implies that the total runtime is in $\mathcal{O}(\alpha^2 m)$. ◀

5.2 Exact computation

The following general result, of independent interest, is the cornerstone of Theorem 26:

► **Lemma 25.** *Let u be a vertex in a prime graph $G = (V, E)$. If G has q extremities, then we can compute in $\mathcal{O}(qm)$ time the value $\ell(u) = \max\{e(x) \mid x \in N[u]\}$, and a $x \in N[u]$ of eccentricity $\ell(u)$. This is $\mathcal{O}(\alpha m^{3/2})$ time if $G \in \text{Ext}_\alpha$.*

The proof of Lemma 25 involves several cumbersome intermediate lemmas. We postpone the proof of Lemma 25 to the end of this section, proving first our main result:

► **Theorem 26.** *For every graph $G = (V, E) \in \text{Ext}_\alpha$, we can compute its diameter in deterministic $\mathcal{O}(\alpha^3 m^{3/2})$ time.*

Proof (Assuming Lemma 25). By Lemma 3, we may assume G to be prime. Let us further assume that $|V| \geq 3$, and so that G cannot have a universal vertex.

Algorithm. We subdivide the procedure in three main phases, the two first of which being common to both Theorems 24 and 26.

- Let x_1 be the last vertex numbered in a LexBFS. We execute a LexBFS with start vertex x_1 . Let x_2 be the last vertex numbered in a LexBFS(x_1). We compute a $c \in I(x_1, x_2)$ so that $d(c, x_1) = \lfloor d(x_1, x_2)/2 \rfloor$. Let P_1, P_2 be shortest x_1c -path and x_2c -path, respectively.
- We set $H := P_1 \cup P_2$. While H is not a dominating set of G , we compute a new extremity $x_i \notin N[H]$, an arbitrary shortest $x_i c$ -path P_i , then we set $H := H \cup P_i$. In what follows, we denote by x_1, x_2, \dots, x_t the extremities computed in the two first phases of the algorithm. Let P_1, P_2, \dots, P_t be the corresponding shortest paths, whose union equals H .
- Finally, for every $1 \leq i \leq t$, let $U_i \subseteq V(P_i)$ contain the $\min\{d(x_i, c) + 1, 42\alpha - 11\}$ closest vertices to x_i . Let $L_i := \max\{\ell(u_i) \mid u_i \in U_i\}$. We output $L := \max_{1 \leq i \leq t} L_i$ as the diameter value.

Complexity. The first phase of the algorithm can be done in $\mathcal{O}(m)$ time, and its second phase in $\mathcal{O}(tm)$ time, with t the number of extremities computed. During the third and final phase, we need to apply Lemma 25 $\mathcal{O}(\alpha)$ times for every shortest path P_1, P_2, \dots, P_t . Hence, the above algorithm runs in $\mathcal{O}(t\alpha qm)$ time, with q the total number of extremities of G . By Lemma 7, we have that $q = \mathcal{O}(\alpha\sqrt{m})$. See Theorem 24 for a proof that $t \leq \alpha$. As a result, the above algorithm runs in $\mathcal{O}(\alpha^3 m^{3/2})$ time.

Correctness. Let us consider an arbitrary diametral pair (u, v) . Since H is a dominating set of G , we have $N[u] \cap H \neq \emptyset$. Let $u^* \in N[u] \cap H$, and observe that $e(u^*) \geq \text{diam}(G) - 1$. Then we claim that if $u^* \in V(P_i)$ for some $1 \leq i \leq t$, we must have $u^* \in U_i$. The proof is similar to what we did for Theorem 24. Specifically, let us choose δ such that G is δ -hyperbolic. By Lemma 6, $e(c) \leq \text{rad}(G) + 5\delta$. Furthermore, if $u^* \in H$ satisfies $e(u^*) \geq \text{diam}(G) - 1$, then by Lemma 4, $e(u^*) \geq 2\text{rad}(G) - 4\delta - 2$. In particular, $d(u^*, c) \geq \text{rad}(G) - 9\delta - 2$ (else, $e(u^*) \leq d(u^*, c) + e(c) \leq 2\text{rad}(G) - 4\delta - 3 < \text{diam}(G) - 1$). For $1 \leq i \leq t$ such

10:14 Obstructions to Faster Diameter Computation: Asteroidal Sets

that $u^* \in V(P_i)$, since P_i is a shortest $x_i c$ -path of length at most $e(c) \leq \text{rad}(G) + 5\delta$, we get that u^* must be one of the $(\text{rad}(G) + 5\delta) - (\text{rad}(G) - 9\delta - 2) + 1 = 14\delta + 3$ closest vertices to x_i . By Lemma 14, $\delta \leq 3\alpha - 1$, therefore $14\delta + 3 \leq 42\alpha - 11$. In this situation, $L \geq L_i \geq \ell(u^*) = e(u) = \text{diam}(G)$. Combined with the trivial inequality $L \leq \text{diam}(G)$, it implies that $L = \text{diam}(G)$. ◀

The actual runtime of Theorem 26 is $\mathcal{O}(\alpha^2 qm)$, where q denotes the number of extremities. By Lemma 7, this is in $\mathcal{O}(\alpha^3 m)$ for bipartite graphs, $\mathcal{O}(\alpha^3 \Delta m)$ for graphs with maximum degree Δ and in $\mathcal{O}(\alpha^5 m)$ for triangle-free graphs. More generally, this is linear time for all graphs of Ext_α with bounded clique number.

The remainder of this section is devoted to the proof of Lemma 25. The key idea here is that for every vertex u , there is an extremity $v \in F(u)$ such that $\max\{d(v, x) \mid x \in N[u]\} = \ell(u)$. Therefore, in order to achieve the desired runtime for Lemma 25, it would be sufficient to iterate over all extremities of G that are contained in $F(u)$. However, due to our inability to compute all extremities in subquadratic time, we are bound to use Lemma 23 for only computing some of these extremities. Therefore, throughout the algorithm, we further need to grow some u -transitive set whose vertices must be carefully selected so that they can be discarded from the search space. The next Lemmas 27 and 28 are about the construction of this u -transitive set.

► **Lemma 27.** *Let u and v be vertices of a graph G such that $v \in F(u)$, and let $X = \{x \in N[u] \mid d(x, v) = e(u)\}$. In $\mathcal{O}(m)$ time we can construct a set Y where:*

- $v \in Y$; Y is u -transitive;
- $d(y, x) \leq d(v, x)$ for every $y \in Y$ and $x \in X$.

We now complete Lemma 27, as follows:

► **Lemma 28.** *Let u and v be vertices of a graph G such that $v \in F(u)$, and $d(x, v) \leq e(u)$ for every $x \in N[u]$. In $\mathcal{O}(m)$ time we can construct a set S' where:*

- $v \in S'$; S' is u -transitive;
- $d(s, x) \leq e(u)$ for every $s \in S'$ and $x \in N[u]$.

Using Lemma 28 we end up the section proving Lemma 25:

Proof of Lemma 25. We may assume that $|V| \geq 3$ and therefore (since G is prime) that u is not a universal vertex. We search for a $v \in V$ at a distance $\ell(u)$ from some vertex of $N[u]$. Note that we have $e(u) \leq \ell(u) \leq e(u) + 1$. In particular we only need to consider the vertices $v \in F(u)$. We next describe an algorithm that iteratively computes some pairs $(v_0, S_0), (v_1, S_1), \dots$ such that, for any $i \geq 0$: (i) $v_i \in F(u)$ is an extremity; and (ii) S_i is u -transitive. We continue until either $d(v_i, x) = e(u) + 1$ for some $x \in N(u)$ or $F(u) \subseteq S_i$.

If $i = 0$ then, let v_0 be the last vertex numbered in a LexBFS(u), that is an extremity by Lemma 19. Otherwise ($i > 0$), let v_i be the output of Lemma 23 applied to u and $S = S_{i-1}$. Furthermore, being given the extremity v_i , let S'_i be computed as in Lemma 27 applied to u and v_i . Let $S_i = S'_i \cup S_{i-1}$ (with the convention that $S_{-1} = \emptyset$). We stress that S_i is u -transitive since it is the union of two u -transitive sets.

By Lemma 28, all vertices in S_i can be safely discarded since they cannot be at distance $e(u) + 1$ from any vertex of $N[u]$. Furthermore, since for every i we have $v_i \in S_i \setminus S_{i-1}$, the sequence $(F(u) \setminus S_{i-1})_{i \geq 0}$ is strictly decreasing with respect to set inclusion. Each step i takes linear time, and the total number of steps is bounded by the number of extremities in $F(u)$. For a graph within Ext_α , this is in $\mathcal{O}(\alpha\sqrt{m})$ according to Lemma 7. ◀

6 Extensions

Our algorithmic framework in Sec. 5.2 can be refined in several ways. We present such refinements for the larger class of all graphs having a dominating target of bounded cardinality. Some general results are first discussed in Sec. 6.1 before we address the special case of chordal graphs in Sec. 6.2.

6.1 More results on dominating targets

We start with the following observation:

► **Lemma 29.** *If a graph G contains a dominating target of cardinality at most k , then so does its quotient graph G' .*

By Lemma 29, we may only consider in what follows *prime* graphs with a dominating target of bounded cardinality.

By $K_{1,t}$, we mean the star with t leaves. A graph is $K_{1,t}$ -free if it has no induced subgraph isomorphic to $K_{1,t}$.

► **Lemma 30.** *Every $K_{1,t}$ -free graph G with a dominating target of cardinality at most k belongs to $\text{Ext}_{k(t-1)}$.*

► **Corollary 31.** *We can compute the diameter of $K_{1,t}$ -free graphs with a dominating target of cardinality at most k in deterministic $\mathcal{O}((kt)^3 m^{3/2})$ time.*

Let us now consider graphs with a dominating target of cardinality at most k and bounded maximum degree Δ . These graphs are $K_{1,\Delta+1}$ -free and therefore, according to Corollary 31, we can compute their diameter in deterministic $\mathcal{O}((k\Delta)^3 m^{3/2})$ time. We improve this runtime to quasi linear, while also decreasing the dependency on Δ , namely:

► **Theorem 32.** *For every $G = (V, E)$ with a dominating target of cardinality at most k and maximum degree Δ , we can compute its diameter in deterministic $\mathcal{O}(k^3 \Delta m \log n)$ time.*

Since under SETH we cannot compute the diameter of graphs with a dominating edge in subquadratic time [42], the dependency on Δ in Theorem 32 is conditionally optimal.

The proof of Theorem 32 is postponed to Appendix D. Being given a vertex c of small eccentricity, our main difficulty here is to compute efficiently a small number of shortest-paths starting from c whose union is a dominating set of G . This could be done by computing a dominating target of small cardinality. However, our framework in the prior Sec. 4 only allows us to compute extremities, and not directly a dominating target. Our strategy consists in including in some candidate subset all the neighbours of the extremities that are computed by our algorithm. By using Lemma 11, we can bound the size of this candidate subset by an $\mathcal{O}(k\Delta)$. Then, by using a greedy set cover algorithm, we manage to compute from this candidate subset a set of $\mathcal{O}(k \log n)$ shortest-paths that cover all but $\mathcal{O}(k\Delta \log n)$ vertices. We apply the prior techniques of Sec. 5 to all these shortest-paths (calling upon Lemma 25), while for the $\mathcal{O}(k\Delta \log n)$ vertices that they miss we compute their eccentricities directly.

6.2 Chordal graphs

By [42, Theorems 6 & 9], we can decide in linear time the diameter of chordal graphs with a dominating pair (resp., with a dominating triple) if the former value is at least 4 (resp., at least 10). It was also asked in [42] whether for every $k \geq 4$, there exists a threshold d_k such

that the diameter of chordal graphs with a dominating target of cardinality at most k can be decided in truly subquadratic time if the former value is at least d_k . We first answer to this question in the affirmative:

► **Theorem 33.** *If $G = (V, E)$ is chordal, with a dominating target of cardinality at most k , and such that $\text{diam}(G) \geq 4$, then we can compute its diameter in deterministic $\mathcal{O}(km)$ time.*

Roughly, we lower the runtime of Theorem 33 to linear by avoiding calling upon Lemma 25. Specifically, we prove that whenever $\text{diam}(G) \geq 4$ there is always one of the $\leq k$ extremities which we compute whose eccentricity equals the diameter. The analysis of Theorem 33 is involved, as it is based on several nontrivial properties of chordal graphs [19, 23, 25, 64, 70].

We complete the above Theorem 33 with an improved algorithm for computing the diameter of chordal graphs with bounded asteroidal number. For that, we combine the algorithmic scheme of Theorem 33 with a previous approach from [42]. More specifically, for the special case $\text{rad}(G) = 2$, we considerably revisit a prior technique from [42, Proposition 2], for split graphs, so that it also works on diameter-three chordal graphs.

► **Theorem 34.** *For every chordal graph $G = (V, E)$ of asteroidal number at most k , we can compute its diameter in deterministic $\mathcal{O}(km)$ time.*

7 Conclusion

We generalized most known algorithmic results for diameter computation within AT-free graphs to the graphs within the much larger class Ext_α , for any $\alpha \geq 2$. The AT-free graphs, but also every DP graph with diameter at least six, belong to Ext_2 . Furthermore, for every $\alpha \geq 3$, every graph of asteroidal number α (and so, every α -polygon graph, every α -moplex graph and every chordal graph of leafage equal to α) belong to Ext_α .

We left open whether our results could be extended to the problem of computing exactly all eccentricities. The algorithm in [42] for the AT-free graphs can also be applied to this more general setting. Specifically, in every AT-free graph, there exist three vertices u, v, w such that every vertex x is at distance $e(x)$ from a vertex in $N[u] \cup N[v] \cup N[w]$. An algorithm is proposed in [42] in order to prune out each of the three neighbourhoods to a clique while keeping the latter property. Now, being given a graph from Ext_α , using our techniques we could also compute a union of $\mathcal{O}(\alpha^2)$ neighbourhoods such that every vertex x is at maximum distance from some vertex in it. However, insofar our approach only allows us to extract from each neighbourhood a vertex of maximum eccentricity, that is less powerful than the pruning method in [42].

It could be also interesting to give bounds on the number of extremities (resp., of pairwise nonadjacent extremities) in other graph classes. Finally, we left as an open problem what the complexity of computing the diameter is within the graphs which can be made AT-free by removing at most k vertices. For stars and their subdivisions it suffices to remove one vertex, and therefore this class of graphs has unbounded asteroidal number even for $k = 1$.

References

- 1 A. Abboud, V. Vassilevska Williams, and J. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the Twenty-seventh Annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 377–391. SIAM, 2016.

- 2 J. Alman and V. Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.
- 3 H.-J. Bandelt, A. Dählmann, and H. Schütte. Absolute retracts of bipartite graphs. *Discrete Applied Mathematics*, 16(3):191–215, 1987.
- 4 H.-J. Bandelt and E. Pesch. Efficient characterizations of n -chromatic absolute retracts. *Journal of Combinatorial Theory, Series B*, 53(1):5–31, 1991.
- 5 J. Beisegel. Characterising AT-free graphs with BFS. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 15–26. Springer, 2018.
- 6 A. Berry and J.-P. Bordat. Separability generalizes Dirac’s theorem. *Discrete Applied Mathematics*, 84(1-3):43–53, 1998.
- 7 A. Berry and J.-P. Bordat. Local LexBFS properties in an arbitrary graph. In *Proceedings of Journées Informatiques Messines (JIM 2000)*, 2000.
- 8 A. Berry and J.-P. Bordat. Asteroidal triples of mplexes. *Discrete Applied Mathematics*, 111(3):219–229, 2001.
- 9 J. A. Bondy and U. S. R. Murty. *Graph theory*. Springer London, 2008.
- 10 M. Borassi, P. Crescenzi, and L. Trevisan. An axiomatic and an average-case analysis of algorithms and heuristics for metric properties of graphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 920–939. SIAM, 2017.
- 11 N. Bousquet and S. Thomassé. VC-dimension and Erdős–Pósa property. *Discrete Mathematics*, 338(12):2302–2317, 2015.
- 12 A. Brandstädt, V. Chepoi, and F.F. Dragan. The algorithmic use of hypertree structure and maximum neighbourhood orderings. *Discrete Applied Mathematics*, 82(1-3):43–77, 1998.
- 13 A. Brandstädt, P. Fičur, A. Leitert, and M. Milanič. Polynomial-time algorithms for weighted efficient domination problems in AT-free graphs and dually chordal graphs. *Information Processing Letters*, 115(2):256–262, 2015.
- 14 K. Bringmann, T. Husfeldt, and M. Magnusson. Multivariate Analysis of Orthogonal Range Searching and Graph Distances. *Algorithmica*, pages 1–24, 2020.
- 15 H. Broersma, T. Kloks, D. Kratsch, and H. Müller. Independent sets in asteroidal triple-free graphs. *SIAM Journal on Discrete Mathematics*, 12(2):276–287, 1999.
- 16 S. Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. *ACM Transactions on Algorithms (TALG)*, 15(2):1–38, 2018.
- 17 S. Cabello and C. Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. *Computational Geometry*, 42(9):815–824, 2009.
- 18 V. Chepoi, F. Dragan, and Y. Vaxès. Center and diameter problems in plane triangulations and quadrangulations. In *Symposium on Discrete Algorithms (SODA’02)*, pages 346–355, 2002.
- 19 V. Chepoi and F.F. Dragan. A linear-time algorithm for finding a central vertex of a chordal graph. In *European Symposium on Algorithms*, pages 159–170. Springer, 1994.
- 20 V. Chepoi, F.F. Dragan, B. Estellon, M. Habib, and Y. Vaxès. Diameters, centers, and approximating trees of δ -hyperbolic geodesic spaces and graphs. In Monique Teillaud, editor, *Proceedings of the 24th ACM Symposium on Computational Geometry (SoCG)*, pages 59–68. ACM, 2008.
- 21 V. Chepoi, F.F. Dragan, M. Habib, Y. Vaxès, and H. Alrasheed. Fast approximation of eccentricities and distances in hyperbolic graphs. *Journal of Graph Algorithms and Applications*, 23(2):393–433, 2019.
- 22 V. Chepoi, B. Estellon, and Y. Vaxès. Covering planar graphs with a fixed number of balls. *Discrete & Computational Geometry*, 37(2):237–244, 2007.
- 23 D. Corneil, F. Dragan, M. Habib, and C. Paul. Diameter determination on restricted graph families. *Discrete Applied Mathematics*, 113(2-3):143–166, 2001.
- 24 D. Corneil, S. Olariu, and L. Stewart. Linear time algorithms for dominating pairs in asteroidal triple-free graphs. *SIAM Journal on Computing*, 28(4):1284–1297, 1999.

- 25 D.G. Corneil, F.F. Dragan, and E. Köhler. On the power of BFS to determine a graph's diameter. *Networks: An International Journal*, 42(4):209–222, 2003.
- 26 D.G. Corneil, S. Olariu, and L. Stewart. A linear time algorithm to compute a dominating path in an AT-free graph. *Information Processing Letters*, 54(5):253–257, 1995.
- 27 D.G. Corneil, S. Olariu, and L. Stewart. Asteroidal triple-free graphs. *SIAM Journal on Discrete Mathematics*, 10(3):399–430, 1997.
- 28 D.G. Corneil and J. Stacho. Vertex ordering characterizations of graphs of bounded asteroidal number. *Journal of Graph Theory*, 78(1):61–79, 2015.
- 29 D. Coudert, G. Ducoffe, and A. Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. *ACM Transactions on Algorithms (TALG)*, 15(3):1–57, 2019.
- 30 M. Dalirrooyfard and J. Kaufmann. Approximation Algorithms for Min-Distance Problems in DAGs. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 60:1–60:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
- 31 C. Dallard, R. Ganian, M. Hatzel, M. Krnc, and M. Milanič. Graphs with two moplexes. In *Proceedings of LAGOS'21*, volume 195 of *Procedia Computer Science*, pages 248–256. Elsevier, 2021.
- 32 J. De Rumeur. *Communications dans les réseaux de processeurs*. Masson Paris, 1994.
- 33 F.F. Dragan. HT-graphs: centers, connected r-domination and Steiner trees. *The Computer Science Journal of Moldova*, 1(2):64–83, 1993.
- 34 F.F. Dragan, G. Ducoffe, and H. Guarnera. Fast Deterministic Algorithms for Computing All Eccentricities in (Hyperbolic) Helly Graphs. In Anna Lubiw and Mohammad R. Salavatipour, editors, *Algorithms and Data Structures (WADS)*, volume 12808 of *Lecture Notes in Computer Science*, pages 300–314. Springer, 2021.
- 35 F.F. Dragan and E. Köhler. An approximation algorithm for the tree t-spanner problem on unweighted graphs via generalized chordal graphs. *Algorithmica*, 69(4):884–905, 2014.
- 36 F.F. Dragan and F. Nicolai. LexBFS-orderings of distance-hereditary graphs with application to the diametral pair problem. *Discrete Applied Mathematics*, 98(3):191–207, 2000.
- 37 F.F. Dragan, F. Nicolai, and A. Brandstädt. LexBFS-orderings and power of graphs. In Fabrizio d'Amore, Paolo Giulio Franciosa, and Alberto Marchetti-Spaccamela, editors, *Graph-Theoretic Concepts in Computer Science (WG)*, volume 1197 of *Lecture Notes in Computer Science*, pages 166–180. Springer, 1996.
- 38 G. Ducoffe. A New Application of Orthogonal Range Searching for Computing Giant Graph Diameters. In *SOSA*, 2019.
- 39 G. Ducoffe. Beyond Helly Graphs: The Diameter Problem on Absolute Retracts. In Łukasz Kowalik, Michał Pilipczuk, and Paweł Rzażewski, editors, *Graph-Theoretic Concepts in Computer Science (WG)*, pages 321–335. Springer International Publishing, 2021.
- 40 G. Ducoffe. Isometric Embeddings in Trees and Their Use in Distance Problems. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
- 41 G. Ducoffe. Optimal centrality computations within bounded clique-width graphs. In *16th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 214 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
- 42 G. Ducoffe. The diameter of AT-free graphs. *Journal of Graph Theory*, 99(4):594–614, 2022.
- 43 G. Ducoffe and F.F. Dragan. A story of diameter, radius, and (almost) Helly property. *Networks*, 77(3):435–453, 2021.

- 44 G. Ducoffe, M. Habib, and L. Viennot. Diameter computation on H -minor free graphs and graphs of bounded (distance) VC-dimension. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1905–1922. SIAM, 2020.
- 45 A. Farley and A. Proskurowski. Computation of the center and diameter of outerplanar graphs. *Discrete Applied Mathematics*, 2(3):185–191, 1980.
- 46 F.V. Fomin, M. Matamala, E. Prisner, and I. Rapaport. AT-free graphs: linear bounds for the oriented diameter. *Discrete Applied Mathematics*, 141(1-3):135–148, 2004.
- 47 T. Gallai. Transitiv orientierbare graphen. *Acta Math. Hungarica*, 18(1):25–66, 1967.
- 48 F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974.
- 49 P. Gawrychowski, H. Kaplan, S. Mozes, M. Sharir, and O. Weimann. Voronoi diagrams on planar graphs, and computing the diameter in deterministic $\tilde{O}(n^{5/3})$ time. In *Symposium on Discrete Algorithms (SODA)*, pages 495–514. SIAM, 2018.
- 50 P. Golovach, P. Heggenes, D. Kratsch, D. Lokshtanov, D. Meister, and S. Saurabh. Bandwidth on AT-free graphs. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 573–582. Springer, 2009.
- 51 P.A. Golovach, D. Paulusma, and E.J. van Leeuwen. Induced disjoint paths in AT-free graphs. In *Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 153–164. Springer, 2012.
- 52 M. C. Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier, 2004.
- 53 J. Gorzny and J. Huang. End-vertices of LBFS of (AT-free) bigraphs. *Discrete Applied Mathematics*, 225:87–94, 2017.
- 54 M. Gromov. *Hyperbolic Groups*, pages 75–263. Springer, New York, NY, 1987.
- 55 M. Habib, R. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1-2):59–84, 2000.
- 56 H. Hempel and D. Kratsch. On claw-free asteroidal triple-free graphs. *Discrete Applied Mathematics*, 121(1-3):155–180, 2002.
- 57 R. Impagliazzo and R. Paturi. On the complexity of k -SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.
- 58 T. Kloks, D. Kratsch, and H. Müller. Asteroidal sets in graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 229–241. Springer, 1997.
- 59 T. Kloks, D. Kratsch, and H. Müller. On the structure of graphs with bounded asteroidal number. *Graphs and Combinatorics*, 17(2):295–306, 2001.
- 60 Dirk Koschützki, Katharina Anna Lehmann, Leon Peeters, Stefan Richter, Dagmar Tenfelde-Podehl, and Oliver Zlotowski. Centrality indices. In *Network Analysis*, pages 16–61. Springer, 2005.
- 61 D. Kratsch and H. Müller. Colouring AT-free graphs. In *European Symposium on Algorithms (ESA)*, pages 707–718. Springer, 2012.
- 62 D. Kratsch, H. Müller, and I. Todinca. Feedback vertex set on AT-free graphs. *Discrete Applied Mathematics*, 156(10):1936–1947, 2008.
- 63 D. Kratsch and J. Spinrad. Between $O(nm)$ and $O(n^\alpha)$. *SIAM Journal on Computing*, 36(2):310–325, 2006.
- 64 R. Laskar and D. Shier. On powers and centers of chordal graphs. *Discrete Applied Mathematics*, 6(2):139–147, 1983.
- 65 I.-J. Lin, T.A. McKee, and D.B. West. The leafage of a chordal graph. *Discussiones Mathematicae Graph Theory*, 18(1):23–48, 1998.
- 66 S. Olariu. A simple linear-time algorithm for computing the center of an interval graph. *International Journal of Computer Mathematics*, 34(3-4):121–128, 1990.
- 67 R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.
- 68 N. Pržulj, D.G. Corneil, and E. Köhler. Hereditary dominating pair graphs. *Discrete Applied Mathematics*, 134(1-3):239–261, 2004.

- 69 L. Roditty and V. Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing (STOC)*, pages 515–524, 2013.
- 70 D. Rose, R. Tarjan, and G. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.
- 71 R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995.
- 72 J. Stacho. 3-colouring AT-free graphs in polynomial time. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 144–155. Springer, 2010.
- 73 L. Stewart and R. Valenzano. On polygon numbers of circle graphs and distance hereditary graphs. *Discrete Applied Mathematics*, 248:3–17, 2018.
- 74 M. Tedder, D. Corneil, M. Habib, and C. Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *ICALP*, pages 634–645. Springer, 2008.
- 75 V. Vassilevska Williams and R.R. Williams. Subcubic equivalences between path, matrix, and triangle problems. *Journal of the ACM (JACM)*, 65(5):1–38, 2018.

A Proof of Lemma 8

The following lemma shall be used in our proofs:

► **Lemma 35** ([59]). *Let S be a minimal separator for a prime graph $G = (V, E)$. For any component C of $G \setminus S$, if C does not contain an extremity of G , then $N(c) \cap S$ is a separator of $G \setminus C$ for every $c \in C$.*

Recall for what follows that a vertex is called universal if and only if all other vertices are adjacent to it.

Proof of Lemma 8. Since we assume that $|V| \geq 3$, we have that x cannot be a universal vertex (otherwise, $V \setminus \{x\}$ would be a nontrivial module, thus contradicting that G is prime). Let $y \in F(x)$ be arbitrary and we assume that y is not an extremity. We shall replace y by some extremity y^* so that $d(x, y) = d(x, y^*) = e(x)$. First we observe that $x \notin N[y]$ because we assume that x is not a universal vertex. Let w be disconnected from x in $G \setminus N[y]$, and let $S \subseteq N[y]$ be a minimal wx -separator of G (obtained by iteratively removing vertices from $N[y]$ while w and x stay disconnected) – possibly, $y \notin S$. –

We continue with a useful property of the connected components of $G \setminus S$. Specifically, let C be any connected component of $G \setminus S$ not containing vertex x . We claim that $d(x, c) = e(x)$ for each $c \in C$. Indeed, every shortest xc -path contains a vertex $s \in S$ and therefore, $d(x, y) \leq 1 + d(s, x) \leq d(c, s) + d(s, x) = d(c, x)$. In particular, $C \subseteq N(S)$ (otherwise, $d(x, y) < 2 + d(s, x) \leq d(x, c)$ for any $c \in C \setminus N(S)$ and any $s \in S$ that is on a shortest cx -path). It implies that, if $c \in C$ is an extremity, then $d(c, y) \leq 2$.

In what follows, let X the connected component of x in $G \setminus S$. Then, amongst all vertices of $G \setminus S$ in another connected component than x , let a be minimizing $|N(a) \cap S|$ and let A be its connected component in $G \setminus S$. Let us assume that A does not contain an extremity of G (else, we are done). By Lemma 35, $N(a) \cap S$ is a separator of $G \setminus A$. Let b be separated from vertex x in $G \setminus (A \cup N(a))$. We claim that $b \notin S$. In order to see that, we first need to observe that X is a full component for S (otherwise, S could not be a minimal wx -separator). Therefore if $b \in S$, then the subset $X \cup \{b\}$ would be connected, thus contradicting that x and b are disconnected in $G \setminus (A \cup N(a))$. This proves our claim, and from now on we denote B the connected component of b in $G \setminus S$. Observe that $B \neq X$ (otherwise, $N(a) \cap S$ could not be a bx -separator in $G \setminus A$). We further claim that $N(B) \subseteq N(a)$. Indeed, recall

that X is a full component for S . Hence, if it were not the case that $N(B) \subseteq N(a)$ then the subset $X \cup B \cup (N(B) \setminus N(a))$ would be connected, thus contradicting that x and b are disconnected in $G \setminus (A \cup N(a))$.

By the above claim, we get $N(b') \cap S \subseteq N(B) \subseteq N(a) \cap S$, for every $b' \in B$. Thus, by minimality of $|N(a) \cap S|$, we obtain $N(b') \cap S = N(a) \cap S = N(B)$ for every $b' \in B$. But then, B is a module of G , and therefore $B = \{b\}$ because G is prime. Finally, suppose by contradiction b is not an extremity. By repeating the exact same arguments for b instead of a , we find another connected component $C = \{c\}$ of $G \setminus S$ so that: c is separated from x in $G \setminus (B \cup N(b)) = G \setminus N[b]$, and $N(c) = N(b) = N(a) \cap S$ (possibly, $C = A$ and $a = c$). However, it implies that b and c are twins, a contradiction. Overall, we may choose for our vertex y^* either an extremity of A (if there exists one) or vertex b . ◀

B Proof of Lemma 16

Proof of Lemma 16. Since the property of being a DP graph is hereditary, the quotient graph of any DP graph is also a DP graph. In particular, it suffices to prove even more strongly that an arbitrary DP graph G of diameter at least six (not necessarily prime) cannot contain three pairwise nonadjacent extremities. Suppose by contradiction the existence of three such extremities u, v, w . Let (x, y) be a dominating pair. By Lemma 11, $u, v, w \in N[x] \cup N[y]$. Without loss of generality, let $u, v \in N(x)$.

We claim that $S = N(u) \cap N(v)$ is not a separator of G . Suppose by contradiction that it is the case. Let $A = N[u] \setminus N(v)$, $B = N[v] \setminus N(u)$ and $X = V \setminus (A \cup B \cup S)$. Since $S \subseteq N(u)$ and u is an extremity of G , $B \cup X$ must be contained in some connected component of $G \setminus S$. But similarly, since $S \subseteq N(v)$ and v is an extremity of G , $A \cup X$ must be also contained in some connected component of $G \setminus S$. As a result, $X = \emptyset$, and the only two components of $G \setminus S$ are A and B . In particular, $w \in A \cup B \subseteq N[u] \cup N[v]$, that is a contradiction. Therefore, we proved as claimed that S is not a separator of G .

We now claim that u, v are still extremities in the subgraph $G \setminus S$. By symmetry, it suffices to prove the result for vertex u . Observing that removing all of $N[u] \setminus S$ leaves us with $G \setminus N[u]$, we are done because u is an extremity of G . – However, please note that vertex w may not be an extremity of $G \setminus S$. –

Since $G \setminus S$ is a DP graph, there exists a dominating pair (x', y') in this subgraph. Again by Lemma 11 we have $u, v \in N[x'] \cup N[y']$. Without loss of generality, let $u \in N[x']$, $v \in N[y']$ (possibly, $u = x'$, resp. $v = y'$). Now, let $P = (z_0 = x', z_1, \dots, z_\ell = y')$ be a shortest $x'y'$ -path of $G \setminus S$. By construction, P is a dominating path of $G \setminus S$. In order to derive a contradiction, we shall prove, using P , that $e(x) \leq 4$. Indeed, doing so, since (x, y) is a dominating pair, we obtain that $\text{diam}(G) \leq e(x) + 1 \leq 5$, a contradiction. For that, let $t \in V$ be arbitrary. We may further assume $t \notin N[x]$. If $t \in S$, then $u, v \in N(t) \cap N(x)$, therefore $d(x, t) \leq 2$. From now on, let us assume that $t \notin S$. Consider some index i such that $t \in N[z_i]$. Let Q_u be an induced xz_i -path such that $V(Q_u) \subseteq \{x, u, z_0, z_1, \dots, z_i\}$. In the same way, let Q_v be an induced xz_i -path such that $V(Q_v) \subseteq \{x, v, z_\ell, z_{\ell-1}, \dots, z_i\}$. Let us first assume that $V(Q_u) \cup V(Q_v)$ induces a cycle C . Then, the length of C must be ≤ 6 because C is a DP graph and no cycle of length ≥ 7 contains a dominating pair. As a result, $d(x, z_i) \leq 3$, and so $d(x, t) \leq 4$. For the remainder of the proof, we assume that there exists a chord in the cycle C induced by $V(Q_u) \cup V(Q_v)$. Since the three of P, Q_u, Q_v are induced paths, the only possible chords are: uz_j , for some $i + 1 \leq j \leq \ell$; or vz_j , for some $0 \leq j \leq i - 1$. By symmetry, let uz_j be a chord of C . Since P is a shortest $x'y'$ -path of $G \setminus S$, and $u \in N[x']$, we obtain that $j \in \{0, 1, 2\}$. In particular, we obtain that $i \leq 1$, and so, $d(x, t) \leq 1 + d(x, z_i) \leq 2 + d(u, z_i) \leq 4$. ◀

C Proof of Lemma 23

Proof of Lemma 23. We describe the algorithm before proving its correctness and analysing its runtime.

Algorithm. Let σ be any LexBFS(u) order of G , and let $w \notin S$ be minimizing $\sigma^{-1}(w)$. First we compute the maximum index i , $n > i \geq \sigma^{-1}(w)$, so that $\lambda(w, \sigma(i)) = \lambda(\sigma(i), \sigma(i)) = \lambda_i$. We set $j := 0$, $S_j := S$ and $M_j := \{v \notin S \mid w \preceq v \preceq \sigma(i)\}$. Then, while $|M_j| > 1$, we apply the following procedure:

- We partition M_j into groups $A_j^1, A_j^2, \dots, A_j^{p_j}$ so that two vertices are in the same group if and only if they have the same neighbours in S_j .
- Without loss of generality let A_j^1 be minimizing $|N(A_j^1) \cap S_j|$. We set $M_{j+1} = A_j^1$, $S_{j+1} = M_j \setminus M_{j+1}$.
- We set $j := j + 1$.

If $|M_j| = 1$ (end of the while loop), then we output the unique vertex $v \in M_j$.

Correctness. We prove by induction that the following three properties hold, for any $j \geq 0$: (i) M_j is a module of $G \setminus S_j$; (ii) every $v \in M_j$ is a vertex of $V \setminus S$ such that $d(u, v)$ is maximized; and (iii) for every $v \in M_j$, either v is an extremity of G or every connected component C of $G \setminus N[v]$ that does not contain vertex u satisfies $C \subseteq M_j$.

First, we consider the base case $j = 0$. Recall that $M_0 = \{v \notin S \mid w \preceq v \preceq \sigma(i)\}$, where i is the maximum index such that $\lambda(w, \sigma(i)) = \lambda(\sigma(i), \sigma(i)) = \lambda_i$. By Corollary 18 we have $\lambda(v, \sigma(i)) = \lambda_i$ for each $v \in M_0$, and so, $N_{\succ}(\sigma(i)) \subseteq N(v)$. The latter implies that $d(u, v) = d(u, \sigma(i))$ because σ is a (Lex)BFS order. Equivalently, $d(u, v) = d(u, w)$, and by the minimality of $\sigma^{-1}(w)$ we have that w is a vertex of $V \setminus S$ maximizing $d(u, w)$ (Property (ii)). Moreover, $N(v) \setminus (S \cup M_0) = N_{\succ}(\sigma(i)) \setminus S$ for each $v \in M_0$, therefore M_0 is a module of $G \setminus S$ (Property (i)). Now, let $v \in M_0$ be arbitrary. If v is not an extremity of G , then let C be any connected component of $G \setminus N[v]$ not containing vertex u . We claim that $C \subseteq M_0$. Indeed, suppose by contradiction $C \cap S \neq \emptyset$. By maximality of $d(u, v)$, we have $v \notin N[u]$ (for else, $V \setminus S \subseteq N[u]$). But then, we would get $s \perp_v u$ for any $s \in C \cap S$, and therefore by the definition of S we should have $v \in S$. A contradiction. As a result we have $C \cap S = \emptyset$. Furthermore, we have $w \preceq c$ for each $c \in C$, that follows from the minimality of $\sigma^{-1}(w)$. Let $z \in C$ be maximizing $\sigma^{-1}(z)$. In order to prove that $C \subseteq M_0$, it now suffices to prove that $\sigma^{-1}(z) \leq i$. Suppose by contradiction that it is not the case. We first observe $N_{\succ}(z) \subseteq N(v)$ by maximality of $\sigma^{-1}(z)$. Therefore (since in addition, $v \preceq \sigma(i) \prec z$), $\lambda(v, z) = \lambda(z, z)$. Since we suppose $v \preceq \sigma(i) \prec z$, we also get by Corollary 18 that $\lambda(\sigma(i), z) = \lambda(z, z)$. Then, $N_{\succ}(z) \subseteq N_{\succ}(\sigma(i)) \subseteq N(w)$. However, it implies $\lambda(w, z) = \lambda(z, z)$, thus contradicting the maximality of i ($< \sigma^{-1}(z)$) for this property.

Then, let us assume that M_j, S_j satisfy all of properties (i), (ii) and (iii), and that $|M_j| > 1$. By construction, all vertices in M_{j+1} have the same neighbours in S_j . Since $M_{j+1} \subset M_j$ and M_j is a module of $G \setminus S_j$, we obtain that M_{j+1} is a module of $G \setminus (M_j \setminus M_{j+1}) = G \setminus S_{j+1}$ (Property (i)). Property (ii) also holds because it holds for M_j and $M_{j+1} \subset M_j$. Now, let $v \in M_{j+1}$ be arbitrary, and let us assume it is not an extremity of G . Let C be any component of $G \setminus N[v]$ not containing vertex u . By Property (iii), $C \subseteq M_j$. Suppose by contradiction $C \not\subseteq M_{j+1}$. Let $z \in C \setminus M_{j+1}$. Since we have $C \cap S_j = \emptyset$, we obtain $N(z) \cap S_j \subseteq N(v) \cap S_j$. By minimality of $|N(v) \cap S_j| = |N(M_{j+1}) \cap S_j|$, we get $N(z) \cap S_j = N(v) \cap S_j$, which contradicts that $z \notin M_{j+1}$.

The above Property (iii) implies that, if $|M_j| = 1$, then the unique vertex $v \in M_j$ is indeed an extremity. Furthermore, by Property (ii), v is a vertex of $V \setminus S$ that maximizes $d(u, v)$. Hence, in order to prove correctness of the algorithm, all that remains to prove is that this algorithm eventually halts. For that, we claim that if $|M_j| > 1$ then $|M_{j+1}| < |M_j|$. Indeed, Property (i) asserts that M_j is a module of $G \setminus S_j$. Let $A_j^1, A_j^2, \dots, A_j^{p_j}$ be the partition of M_j such that two vertices are in the same group if and only if they have the same neighbours in S_j . Since G is prime, M_j cannot be a nontrivial module of G , and therefore $p_j \geq 2$. Hence, $|M_{j+1}| = |A_j^1| < |M_j|$, as claimed. This above claim implies that eventually we reach the case when $|M_j| = 1$, and so, the algorithm eventually halts.

Complexity. Computing the LexBFS ordering σ can be done in linear time [70]. Then once we computed vertex w in additional $\mathcal{O}(n)$ time, we can compute the largest index i such that $\lambda(w, \sigma(i)) = \lambda(\sigma(i), \sigma(i))$ as follows. We mark all the neighbours of vertex w , then we scan the vertices by decreasing LexBFS number, and we stop at the first encountered vertex $x \neq u$ such that all vertices in $N_{\succ}(x)$ are marked. Since all the neighbour-sets need to be scanned at most once, the total runtime for this step is linear. Finally, we dynamically maintain some partition such that, at the beginning of any step $j \geq 0$, this partition equals (M_j) . If $|M_j| > 1$, then we consider each vertex $s \in S_j$ sequentially, and we replace every group X in the partition by the nonempty groups amongst $X \setminus N(s), X \cap N(s)$. In doing so, we obtain the partition $(A_j^1, A_j^2, \dots, A_j^{p_j})$. We remove all groups but M_{j+1} , computing $S_{j+1} = M_j \setminus M_{j+1}$ along the way. By using standard partition refinement techniques [55, 67], after an initial processing in $\mathcal{O}(|M_0|)$ time each step j can be done in $\mathcal{O}\left(\sum_{s \in S_j} |N(s)| + |M_j \setminus M_{j+1}|\right)$ time. Because all the sets S_j are pairwise disjoint the total runtime is linear. ◀

D Proof of Theorem 32

Proof of Theorem 32. By Lemma 29, we may assume G to be prime. We proceed as follows:

- We compute a vertex c of eccentricity at most $rad(G) + 15k - 5$.
- Then, we set $H = \{c\}$, $X = \emptyset$. While H is not a dominating set of G , we compute an extremity $x_i \notin N[H]$, we add in X all vertices of $N[x_i]$ and, for every $y \in N[x_i]$, we add to H some arbitrary shortest yc -path P_y .
- Let $\mathcal{S} = \{N[P_x] \mid x \in X\} \cup \{N[v] \mid v \in V\}$. We apply a greedy set cover algorithm in order to extract from \mathcal{S} a sub-family \mathcal{S}' so that $\bigcup \mathcal{S}' = V$. Specifically, we set $\mathcal{S}' := \emptyset$, $U := V$, where U represents the uncovered vertices. While $U \neq \emptyset$, we add to \mathcal{S}' any subset $S \in \mathcal{S}$ such that $|S \cap U|$ is maximized, and then we set $U := U \setminus S$.
- Let $A = \{x \in X \mid N[P_x] \in \mathcal{S}'\}$ and let $B = \{v \in V \setminus A \mid N[v] \in \mathcal{S}'\}$.
 - For every $x \in A$, let W_x contain the $\min\{d(x, c) + 1, 42k - 11\}$ closest vertices to x in P_x . For each $w \in W_x$, we compute $\ell(w) = \max\{e(w') \mid w' \in N[w]\}$.
 - For every $v \in B$, we directly compute the eccentricities of all vertices in $N[v]$.
- We output the maximum eccentricity computed as the diameter value.

Correctness. In order to prove correctness of this above algorithm, let $u \in V$ be such that $e(u) = diam(G)$. By construction, \mathcal{S}' covers V , and therefore, either there exists a $x \in A$ such that $u \in N[P_x]$, or there exists a $v \in B$ such that $u \in N[v]$. In the latter case, we computed the eccentricities of all vertices in $N[v]$, including $e(u) = diam(G)$. Therefore, we only need to consider the former case. Specifically, let $u^* \in V(P_x) \cap N[u]$. To prove

correctness of the algorithm, it suffices to prove that $u^* \in W_x$. The proof that it is indeed the case is identical to that of Theorem 26. Indeed, since G has a dominating target of cardinality at most k , it contains an additive tree $(3k - 1)$ -spanner [59] and so – the same as graphs in Ext_k –, it is $(3k - 1)$ -hyperbolic [20, 35].

Complexity. By Lemma 6, we can compute a vertex c of eccentricity at most $rad(G) + 15k - 5$ in linear time. Then, during the second phase of the algorithm, we claim that each step can be done in $\mathcal{O}(\Delta m)$ time. Indeed, while H is not a dominating set of G , a new extremity $x_i \notin N[H]$ can be computed by applying Lemma 23. Furthermore, we can compute the shortest paths P_y , for $y \in N[x_i]$, by executing at most $\Delta + 1$ BFS. Overall, the runtime of this second phase is in $\mathcal{O}(t\Delta m)$, with t the number of extremities computed. We claim that $t \leq k$. Indeed, let D be a fixed (but unknown) dominating target of cardinality at most k . Let x_1, x_2, \dots, x_t denote all the extremities computed during this second phase. By Lemma 11, each extremity x_i computed is in $N[D]$. In order to prove that there are at most k extremities computed, it suffices to prove that $N[x_i] \cap N[x_j] \cap D = \emptyset$ for every $i < j$. Suppose by contradiction that there exists a vertex $v \in D$ such that $v \in N[x_i] \cap N[x_j]$. At step i , we add v in X , and therefore from this point on $x_j \in N[H]$. It implies that we cannot select x_j at step j , a contradiction. Hence, the total runtime of this second phase is in $\mathcal{O}(k\Delta m)$.

In order to bound the runtime of the third phase of the algorithm, we first need to bound the minimum number of subsets of \mathcal{S} needed in order to cover V . We claim that it is no more than $2k$. Indeed, as before let D be a fixed (but unknown) dominating target of cardinality at most k . For each $x \in D \cap X$, we select the set P_x . For each $v \in D \setminus X$, we select the sets $N[v]$ and $P_{x'}$, for some arbitrary $x' \in X$ such that $v \in N(P_{x'})$. The claim follows since we assume D to be a dominating target. It implies that the greedily computed sub-family \mathcal{S}' has its cardinality in $\mathcal{O}(k \log n)$. The cumulative size of all subsets in \mathcal{S} is at most $|X|n + 2m \leq k(\Delta + 1)n + 2m$. Furthermore, the greedy set cover algorithm runs in $|\mathcal{S}'| = \mathcal{O}(k \log n)$ steps. As a result, the total runtime for this third phase is in $\mathcal{O}(k \log n \cdot (k\Delta n + m)) = \mathcal{O}(k^2 \Delta m \log n)$.

Lastly, during the fourth and final phase, we need to apply Lemma 25 $\sum_{x \in A} |W_x| = \mathcal{O}(k|A|)$ times, and we need to execute $\sum_{v \in B} |N[v]| = \mathcal{O}(\Delta|B|)$ BFS. Each call to Lemma 25 takes $\mathcal{O}(qm)$ time, with q the number of extremities. By Lemma 11, $q = \mathcal{O}(k\Delta)$. Furthermore, we recall that $|A| + |B| \leq |\mathcal{S}'| = \mathcal{O}(k \log n)$. Therefore, the total runtime for this phase – and also for the whole algorithm – is in $\mathcal{O}(k^3 \Delta m \log n)$. ◀

On the Parameterized Complexity of Symmetric Directed Multicut

Eduard Eiben  

Department of Computer Science, Royal Holloway, University of London, Egham, UK

Clément Rambaud 

DIENS, École Normale Supérieure, CNRS, PSL University, Paris, France

Magnus Wahlström  

Department of Computer Science, Royal Holloway, University of London, Egham, UK

Abstract

We study the problem SYMMETRIC DIRECTED MULTICUT from a parameterized complexity perspective. In this problem, the input is a digraph D , a set of *cut requests* $C = \{(s_1, t_1), \dots, (s_\ell, t_\ell)\}$ and an integer k , and the task is to find a set $X \subseteq V(D)$ of size at most k such that for every $1 \leq i \leq \ell$, X intersects either all (s_i, t_i) -paths or all (t_i, s_i) -paths. Equivalently, every strongly connected component of $D - X$ contains at most one vertex out of s_i and t_i for every i . This problem is previously known from research in approximation algorithms, where it is known to have an $O(\log k \log \log k)$ -approximation. We note that the problem, parameterized by k , directly generalizes multiple interesting FPT problems such as (UNDIRECTED) VERTEX MULTICUT and DIRECTED SUBSET FEEDBACK VERTEX SET. We are not able to settle the existence of an FPT algorithm parameterized purely by k , but we give three partial results: An FPT algorithm parameterized by $k + \ell$; an FPT-time 2-approximation parameterized by k ; and an FPT algorithm parameterized by k for the special case that the cut requests form a clique, SYMMETRIC DIRECTED MULTIWAY CUT. The existence of an FPT algorithm parameterized purely by k remains an intriguing open possibility.

2012 ACM Subject Classification Theory of computation \rightarrow Fixed parameter tractability

Keywords and phrases Parameterized complexity, directed graphs, graph separation problems

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.11

1 Introduction

Graph separation problems have been studied in parameterized complexity for a long time, and with significant success. In particular for undirected graphs, a wide range of powerful FPT algorithms have been constructed, from the early results on ODD CYCLE TRANSVERSAL by Reed et al. [21] and MULTIWAY CUT by Marx [16], to quite generic problems such as VERTEX MULTICUT [2, 17]. In the latter problem, the input is an undirected graph G , a set of *cut requests* $C = \{(s_1, t_1), \dots, (s_\ell, t_\ell)\}$, and an integer k , and the goal is to find, if it exists, a set of at most k vertices whose removal disconnects s_i from t_i , for every $1 \leq i \leq \ell$. Marx showed an FPT algorithm for this problem parameterized by $k + \ell$ [16], but the question of an FPT algorithm parameterized by k alone remained open for a long time, until finally settled simultaneously by Bousquet et al. [2] and Marx and Razgon [15].

For directed graphs, by comparison, the success is more limited, and the line between FPT and W[1]-hard cut problems is much less clear. On the one hand, some high profile FPT algorithms do exist for directed graph problems. One of the earliest was DIRECTED FEEDBACK VERTEX SET, where the goal is to find a set of at most k vertices in a directed graph which intersects all directed cycles. This problem was shown to be FPT in 2007 by Chen et al. [3] by reduction to an auxiliary directed graph separation problem later dubbed SKEW MULTICUT. Later FPT results, following the FPT algorithms for MULTICUT on



© Eduard Eiben, Clément Rambaud, and Magnus Wahlström;
licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 11; pp. 11:1–11:17

Leibniz International Proceedings in Informatics



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

11:2 On the Parameterized Complexity of Symmetric Directed Multicut

undirected graphs, include the problems DIRECTED MULTIWAY CUT [6] and DIRECTED SUBSET FEEDBACK VERTEX SET [5]. However, other problems which are FPT on undirected graphs are intractable on digraphs. DIRECTED ODD CYCLE TRANSVERSAL was shown to be $W[1]$ -hard by Lokshtanov et al. [14], although it admits an FPT 2-approximation. For another example, DIRECTED MULTICUT is the natural generalization of MULTICUT to digraphs. Here, the input is a digraph D , a set of cut requests $C = \{(s_1, t_1), \dots, (s_\ell, t_\ell)\}$ and an integer k , and the goal is to find, if it exists, a set of at most k vertices whose removal cuts every path from s_i to t_i , for every $1 \leq i \leq \ell$. This problem is $W[1]$ -hard parameterized by k alone [17], even on directed acyclic graphs (DAGs) [13] or for just four cut requests [19].

With this background, it may be considered highly unlikely to find a natural cut problem on digraphs that directly generalizes VERTEX MULTICUT and which is FPT parameterized by the solution size alone. Yet, we consider a problem for which this appears intriguingly plausible.

For a first attempt at a modified problem definition, consider the variant where for every cut request (s_i, t_i) we require both directions (s_i, t_i) and (t_i, s_i) to be cut. However, this problem remains $W[1]$ -hard; indeed, it is equivalent to the original problem if the input graph is a DAG. Furthermore, it captures DIRECTED VERTEX MULTICUT on general digraphs: if $I = (D, T, k)$ is a DIRECTED VERTEX MULTICUT instance, construct D' by adding a new vertex s'_i and an arc $s'_i s_i$ for every $(s_i, t_i) \in T$. Then, there is no (t_i, s'_i) -path in D' , and cutting every (s'_i, t_i) -paths and (t_i, s'_i) -paths is equivalent to cut every (s_i, t_i) -path. This shows that this first symmetric version of DIRECTED VERTEX MULTICUT is $W[1]$ -hard too, even for $\ell = 4$.

However, another directed generalization of VERTEX MULTICUT has still unknown parameterized complexity.

SYMMETRIC DIRECTED VERTEX MULTICUT

Input: a digraph D , a set of pairs of vertices $C = \{(s_1, t_1), \dots, (s_\ell, t_\ell)\}$, and an integer k .

Parameter: k

Output: find, if there exists, a set X of at most k vertices whose removal cuts, for every $i = 1, \dots, \ell$, either all (s_i, t_i) -paths or all (t_i, s_i) -paths.

As with many directed cut problems, there are simple reductions between the edge- and the vertex deletion variants. We focus on the vertex deletion variant since it is easier to work with (cf. *shadow removal*, discussed below).

Let us make a few observations to get a feeling for the problem. Let $I = (D, C, k)$ be an instance of SYMMETRIC DIRECTED VERTEX MULTICUT (SYMMETRIC MULTICUT for short), and note that a set $X \subseteq V(D)$ is a solution if and only if s_i and t_i are in distinct strongly connected components in $D - X$ for every cut request (s_i, t_i) . This observation is important for understanding the structure of the problem.

We also note that SYMMETRIC MULTICUT generalizes several of the above-mentioned landmark FPT problems. Indeed, first consider VERTEX MULTICUT. Let $I = (G, C, k)$ be an instance of this problem. We can then produce an instance $I' = (D, C, k)$ of SYMMETRIC MULTICUT simply by replacing every edge $uv \in E(G)$ by the arcs uv and vu . Indeed, for every set $X \subseteq V(D)$, the strong and weak components of $D - X$ coincide. Hence X is a symmetric multicut in D if and only if it is a vertex multicut in G .

Next, let D be a digraph, and let $C = \binom{V(D)}{2}$ be the set containing all pairs of vertices over D . Then $I = (D, C, k)$ captures DIRECTED FEEDBACK VERTEX SET. More generally, consider DIRECTED SUBSET FEEDBACK VERTEX SET. In this problem, the input is a

digraph D , a set of arcs $S \subseteq E(D)$, and an integer k , and the goal is to find a set of at most k vertices which intersects every cycle containing an arc of S . By the above observation, $I = (D, S, k)$ can be interpreted as-is as an equivalent instance of SYMMETRIC MULTICUT. Thus, if SYMMETRIC MULTICUT is indeed FPT parameterized by k , it would make a significant generalization over the previous state of the art.

Our results. We are not able to settle the status of SYMMETRIC MULTICUT parameterized by k , but we give three partial results. First, we give an FPT algorithm for the combined parameter of $k + \ell$. Second, we show an FPT 2-approximation for SYMMETRIC MULTICUT with parameter k . Finally, we consider the problem SYMMETRIC DIRECTED MULTIWAY CUT, where the cut requests are a set $C = \binom{T}{2}$ containing all pairs over a set of terminals T ; i.e., every strongly connected component of $D - X$ is allowed to contain at most one vertex of T . We show that this restricted variant is FPT parameterized by k .

Technical overview. The first of these results is relatively straight-forward. We consider the solution structure of the problem, and show a simple FPT reduction to SKEW MULTICUT. Since SKEW MULTICUT is FPT parameterized by k , this finishes the result. This is analogous to the FPT algorithm for VERTEX MULTICUT parameterized by $k + \ell$ via reduction to MULTIWAY CUT, noted by Marx [16].

The FPT 2-approximation is more interesting. First, by iterative compression we can assume that we have a solution Y , say $|Y| \leq 2k + 1$, and want to determine the existence of a solution X with $|X| < |Y|$ (or otherwise prove that there is no solution of cardinality at most k). By branching on the intersection $X \cap Y$ we can assume that no vertex of Y is to be deleted. Furthermore, recall from above that a solution X to an instance $I = (D, C, k)$ is characterized by the strongly connected component structure of $D - X$. Hence, we may also guess a partition of Y into strongly connected components and a topological order on these components. After all these steps, we have an instance $I = (D', C, k')$ and a set $Y = \{y_1, \dots, y_r\} \subseteq V(D)$, such that Y is a symmetric multicut for (D, C) and with the assumption that we are looking for a symmetric multicut X such that $X \cap Y = \emptyset$ and in $D' - X$, y_i reaches y_j only if $i \leq j$. Thus, there are two remaining tasks to coordinate. X cuts all paths from y_j to y_i for $i < j$, and simultaneously, for every terminal y_i and cut pair (s_j, t_j) , X cuts at least one of s_j and t_j from the strongly connected component of y_i . We achieve a 2-approximation by treating these steps separately. The first property can be ensured by a reduction to SKEW MULTICUT; we note that SKEW MULTICUT is still FPT (using the algorithm of Chen et al. [3]) even if the underlying graph is not a DAG. The key observation is now that after deleting such a skew multicut for Y , the remaining task separates into $|Y|$ disjoint instances, one for each terminal $y \in Y$. Hence, it remains to solve the problem for an instance where there is a central vertex y such that for every cut request (s_i, t_i) , every closed walk on s_i and t_i passes through y . Solving this problem in FPT time finally yields and FPT-time 2-approximation for SYMMETRIC MULTICUT.

The FPT algorithm for SYMMETRIC DIRECTED MULTIWAY CUT is more technical. It works by adapting the algorithm for DIRECTED SUBSET FEEDBACK VERTEX SET of Chitnis et al. [5], but there are some technical complications. First, as a more robust formulation we consider the following setting. The input is a digraph D , a list A_1, \dots, A_ℓ of sets of arcs of D , and an integer k , with the restriction that each A_i is a “near-biclique”, $A_i = S_i \times T_i$ for some possibly overlapping vertex sets S_i and T_i . The task is to find a set $X \subseteq V(D)$ of at most k vertices such that no closed walk in $D - X$ contains arcs from two distinct sets A_i and A_j . Note that this version allows us to capture both the setting where terminals are

deletable and where terminals are non-deletable, e.g., by replacing a non-deletable terminal by $k + 1$ false twins, and for each terminal $t_i \in T$ letting S_i contain the twin copies of t_i and T_i their out-neighbours. More importantly, arc sets of the form $A_i = S_i \times T_i$ are closed under the vertex bypassing operation used in *shadow removal*, which the original problem formulation is not. (See Section 5.)

By the same setup as the FPT 2-approximation (and as Chitnis et al. [5]), we reduce to the iterative compression version where we additionally have a solution set Y and an ordering $y_1 < \dots < y_r$ over Y , with the assumption that y_i reaches y_j in $D - X$ if and only if $i < j$. We can now apply the shadow removal technique and consider the set of vertices R reachable from y_r in $D - X$. By shadow removal, this set is strongly connected to y_r in $D - X$. But here is the second complication. In DIRECTED SUBSET FEEDBACK VERTEX SET, R cannot contain any “terminal arc” at all, which allows the algorithm to proceed via an intricate branching step over graph separations in an auxiliary graph (using the so-called *anti-isolation lemma* and important separators branching). In our setting there can be an index i_0 such that R contains arcs of i_0 (and A_{i_0} can be unboundedly big). However, via an extra color-coding step, we are able to modify the method of Chitnis et al. [5], to allow us to guess i_0 and find R . We can then find a solution by repeating the process. In total, we show that SYMMETRIC DIRECTED MULTIWAY CUT has an algorithm in time $\mathcal{O}^*(2^{\mathcal{O}(k^3)})$.

Related work. The problem SYMMETRIC MULTICUT was first studied by Klein et al. [12] in the context of approximation algorithms. The results were improved upon by Even et al. [9], who showed that SYMMETRIC MULTICUT admits an $O(\log k \log \log k)$ -approximation, where k is the size of the optimal solution. By contrast, the best approximation ratio we are aware of for DIRECTED MULTICUT is just slightly better than $O(\sqrt{n})$ (Agarwal et al. [1], improving on previous work [4, 10]). Chuzhoy and Khanna [7] showed that achieving a subpolynomial approximation ratio for DIRECTED MULTICUT is hard.

We will make use of much of the toolbox developed for FPT algorithms for graph separation problems. In particular, the method of *iterative compression*, first used for ODD CYCLE TRANSVERSAL by Reed et al. [21]; the notion of *important separators*, which underpins Marx’ results on MULTIWAY CUT and related problems [16]; and the notion of *shadow removal*, developed by Marx and Razgon for VERTEX MULTICUT [17]. These notions are explained in Section 2. The work that is closest to our results is the FPT algorithm for DIRECTED SUBSET FEEDBACK VERTEX SET of Chitnis et al. [5].

Kim et al. [11] recently further extended the toolbox for directed graph separation problems by a method of *flow augmentation* for directed graph cuts. This settled several long-standing problems, among other results developing an FPT algorithm for the notorious ℓ -CHAIN SAT problem. Unfortunately, this method is not directly applicable to SYMMETRIC MULTICUT as the cut structure in the latter problem is more complex than simple (s, t) -cuts.

Ramanujan and Saurabh [20] considered SKEW-SYMMETRIC MULTICUTS, a problem family of multicuts on *skew-symmetric digraphs* (which is effectively a generalization of ALMOST 2-SAT). However, except for the problem name, this bears no relation to SYMMETRIC MULTICUT, as studied in this paper, or to SKEW MULTICUT, the auxiliary problem in the classic FPT algorithm for DIRECTED FEEDBACK VERTEX SET [3].

Structure of the paper. After introducing some useful tools in Section 2, we show in Section 3 that SYMMETRIC DIRECTED VERTEX MULTICUT is FPT when parameterized by both k and ℓ . Then, in Section 4, we give a 2-approximation algorithm with running time $f(k)n^{\mathcal{O}(1)}$. Finally, in Section 5, we show that a particular case, called SYMMETRIC DIRECTED MULTIWAY VERTEX CUT, is FPT.

2 Preliminaries

2.1 Important cuts

In a digraph D , if X, Y are disjoint sets of vertices, an (X, Y) -cut S is a set of vertices in $V(D) \setminus (X \cup Y)$ such that there is no (X, Y) -path in $D - S$. A classical tool in the design of FPT algorithms for problems of cut in a graph is the notion of important cut. An (X, Y) -cut is said to be important if there is no (X, Y) -cut further from X with smaller or equal size.

► **Definition 1.** Let D be a digraph and X, Y be two disjoint sets of vertices. An (X, Y) -cut S with set R of vertices reachable from X in $D - S$ is said to be important if

1. S is an inclusion-wise minimal (X, Y) -cut, and
2. there is no (X, Y) -cut $S' \neq S$ of size at most $|S|$ such that the set of vertices reachable from X in $D - S'$ is a superset of R .

Symmetrically, S is said to be anti-important if it is an important (Y, X) -cut in D^{op} , the digraph obtained from D by reversing every arc.

All fundamental results on important cuts are summarised in the following property. We refer the reader to [8, Part 8.5] for proofs.

► **Proposition 2.** Let D be a digraph, X, Y be disjoint sets of vertices and k be an integer.

1. One can test in polynomial time whether an (X, Y) -cut S is important.
2. If S is an (X, Y) -cut with set R of vertices reachable from X in $D - S$, one can compute in polynomial time an important (X, Y) -cut S' such that $|S'| \leq |S|$ and the set of vertices reachable from X in $D - S'$ contains R .
3. If \mathcal{S} is the set of important (X, Y) -cuts, then $\sum_{S \in \mathcal{S}} 4^{-|S|} \leq 1$.
4. If \mathcal{S}_k is the set of important (X, Y) -cuts of size at most k , then $|\mathcal{S}_k| \leq 4^k$ and \mathcal{S}_k can be enumerated in time $4^k n^{\mathcal{O}(1)}$.

2.2 Iterative compression

Iterative compression is a standard method in the design of FPT algorithms.

To avoid repetition, we give here a general property to deal with iterative compression. Let \mathcal{L} be a parameterized algorithmic problem such that an instance of \mathcal{L} has the form $I = (D, T, k)$ where D is a digraph, T depends on the problem and k is an integer. We suppose a few properties on \mathcal{L} :

- an instance $I = (D, T, k)$ is a yes-instance if and only if there exists a set X of at most k vertices satisfying a given property $P(D, T, X)$, which is supposed to be checkable in polynomial time,
- if D is empty, then \emptyset is a solution, and
- for every vertex $v \in V(D)$, if X satisfies $P(D - v, T, X)$, then $X \cup \{v\}$ satisfies $P(D, T, X \cup \{v\})$.

These three properties will clearly hold for every problems considered in this paper.

We say that an algorithm \mathcal{A} is an α -approximation for some $\alpha \geq 1$ if for every input instance (D, T, k) , either it concludes that there is no solution of size at most k , or it returns a solution of size at most αk . For $\alpha = 1$, this is an exact algorithm.

We now define the *compression* problem \mathcal{L}' by: given $I' = (D, T, Y, k)$ where (D, T, Y) satisfies P , find a solution of the \mathcal{L} instance (D, T, k) . The parameters are now $(k, |Y|)$. The compression problem is equivalent to the original one in the following sense:

► **Proposition 3.** *Let $\alpha \geq 1$, and $t(k, |Y|)$ be a real function which is increasing for each parameter if the other one is fixed, and $c \geq 0$ a constant. If there exists an algorithm \mathcal{A}' finding an α -approximation for \mathcal{L}' in time $t(k, |Y|)n^c$ then there exists an algorithm \mathcal{A} finding an α -approximation for \mathcal{L} in time $t(k, \alpha k + 1)n^{c+1}$. In particular, if \mathcal{L}' is FPT, then \mathcal{L} is FPT too.*

The proof is in the appendix. For further information on iterative compression we refer to [8, Chapter 4].

2.3 A general framework for shadow removal

The concept of shadow was first introduced by Marx and Razgon [17]. The idea is to make the problem easier by assuming that there exists a solution X such that every vertex $v \in V(D) \setminus X$ is reachable from a given set of vertices T , and can also reach T in $D - X$. Here, we give a general framework that was designed by Chitnis et al. [5].

Let D be a digraph and T a set of vertices. For every set of vertices X disjoint from T , we define the *shadow* of X to be the set of vertices in $V(D) \setminus (T \cup X)$ that either can not reach T in $D - X$, or are not reachable from T in $D - X$. Chitnis et al. [5] provided a set of sufficient conditions under which we can compute an over-approximation of the shadow of a solution to a problem; in other words, we can compute a set W , disjoint from T , such that there exists a solution X , disjoint from W , where the shadow of X is contained in W .

To state the result we need a few definitions from Chitnis et al. [5].

► **Definition 4.** *Let $\mathcal{F} = \{F_1, \dots, F_q\}$ be a set of subgraphs of D . We say that \mathcal{F} is T -connected if for every $i = 1, \dots, q$, every vertex in F_i can reach T by a walk completely in F_i , and is reachable from T by a walk completely in F_i . A set of vertices $X \subseteq V(D)$ is said to be an \mathcal{F} -transversal if for every $i \in \{1, \dots, q\}$, $F_i \cap X \neq \emptyset$.*

For example, if \mathcal{F} is a set of walks, as is the case in our application, then X is an \mathcal{F} -transversal if and only if X cuts every walk in \mathcal{F} . We can now give the main theorem that gives a superset of the shadow.

► **Theorem 5 ([5]).** *Let $T \subseteq V(D)$ and $k \in \mathbb{N}$. One can construct in time $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(1)}$ a family Z_1, \dots, Z_t of $t = 2^{\mathcal{O}(k^2)} \log^2 n$ sets of vertices such that for any set \mathcal{F} of T -connected subgraphs of D , if there exists an \mathcal{F} -transversal of size at most k , then there exists an \mathcal{F} -transversal X and $i \in \{1, \dots, t\}$ such that:*

1. $|X| \leq k$,
2. $X \cap Z_i = \emptyset$,
3. the shadow of X is included in Z_i .

2.4 SKEW VERTEX MULTICUT is FPT

In this section, we present a problem which is known to be FPT. This problem was first introduced by [3] in the first proof that DIRECTED FEEDBACK VERTEX SET is FPT.

SKEW VERTEX MULTICUT

Input: a digraph D , an ordered list of pair of vertices $(s_1, t_1), \dots, (s_r, t_r)$ and an integer k .

Parameter: k

Output: find, if there exists, a set X of at most k vertices such that there is no (s_j, t_i) -path in $D - X$ if $j \geq i$.

► **Theorem 6 ([3]).** *The problem SKEW VERTEX MULTICUT is FPT and can be solved in time $\mathcal{O}(4^k k^3 n^2)$.*

3 An FPT algorithm when parameterized by $k + \ell$

This section aims to prove the following theorem (remember that in SYMMETRIC DIRECTED VERTEX MULTICUT, k is the size of the desired solution, and ℓ is the number of cut requests).

► **Theorem 7.** *There is an algorithm that solves SYMMETRIC DIRECTED VERTEX MULTICUT in time $\mathcal{O}((2\ell + 1)^{2\ell} 4^k k^3 n^2)$.*

Proof. Let $I = (D, C, k)$ be a SYMMETRIC DIRECTED VERTEX MULTICUT instance. We suppose that I is a yes-instance and let X_{OPT} be a solution for I . Let $T = \bigcup_{(s,t) \in C} \{s, t\}$.

Let T_0, T_1, \dots, T_r with $r \leq 2\ell$ be a partition of T such that:

- $T_0 = X_{OPT} \cap T$,
- for every $i \in \{1, \dots, r\}$ and every $t, t' \in T_i$, t and t' are strongly connected in $D - X_{OPT}$,
- there is no (T_j, T_i) -path in $D - X_{OPT}$ if $j > i$.

Such a partition exists: consider the strongly connected components of $D - X_{OPT}$ and order them into a topological order C_1, C_2, \dots, C_r , that is an ordering such that for every arc uv in $D - X_{OPT}$ with $u \in C_i$ and $v \in C_j$, we have $i \leq j$. Then set $T_i = C_i \cap T$ for every $i \in \{1, \dots, r\}$.

The first step of our algorithm guesses that partition, thereby multiplying the running time by at most $(2\ell + 1)^{2\ell}$. Reject any partition where $s, t \in T_i$ for any $(s, t) \in C$ and any i . Now, we consider the digraph D' obtained by removing T_0 from D and merging each T_i into a single vertex t_i , for every $i = 1, \dots, r$.

Let $I' = (D', \{(t_1, t_2), \dots, (t_{r-1}, t_r)\}, k - |T_0|)$, a SKEW VERTEX MULTICUT instance. Clearly, $X_{OPT} \setminus T_0$ is a solution for I' , by definition of T_0, \dots, T_r . Reciprocally, if I' has a solution X' , then consider $X = T_0 \cup X'$, which has size at most $(k - |T_0|) + |T_0| = k$. If X is not a solution for I , then there exists $(s, t) \in C$ strongly connected in $D - X$. Then, s and t are in the same T_i for some i , and thus s and t are strongly connected in $D - X_{OPT}$, contradicting the fact that X_{OPT} is a solution for I .

Thus, one can solve SYMMETRIC DIRECTED VERTEX MULTICUT by first guessing T_0, \dots, T_r and then solving that SKEW VERTEX MULTICUT instance using Theorem 6. This algorithm has running time at most $\mathcal{O}((2\ell + 1)^{2\ell} 4^k k^3 n^2)$. ◀

4 A 2-approximation algorithm

In this part, we give an FPT algorithm that finds a solution of size at most $2k$ for SYMMETRIC DIRECTED VERTEX MULTICUT if it is known that there exists a solution of size at most k .

4.1 Iterative compression and first guesses

This section aims to prove that it is enough to find a 2-approximation algorithm for the following problem:

SYMMETRIC DIRECTED VERTEX MULTICUT COMPRESSION

Input: A digraph D , a set of pair of vertices $C = \{(s_1, t_1), \dots, (s_\ell, t_\ell)\}$, an integer k , and a solution Y of the SYMMETRIC VERTEX MULTICUT instance (D, C, k) , of size at most $2k + 1$, with an ordering y_1, \dots, y_r of Y .

Parameter: $(k, |Y|)$

Output: Find, if there exists, a set X of at most k vertices disjoint from Y such that:

1. for every pair of terminals $(s, t) \in C$ with $s, t \notin X$, s and t are not strongly connected in $D - X$, and
2. there is no (y_j, y_i) -path in $D - X$ if $j > i$.

► **Proposition 8.** *Let $t(k, |Y|)$ be a positive function that is non decreasing if one parameter is fixed, and $c \geq 2$ a constant.*

If SYMMETRIC DIRECTED VERTEX MULTICUT COMPRESSION has a 2-approximation algorithm \mathcal{A}' with time complexity $t(k, |Y|)n^c$, then SYMMETRIC DIRECTED VERTEX MULTICUT has a 2-approximation algorithm \mathcal{A} with time complexity at most $(2k+2)^{2k+1}t(k, 2k+1)n^{c+1}$.

Proof. First, we directly apply Property 3 with $\alpha = 2$ and thus it is enough to reduce the compression problem of SYMMETRIC DIRECTED VERTEX MULTICUT to SYMMETRIC DIRECTED VERTEX MULTICUT COMPRESSION.

Consider an instance $I = (D, C, k, Y)$ of that compression problem which is supposed to be a yes-instance, with an optimal solution X_{OPT} . It is enough to show that a 2-approximation for I can be found with at most $(|Y| + 1)^{|Y|}$ calls to \mathcal{A}' . To do that, we guess the structure of Y in $D - X_{OPT}$. More precisely, we guess a partition of Y into Y_0, Y_1, \dots, Y_r such that:

1. $Y_0 = X_{OPT} \cap Y$, and
2. if $y, y' \in Y_i$ then y and y' are strongly connected in $D - X_{OPT}$, and
3. there is no (Y_j, Y_i) -path in $D - X_{OPT}$ if $j > i$.

Such a partition exists by taking the intersection of the strongly connected components of $D - X_{OPT}$ with Y . This guess multiplies the running time by at most $(|Y| + 1)^{|Y|} \leq (2k + 2)^{2k+1}$.

We now claim that the instance of the compression problem I' obtained by

1. removing Y_0 from D and decreasing k by $|Y_0|$, and
2. merging each Y_i into a single vertex y_i ,

is equivalent to I . More precisely, if I is a yes-instance, then I' too by taking $X_{OPT} \setminus Y$ as a solution. Reciprocally, if I' has a solution X' of size at most $2(k - |Y_0|)$ then $X' \cup Y_0$ is a solution for I of size at most $2(k - |Y_0|) + |Y_0| \leq 2k$. This proves the property. ◀

The remaining of this section shows that SYMMETRIC DIRECTED VERTEX MULTICUT COMPRESSION has a 2-approximation algorithm.

4.2 Finding a skew multicut of Y

The first step of our algorithm computes a set $X_0 \subseteq V(D) \setminus Y$ of at most k vertices such that there is no (y_j, y_i) -path in $D - X_0$ if $j > i$.

To do that, we use the problem SKEW VERTEX MULTICUT that is known to be FPT. We directly apply Theorem 6 to the instance $(D, ((y_1, y_2), (y_2, y_3), \dots, (y_{r-1}, y_r)), k)$ to compute a set X_0 of at most k vertices as wanted. Indeed, by definition of SKEW VERTEX MULTICUT, for every $j > i$, there is no (y_j, y_i) -path in $D - X_0$. This strong property will allow us to find in the next subsection a solution of size at most k in $D - X_0$.

4.3 Finding a solution in the simplified instance

This section shows how to compute a solution for $I = (D - X_0, C, k, Y)$. This will result in a set X_1 of size at most k such that there is no pair s_i, t_i strongly connected in $D - X_0 - X_1$, that is, $X_0 \cup X_1$ is a solution of size at most $2k$.

To do that, first note that any vertex $v \in V(D) \setminus Y$ can be strongly connected with at most one vertex in Y in $D - X_0$. Our first claim shows that we can assume that exactly one vertex in Y is strongly connected with v .

▷ **Claim 9.** If $v \in V(D) \setminus (X_0 \cup Y)$ is strongly connected to no vertex in Y in $D - X_0$, then $I' = (D - X_0 - v, C \setminus \{ab \in C \mid a = v \text{ or } b = v\}, k, Y)$ and I have the same set of solutions.

Proof. Clearly, if I has a solution X' , then X' is a solution for I' as every closed walk in $D - X_0 - v$ is also in $D - X_0$. Reciprocally, if X' is a solution for I' , then adding v to $D - X_0 - v - X'$ does not create any closed walk passing through at least one vertex in Y . But any closed walk passing through a cut request $(s, t) \in C$ must pass through at least one vertex in Y . It follows that no pair of terminals is strongly connected in $D - X_0 - X'$ and X' is a solution for I . \triangleleft

Thus, we can remove every vertex strongly connected to no vertex in Y . We now denote by $\ell(v)$ the unique integer such that v is strongly connected with $y_{\ell(v)}$.

\triangleright **Claim 10.** Let $(s, t) \in C$ be a terminal arc. If $\ell(s) \neq \ell(t)$, then $I'' = (D, C \setminus \{(s, t)\}, k, Y)$ and I' have the same set of solutions.

Proof. Clearly, if I' has a solution, then I'' too. Reciprocally, if I'' has a solution X'' , then every terminal arc different from s, t is not strongly connected in $D - X_0 - X''$. But s and t can not be strongly connected as s and t are not strongly connected in $D - X_0$. Thus, X'' is a solution for I' too. \triangleleft

We now assume that for every pair of terminal s, t , $\ell(s) = \ell(t)$. The next claim shows that we can process each strongly connected component in $D - X_0$ independently.

\triangleright **Claim 11.** If there is an arc uv with u and v not strongly connected in $D - X_0$, then $I'' = (D - uv, C, k, Y)$ and I' have the same set of solutions.

Proof. If I' has a solution X' , then X' is clearly a solution for I'' . Reciprocally, if X'' is a solution for I'' , then adding uv to $D - X_0 - uv$ does not create any closed walk, and thus X'' is a solution for I' too. \triangleleft

Now, we assume that $D - X_0$ has $|Y|$ weakly connected components Y_1, \dots, Y_r such that for every i , $V(Y_i) \cap Y = \{y_i\}$. Observe that now the weakly connected components are strongly connected. Let X_{OPT} be an optimal solution for I' . Then we guess the values $k_i = |X_{OPT} \cap Y_i|$, which multiplies the complexity of our algorithm by at most $(k+1)^{|Y|} = k^{\mathcal{O}(k)}$. Now, we solve each instance $I_i = (Y_i, C, k_i, \{y_i\})$ independently.

The key result is the following “pushing” claim, that shows how to construct X_1 as a union of important cuts. We denote by $X_{i,OPT} = X_{OPT} \cap Y_i$ a solution of I_i , that we suppose to exist.

\triangleright **Claim 12.** Let $(s, t) \in C$ be a terminal arc strongly connected in Y_i . Let $(a, b) \in \{(s, y_i), (y_i, s), (t, y_i), (y_i, t)\}$ be such that $X_{i,OPT}$ includes an (a, b) -cut.

- if $a = y_i$, let S be the set of vertices in $X_{i,OPT}$ with an in-neighbour reachable from y_i in $Y_i - X_{i,OPT}$ and S' be the anti-important (a, b) -cut given by Property 2. Then $X' = (X_{OPT} \setminus S) \cup S'$ is a solution for I_i too,
- symmetrically, if $b = y_i$, let S be the set of vertices in $X_{i,OPT}$ with an out-neighbour that reaches y_i in $Y_i - X_{i,OPT}$ and S' be the important (a, b) -cut given by Property 2. Then $X' = (X_{OPT} \setminus S) \cup S'$ is a solution for I_i too.

Proof. As s and t are not strongly connected in $Y_i - X_{i,OPT}$, $X_{i,OPT}$ must contain an (a, b) -cut for at least one $(a, b) \in \{(s, y_i), (y_i, s), (t, y_i), (y_i, t)\}$. It is enough to show the first point, as the second one is the first one applied to D^{op} the digraph obtained from D by reversing every arc.

First, as $|S'| \leq |S|$, we have $|X'| \leq |X_{i,OPT}| \leq k_i$. It remains to show that there is no pair $(s', t') \in C$ strongly connected in $Y_i - X'$. Suppose that such a counterexample (s', t') exists. Then there exists a closed walk P passing through y_i, s' and t' . This walk must pass

11:10 On the Parameterized Complexity of Symmetric Directed Multicut

through $S' \setminus S$ as it does not exist in $Y_i - X_{i,OPT}$. But then there exists $v \in S \setminus S'$ reachable from y_i in $Y_i - S'$, contradicting the fact that the set of vertices reachable from y_i in $Y_i - S$ includes the set of vertices reachable from y_i in $Y_i - S'$. \triangleleft

We can now give the algorithm that solves $I_i = (Y_i, C, k_i, \{y_i\})$ as Algorithm 1.

■ **Algorithm 1** Algorithm for single-terminal case $I_i = (Y_i, C, k_i, \{y_i\})$.

```

 $X_i \leftarrow \emptyset;$ 
while there exists  $(s, t) \in C \cap V(Y_i)^2$  strongly connected in  $Y_i - X_i$  do
    | guess a direction  $(a, b) \in \{(s, y_i), (y_i, s), (t, y_i), (y_i, t)\};$ 
    | if  $a = y_i$  then
    |   | guess an anti-important  $(a, b)$ -cut  $S'$  of size at most  $k_i - |X_i|;$ 
    | else
    |   | guess an important  $(a, b)$ -cut  $S'$  of size at most  $k_i - |X_i|;$ 
    | end
    | add  $S'$  to  $X_i;$ 
end
return  $X_i;$ 

```

If the algorithm returns a value, then it is clearly a solution. We now show that there exists a sequence of guesses that leads to a solution if it exists. More precisely, we show that the following invariant holds: At every iteration of the loop, there is a possible value of X_i such that X_i can be extended to a solution for I_i if it exists. This invariant initially holds. If the results holds at some iteration for a set X_i , let $X_{i,OPT}$ be a solution that contains X_i , and for the first guess take (a, b) such that $X_{i,OPT}$ contains an (a, b) -cut S . By Claim 12 there exists an important or anti-important (a, b) -cut S' of size at most $|S|$ such that $(X_{i,OPT} \setminus S) \cup S'$ is still a solution. Thus, there exists a solution that contains S' and we can safely add it to X_i .

To see that the algorithm works in time $8^k n^{\mathcal{O}(1)}$, consider the recursion tree formed by recursively branching over all possible values of a guess, for each guess made in the algorithm. We denote by $t(k)$ the number of leaves of this recursion tree in the worst case. We show by induction on k that $t(k)4^{-k} \leq 4^k$. If $k = 0$, the result is clear. Otherwise, if we assume the result for smaller values of k , then we have

$$t(k)4^{-k} \leq 4 \sum_{S \in \mathcal{S}_k} t(k - |S|)4^{-k} \leq \sum_{S \in \mathcal{S}_k} t(k - |S|)4^{-(k-|S|)} \leq \sum_{S \in \mathcal{S}_k} 4^{k-|S|} \leq 4^k \sum_{S \in \mathcal{S}_k} 4^{-|S|}$$

where \mathcal{S}_k is the set of important (or anti-important) (a, b) -cuts that is enumerated in the algorithm. It follows by Property 4 that $t(k) \leq 8^k$. We note that the algorithm can easily be made deterministic by replacing each guessing step by an exhaustive branching; we omit the details.

These two steps give us a 2-approximation algorithm.

► **Theorem 13.** *The exists an algorithm with running time $k^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ such that given an instance of SYMMETRIC DIRECTED VERTEX MULTICUT and an integer k , either it concludes that there is no solution of size at most k , or it returns a solution of size at most $2k$.*

Proof. Let $I = (D, C, k, Y)$ be a SYMMETRIC DIRECTED VERTEX MULTICUT COMPRESSION instance. First, compute a skew multicut of Y using Section 4.2. This gives a set X_0 of at most k vertices, if I has a solution. Then we apply Section 4.3 to find a set X_1 of at most k vertices that is a solution for $(D - X_0, C, k, Y)$. We can now conclude that $X_0 \cup X_1$ is a 2-approximation as $|X_0 \cup X_1| \leq 2k$. \blacktriangleleft

5 An exact algorithm for SYMMETRIC DIRECTED MULTIWAY CUT

In this section, we give an exact (i.e., non-approximate) FPT algorithm for a particular case of SYMMETRIC DIRECTED VERTEX MULTICUT.

SYMMETRIC DIRECTED MULTIWAY VERTEX CUT

Input: A digraph D , a set of terminals $T \subseteq V(D)$, $k \in \mathbb{N}$.

Parameter: k

Output: find, if there exists, $X \subseteq V(D)$ with $|X| \leq k$ such there is no pair of distinct terminals $t, t' \in T \setminus X$ strongly connected in $D - X$.

► **Theorem 14.** SYMMETRIC DIRECTED MULTIWAY VERTEX CUT can be solved in time $2^{\mathcal{O}(k^3)} n^{\mathcal{O}(1)}$.

Actually, we will prove that a more general problem very closely related to DIRECTED SUBSET FEEDBACK ARC SET is FPT. Chitnis et al. [5] proved that the problem DIRECTED SUBSET FEEDBACK ARC SET is FPT. We adapt here their method to the following problem.

ARC TERMINAL SYMMETRIC MULTIWAY CUT

Input: A digraph D having possibly loops, a list A_1, \dots, A_ℓ of arcs in D , such that for every i , $A_i = S_i \times T_i$ for some (not necessarily disjoint) sets S_i and T_i of vertices.

Parameter: k

Output: find, if there exists, a set X of at most k vertices such that any closed walk in $D - X$ intersects at most one A_i .

Note that we allow repetition in the list A_1, \dots, A_ℓ . In this case, if $A_i = A_j$ for some $i \neq j$, then every closed walk intersecting $A_i = A_j$ has to be cut. We will call the arcs in $\bigcup_i A_i$ the *terminal arcs*.

First we show that SYMMETRIC DIRECTED MULTIWAY VERTEX CUT reduces to ARC TERMINAL SYMMETRIC MULTIWAY CUT in FPT time. Indeed, given an instance $I = (D, T = \{t_1, \dots, t_\ell\}, k)$ of SYMMETRIC DIRECTED MULTIWAY VERTEX CUT, we consider the ARC TERMINAL SYMMETRIC MULTIWAY CUT instance $I' = (D, (A_1, \dots, A_\ell), k)$ where $A_i = \{t_i\} \times N_D^+(t_i)$. Now one can easily see that X is a solution for I if and only if it is a solution for I' . Hence it is enough to find an FPT algorithm for ARC TERMINAL SYMMETRIC MULTIWAY CUT.

5.1 Iterative compression and first guesses

By Property 3, it is enough to find an FPT algorithm for the compression problem associated to ARC TERMINAL SYMMETRIC MULTIWAY CUT. Thus suppose that a first solution Y of size $k + 1$ is given, and we want to find a solution X_{OPT} of size at most k . First, we guess the intersection $Y \cap X_{OPT}$, and we remove it. Now we assume that X_{OPT} is disjoint from Y . If two vertices $y, y' \in Y$ are strongly connected in $D - X_{OPT}$, then we can merge them without breaking the solution X_{OPT} , and without making the instance easier. Now we can suppose that no two vertices in Y are strongly connected in $D - X_{OPT}$. Hence there is a topological ordering $y_1, \dots, y_{|Y|}$ of Y such that there is no (y_j, y_i) -path in $D - X_{OPT}$ if $j > i$. Given this ordering, we can add the arc $y_i y_j$ for every $i < j$ without breaking the solution X_{OPT} , and without making the instance easier. To summarise, by multiplying the running time of the algorithm by at most $(k + 2)^{k+1} n^{\mathcal{O}(1)}$, it is enough to find an FPT algorithm for the following problem.

ARC TERMINAL SYMMETRIC MULTIWAY CUT COMPRESSION

Input: A digraph D (having possibly loops), a list A_1, \dots, A_ℓ of arcs in D , such that for every i , $A_i = S_i \times T_i$ for some (not necessarily disjoint) sets S_i and T_i of vertices, and an ordered set $Y = (y_1, \dots, y_r)$ of vertices such that:

1. for every $i \neq j$, no closed walk in $D - Y$ intersects both A_i and A_j , and
2. for every $1 \leq i < j \leq r$, $y_i y_j$ is an arc in D .

Parameter: $k + r$

Output: find, if there exists, a set X of at most k vertices such that

1. X is disjoint from Y ,
2. any closed walk in $D - X$ intersects at most one A_i , and
3. there is no (y_j, y_i) -path in $D - X$ if $j > i$.

5.2 Shadow removal

Let $I = (D, (A_1, \dots, A_\ell), k, Y)$ be an ARC TERMINAL SYMMETRIC MULTIWAY CUT COMPRESSION instance. To show that we can assume the solution to be shadowless, let \mathcal{F} be the family containing all closed walks intersecting at least two distinct sets A_i, A_j and all (y_j, y_i) -walks for $j > i$. Note that \mathcal{F} is Y -connected and that the problem is precisely to find an \mathcal{F} -transversal X disjoint from Y . We apply Theorem 5 with \mathcal{F} , giving us a family of $t = 2^{\mathcal{O}(k^2)} \log^2 n$ sets disjoint from Y , and we guess one of them, say Z , to be such that if I has a solution, then there exists a solution X disjoint from Z and with shadow contained in Z . As we consider the shadow from Y , vertices in Y can not be in the shadow of a solution, so we can assume Z and Y disjoint by replacing Z by $Z \setminus Y$.

We now define another instance $I/Z = (D', (A'_1, \dots, A'_\ell), k, Y)$ equivalent to I in the following sense:

1. if I has a solution that is disjoint from Z and with shadow contained in Z , then I/Z has a shadowless solution, and
2. if I/Z has a solution, then I does too.

The construction is the following. If $D[Z]$ contains a closed walk W such that at least two A_i, A_j intersects W , reject Z . Otherwise construct the following. Let a Z -walk be a walk in D with endpoints in $V(D')$ and internal vertices, if any, in Z .

- $V(D') = V(D) \setminus Z$;
- $E(D')$ is the set of all arcs uv such that there is a Z -walk from u to v in D ;
- for every $i = 1, \dots, \ell$, A'_i is the set of arcs uv such that there is a Z -walk from u to v intersecting A_i . In particular, $A_i \cap E(D') \subseteq A'_i$ as a Z -walk can have no internal vertices.

First, we need to check that I/Z is indeed an instance of ARC TERMINAL SYMMETRIC MULTIWAY CUT COMPRESSION

▷ **Claim 15.** For every $i = 1, \dots, \ell$, $A'_i = S'_i \times T'_i$ for some sets S'_i and T'_i of vertices.

Proof. It is enough to show that if $uv, u'v' \in A'_i$, then $uv' \in A'_i$. By definition, there exists a Z -walk W (resp. W') from u to v (resp. u' to v'), with possibly no internal vertices, which goes through a terminal arc $ab \in A_i$ (resp. $a'b' \in A_i$), where the terminal arc may be a loop. As $A_i = S_i \times T_i$, we have $ab' \in A_i$, and so by combining a prefix of W with a suffix of W' , there is a Z -walk from u to v' containing an arc in A_i . This shows that $uv' \in A'_i$. ◁

▷ **Claim 16.** I/Z is an instance of ARC TERMINAL SYMMETRIC MULTIWAY CUT COMPRESSION.

Proof. By Claim 15, $A'_i = S'_i \times T'_i$ for every i , and the arcs $y_i y_j$, $i < j$ remain in D' . It remains to check that Y is a solution for D' . Assume to the contrary, and let W be a closed walk in $D' - Y$ intersecting two sets A_i and A_j , $i \neq j$. But then W expands into a closed walk W' in D by replacing every arc of W with a corresponding Z -walk. Since $Y \cap Z = \emptyset$, this is a closed walk in D intersecting A_i and A_j , disjoint from Y . This is a contradiction. \triangleleft

\triangleright Claim 17. If I has a solution disjoint from Z and with shadow contained in Z , then I/Z has a shadowless solution.

Proof. Let X be a solution of I disjoint from Z and with shadow contained in Z . We claim that X is a shadowless solution of I/Z .

First, let's see why X is a solution of I/Z . Suppose for contradiction that $D' - X$ contains a closed walk W' containing two terminal arcs $uv \in A'_i$ and $u'v' \in A'_j$ for some distinct indices i and j . Then we construct a closed walk W in $D - X$ intersecting both A_i and A_j : replace in W' the arc uv (resp. $u'v'$) by a Z -walk from u to v (resp. u' to v') intersecting A_i (resp. A_j), and for every other arc $xy \in W'$ which is not in D , replace xy by a Z -walk from x to y . This gives a closed walk W in $D - X$ intersecting both A_i and A_j , contradicting the fact that X is a solution of I . Similarly, if there is a (y_j, y_i) -path P' in $D' - X$ for some $j > i$, then we can expand P' into a (y_j, y_i) -walk W in $D - X$, which can be shortcut into a (y_j, y_i) -path P in $D - X$.

Now we show that X is shadowless in I' . For every vertex $u \in V(D) \setminus Z$, we know that there is a (u, Y) -path P^+ (resp. (Y, u) -path P^-) in $D - X$, as the shadow of X is included in Z . Then we replace every Z -walk in P^+ (resp. P^-) by the arc linking its endpoints. This gives a (u, Y) -path (resp. (Y, u) -path) in $D' - X$, and so v is not in the shadow. This proves that X is shadowless in D' . \triangleleft

\triangleright Claim 18. If I/Z has a solution then I too.

Proof. Suppose that I/Z has a solution X . We claim that X is a solution for I too.

Suppose for contradiction that $D - X$ has a closed walk W intersecting both A_i and A_j for some distinct indices i and j . Then construct the closed walk W' in $D' - X$ as follows: replace every Z -walk in W by the arc linking its endpoints. This creates a closed walk W' in $D' - X$ intersecting both A'_i and A'_j , contradicting the fact that X is a solution for I' . A similar step applies if $D - X$ contains a (y_j, y_i) -path for some $j > i$. \triangleleft

As a consequence, we are able to transform the original instance I into an equivalent instance I/Z which has a shadowless solution. Guessing Z multiplies the running time by at most $2^{\mathcal{O}(k^2)} \log^2 n$, and then computing I/Z is performed in polynomial time.

5.3 Finding a shadowless solution

We now suppose that $I = (D, (A_1, \dots, A_\ell), k, Y)$ has a shadowless solution X_{OPT} . Remember that y_1, \dots, y_r is an ordering of Y such that there is no (y_j, y_i) -path in $D - X_{OPT}$ if $j > i$, and for every $j > i$, $y_i y_j$ is an arc in D . As the solution X_{OPT} we are searching for is shadowless, every vertex in $D - X_{OPT}$ reaches Y , and so y_r (because y_r is dominated by $Y \setminus \{y_r\}$).

Another observation is that for at most one index i_0 , A_{i_0} contains a terminal arc strongly connected with y_r in $D - X_{OPT}$. In what follows, we implicitly suppose that i_0 exists, otherwise we can set by convention $A_{i_0} = \emptyset$. As X_{OPT} is shadowless, an arc uv is strongly connected with y_r in $D - X_{OPT}$ if and only if

1. y_r reaches u in $D - X_{OPT}$ and
2. $v \notin X_{OPT}$.

11:14 On the Parameterized Complexity of Symmetric Directed Multicut

The next claim allows us to find the set of vertices v which violates the second condition. Let R denote the set of vertices reachable from y_r in $D - X_{OPT}$ and note by shadowlessness that R precisely describes the strongly connected component of y_r in $D - X_{OPT}$. Say that A_i is *active in X_{OPT}* if $i \neq i_0$ and $S_i \cap R \neq \emptyset$ (and note that this implies $T_i \subseteq X_{OPT}$).

▷ **Claim 19** (Derived from Theorem 5.4 [5]). One can find in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ a collection of pairs (I, T_c) where $I \subseteq [\ell]$ and $T_c \subseteq V(D)$, such that the following hold:

1. the number of pairs (I, T_c) produced is $k^{\mathcal{O}(1)} \log n$
2. for every pair, $|I| + |T_c| \leq (2k + 1)4^{2k+1}$
3. for at least one pair (I, T_c) we have $i_0 \in I$ if $A_{i_0} \neq \emptyset$, and for every $i \in [\ell]$ such that A_i is active in X_{OPT} we have $T_i \subseteq T_c$

Proof. Assume that $A_{i_0} \neq \emptyset$ as otherwise the result is easier, and let $uv \in A_{i_0}$ with $u, v \in R$. We begin by computing a subset $U \subseteq V(D)$ such that $v \in U$ and $U \cap X_{OPT} = \emptyset$. This can be done randomly with success probability $\Theta(1/k)$ by sampling every vertex independently with probability $1/k$, but the process can also be derandomized by a (n, k, k^2) -splitter; see Naor et al. [18]. In particular, in polynomial time we can compute a family of subsets $U_i \subseteq V(D)$ such that the family contains $k^{\mathcal{O}(1)} \log n$ members and at least one member meets the conditions for U . We repeat the steps below for every member U_i in the family.

From now on, let us assume that we have such a set U . Create a graph D' as follows. For every $v \in V(D)$, create two vertices v^-, v^+ . For every $i \in [\ell]$, create a vertex z_i and add the arcs $\{u^+ z_i \mid u \in S_i\}$ and $\{z_i v^- \mid v \in T_i\}$. For every arc $uv \in E(D)$, add the arc $u^+ v^+$. Finally, add vertices s and t , the arc sy_r^+ , and the arc $v^- t$ for every $v \in V(D)$. Finally, for every vertex $v \in U$ give v^- capacity $2k + 2$ by replacing v^- by a set of $2k + 2$ false twins. Let T'_c be the union of all important (s, t) -cuts in D' of size at most $2k + 1$. By Property 4, T'_c can be computed in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ and $|T'_c| \leq (2k + 1)4^{2k+1}$. Finally we set $I = \{i \mid z_i \in T'_c\}$ and $T_c = \{v \in V(D) \mid v^- \in T'_c\}$. Clearly $|I| + |T_c| \leq |T'_c| \leq (2k + 1)4^{2k+1}$.

We claim that I contains i_0 , and that for every A_i that is active in X_{OPT} we have $T_i \subseteq T_c$. Indeed, define the set $X' = \{v^-, v^+ \mid v \in X_{OPT}\} \cup \{z_{i_0}\}$ and recall by assumption that $X_{OPT} \cap U = \emptyset$. Note that X' is an (s, t) -cut. Indeed, assume to the contrary that there is an (s, t) -path P in $D' - X'$. Then the last arcs of P must be $u^+ z_i$, $z_i v^-$ and $v^- t$ for some $i \in [\ell]$, $uv \in A_i$. We may also assume that the entire prefix of P before z_i visits only s and vertices w^+ , $w \in V(D)$. But then that prefix proves $u \in R$; $z_i \notin X'$ implies $i \neq i_0$; and $v^- \notin X'$ implies $v \notin X_{OPT}$. This contradicts that only A_{i_0} is strongly connected to y_r in $D - X_{OPT}$. Also note $|X'| \leq 2k + 1$. Now by Property 2 we can push X' to an important (s, t) -cut X'' of size at most $2k + 1$, hence $X'' \subseteq T'_c$.

We claim that $z_{i_0} \in X''$ and for every A_i active in X_{OPT} we have $\{v^- \mid v \in T_i\} \subseteq X''$. For the former, by assumption $u \in R$, hence either $z_{i_0} \in X''$ or the cut has been pushed closer to t . But since $v \in U$ and v has been given high capacity, pushing the cut past z_{i_0} would contradict the size bound of $2k + 1$. Hence $z_{i_0} \in X''$. For the latter, assume that A_i is active in X_{OPT} . Then there is a vertex $u' \in S_i \cap R$, hence $z_i \in R$, and the cut cannot push past the vertices $v^-, v \in T_i$ since $v^- t \in E(D')$. ◁

Now we can guess the correct pair (I, T_c) . Therefore, we can guess $i_0 \in I$ (or the case that $A_{i_0} = \emptyset$) and $X_{OPT} \cap T_c$, and remove these vertices from D . This multiplies the running time by at most $(2k + 1)4^{2k+1} \binom{(2k+1)4^{2k+1}}{k} \log n = 2^{\mathcal{O}(k^2)} \log n$, and now we can assume that for every $i \in [\ell]$ except i_0 , A_i is not active. Furthermore, if $A_{i_0} \neq \emptyset$ then we add all arcs $\{y_r\} \times T_{i_0}$ to the graph. Next claim shows how to start the construction of a solution using these assumptions.

▷ **Claim 20.** Adding the arcs $\{y_r\} \times T_{i_0}$ does not affect the solution. Furthermore, let S be the set of vertices in X_{OPT} which have an in-neighbour reachable from y_r in $D - X_{OPT}$. There exists an important $(\{y_r\}, Y \setminus \{y_r\} \cup \bigcup_{i \neq i_0} S_i)$ -cut S' of size at most $|S|$ such that $(X_{OPT} \setminus S) \cup S'$ is a solution to I .

Proof. We first note that since $R \cap S_{i_0} \neq \emptyset$, then for every $v \in T_{i_0}$ either $v \in R$ or $v \in X_{OPT}$ (for example due to blocking paths from y_r to some y_i , $i < r$). Hence adding the arcs $\{y_r\} \times T_{i_0}$ has no effect on the solution. However, it does simplify the important separator step below.

Now observe that S is a $(\{y_r\}, Y \setminus \{y_r\} \cup \bigcup_{i \neq i_0} S_i)$ -cut. By Property 2, there exists an important $(\{y_r\}, Y \setminus \{y_r\} \cup \bigcup_{i \neq i_0} S_i)$ -cut S' with $|S'| \leq |S|$ such that every vertex reachable from y_r in $D - S$ is still reachable from y_r in $D - S'$. We prove that $X' := (X_{OPT} \setminus S) \cup S'$ is a solution for I . Clearly $|X'| \leq k$, so we only need to show that X' cuts all the closed walks intersecting several of the sets A_1, \dots, A_ℓ and all (y_j, y_i) -paths, $j > i$.

Suppose for contradiction that there exists two distinct indices $i \neq j$ and a closed walk W such that W intersects both A_i and A_j . First, $i \neq i_0$ and $j \neq i_0$: since the arc $y_r v$ is added for every $v \in T_{i_0}$, either $v \in X_{OPT}$ or $v \in R$. Thus there is no path from T_{i_0} to S_i for any $i \neq i_0$ in $D - X'$ by the choice of the cut S' . Moreover, W must intersect S , as otherwise W is a closed walk in $D - X_{OPT}$, contradicting the fact that X_{OPT} is a solution. Let s be a vertex in $S \cap W$, then either $s \in S'$, and so S' intersects W ; or s is reachable from y_r in $D - S'$. But then S_i is reachable from y_r in $D - S'$, contradicting the fact that S' is an $(y_r, \bigcup_{i \neq i_0} S_i)$ -cut. This contradiction proves that X' is a solution. By a similar argument, X' also cuts all (y_j, y_i) -paths for $j > i$. ◁

Note that $(X_{OPT} \setminus S) \cup S'$ might have a non empty shadow. This is not a problem as we will apply the shadow removal procedure at each step.

We can now give the algorithm \mathcal{A}' on the instance $(D, (A_i), k, Y)$ of ARC TERMINAL SYMMETRIC DIRECTED MULTIWAY CUT COMPRESSION:

1. reduce to the shadowless case by applying Subsection 5.2;
2. compute (and guess) (I, T_c) with Claim 19, guess $i_0 \in I \cup \{0\}$ and $X_c := X_{OPT} \cap T_c \subseteq T_c$;
3. let $D' = D - X_c$, and if $i_0 \neq 0$, add all arcs $\{y_r\} \times T_{i_0}$;
4. guess an important $(\{y_r\}, Y \setminus \{y_r\} \cup \bigcup_{i \neq i_0} S_i)$ -cut S of size at most $k - |X_c|$ in D' ;
5. if $\mathcal{A}'(D - S - X_c, (A_i), k - |S| - |X_c|, Y \setminus \{y_r\})$ returns a solution X' , return $S \cup X_c \cup X'$; otherwise proceed with the next guess or return “no solution”.

First, it is easy to see that if this algorithm returns a set X , then X is a solution of the input instance. Moreover, by all the previous claims, if there exists a solution, then there exists a sequence of guesses which will find it. This algorithm explores a tree of depth at most k with maximum degree $2^{\mathcal{O}(k^2)} \log^3 n$, and each node is processed in time $2^{\mathcal{O}(k^2)} n^{\mathcal{O}(1)}$. Hence the total running time is at most

$$\left(2^{\mathcal{O}(k^2)} \log^3 n\right)^k 2^{\mathcal{O}(k^2)} n^{\mathcal{O}(1)} = 2^{\mathcal{O}(k^3)} n^{\mathcal{O}(1)}$$

using in particular Lemma 21 from the appendix. This completes the proof of Theorem 14.

References

- 1 Amit Agarwal, Noga Alon, and Moses Charikar. Improved approximation for directed cut problems. In *STOC*, pages 671–680. ACM, 2007.
- 2 Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. *SIAM J. Comput.*, 47(1):166–207, 2018.

- 3 Jianer Chen, Yang Liu, Songjian Lu, Barry O’Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5), 2008. doi:10.1145/1411509.1411511.
- 4 Joseph Cheriyan, Howard J. Karloff, and Yuval Rabani. Approximating directed multicuts. *Comb.*, 25(3):251–269, 2005.
- 5 Rajesh Hemant Chitnis, Marek Cygan, Mohammad Taghi Hajiaghayi, and Dániel Marx. Directed subset feedback vertex set is fixed-parameter tractable. *ACM Trans. Algorithms*, 11(4):28:1–28:28, 2015.
- 6 Rajesh Hemant Chitnis, Mohammad Taghi Hajiaghayi, and Dániel Marx. Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. *SIAM J. Comput.*, 42(4):1674–1696, 2013.
- 7 Julia Chuzhoy and Sanjeev Khanna. Polynomial flow-cut gaps and hardness of directed cut problems. *J. ACM*, 56(2):6:1–6:28, 2009.
- 8 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.
- 9 Guy Even, Joseph Naor, Satish Rao, and Baruch Schieber. Divide-and-conquer approximation algorithms via spreading metrics. *J. ACM*, 47(4):585–616, 2000.
- 10 Anupam Gupta. Improved results for directed multicut. In *SODA*, pages 454–455. ACM/SIAM, 2003.
- 11 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Directed flow-augmentation. In *STOC*, pages 938–947. ACM, 2022.
- 12 Philip N. Klein, Serge A. Plotkin, Satish Rao, and Éva Tardos. Approximation algorithms for steiner and directed multicuts. *J. Algorithms*, 22(2):241–269, 1997.
- 13 Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Magnus Wahlström. Fixed-parameter tractability of multicut in directed acyclic graphs. *SIAM J. Discret. Math.*, 29(1):122–144, 2015.
- 14 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. In *SODA*, pages 2181–2200. SIAM, 2020.
- 15 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.
- 16 Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006. Parameterized and Exact Computation. doi:10.1016/j.tcs.2005.10.007.
- 17 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset, 2013. arXiv:1010.3633.
- 18 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *FOCS*, pages 182–191. IEEE Computer Society, 1995.
- 19 Marcin Pilipczuk and Magnus Wahlström. Directed multicut is W[1]-hard, even for four terminal pairs. *ACM Trans. Comput. Theory*, 10(3):13:1–13:18, 2018. doi:10.1145/3201775.
- 20 M. S. Ramanujan and Saket Saurabh. Linear-time parameterized algorithms via skew-symmetric multicuts. *ACM Trans. Algorithms*, 13(4):46:1–46:25, 2017.
- 21 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.

A Missing proofs

Proof of Proposition 3. Let $A'(D, T, k)$ be an algorithm solving the problem \mathcal{L}' in time $t(k, |Y|)n^c$. We now solve the original problem \mathcal{L} as follows. Consider an arbitrary ordering v_1, \dots, v_n of $V(D)$. We will compute iteratively a set $X_i \subseteq \{v_1, \dots, v_i\}$ of size at most αk which is a solution of the partial instance I_i induced by $\{v_1, \dots, v_i\}$.

We start with $X_0 = \emptyset$, which is a solution of I_0 by assumption. Then, if V_i is already computed, we apply \mathcal{A}' to $(D[\{v_1, \dots, v_{i+1}\}], T, X_i \cup \{v_{i+1}\}, k)$, which returns by assumption a solution of size at most αk , or says that there is no solution of size at most k , and in this latter case we return "no" directly. This call is valid because $X_i \cup \{v_{i+1}\}$ is a solution of $(D[\{v_1, \dots, v_{i+1}\}], T, X_i \cup \{v_{i+1}\})$ of size at most $\alpha k + 1$.

This algorithm consists in n calls to \mathcal{A}' with the solution to compress of size at most $\alpha k + 1$. Hence its running time is at most $t(k, \alpha k + 1)n^{c+1}$. ◀

► **Lemma 21.** *If $n \geq 2^{16}$ and $p \geq 0$, then $(\log n)^p \leq n + p^{2p}$.*

Proof. If $p \geq \sqrt{\log n}$ then $n \leq 2^{p^2}$ and $(\log n)^p \leq p^{2p}$.

Otherwise, $p < \sqrt{\log n}$. First, we show the following property:

$$n \geq 2^{16} \Rightarrow \sqrt{\log n} \leq \frac{\log n}{\log \log n}$$

To prove that, note that this property is equivalent to $2 \log N \leq N$ with $N = \sqrt{\log n}$. Then $N \geq 4$ is a sufficient condition, and $n \geq 2^{16}$ too. Now we apply this result and we get $p \leq \sqrt{\log n} \leq \frac{\log n}{\log \log n}$. It follows that $(\log n)^p \leq n$. ◀

Computing Generalized Convolutions Faster Than Brute Force

Barış Can Esmer ✉ 

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Ariel Kulik ✉

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Dániel Marx ✉ 

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Philipp Schepper ✉ 

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Karol Węgrzycki ✉ 

Saarland University, Saarbrücken, Germany

Max Planck Institute for Informatics, Saarbrücken, Germany

Abstract

In this paper, we consider a general notion of convolution. Let D be a finite domain and let D^n be the set of n -length vectors (tuples) of D . Let $f: D \times D \rightarrow D$ be a function and let \oplus_f be a coordinate-wise application of f . The f -CONVOLUTION of two functions $g, h: D^n \rightarrow \{-M, \dots, M\}$ is

$$(g \otimes_f h)(\mathbf{v}) := \sum_{\substack{\mathbf{v}_g, \mathbf{v}_h \in D^n \\ \text{s.t. } \mathbf{v} = \mathbf{v}_g \oplus_f \mathbf{v}_h}} g(\mathbf{v}_g) \cdot h(\mathbf{v}_h)$$

for every $\mathbf{v} \in D^n$. This problem generalizes many fundamental convolutions such as Subset Convolution, XOR Product, Covering Product or Packing Product, etc. For arbitrary function f and domain D we can compute f -CONVOLUTION via brute-force enumeration in $\tilde{O}(|D|^{2n} \cdot \text{polylog}(M))$ time.

Our main result is an improvement over this naive algorithm. We show that f -CONVOLUTION can be computed exactly in $\tilde{O}((c \cdot |D|^2)^n \cdot \text{polylog}(M))$ for constant $c := 5/6$ when D has even cardinality. Our main observation is that a *cyclic partition* of a function $f: D \times D \rightarrow D$ can be used to speed up the computation of f -CONVOLUTION, and we show that an appropriate cyclic partition exists for every f .

Furthermore, we demonstrate that a single entry of the f -CONVOLUTION can be computed more efficiently. In this variant, we are given two functions $g, h: D^n \rightarrow \{-M, \dots, M\}$ alongside with a vector $\mathbf{v} \in D^n$ and the task of the f -QUERY problem is to compute integer $(g \otimes_f h)(\mathbf{v})$. This is a generalization of the well-known Orthogonal Vectors problem. We show that f -QUERY can be computed in $\tilde{O}(|D|^{\frac{\omega}{2}n} \cdot \text{polylog}(M))$ time, where $\omega \in [2, 2.373)$ is the exponent of currently fastest matrix multiplication algorithm.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Algorithm design techniques

Keywords and phrases Generalized Convolution, Fast Fourier Transform, Fast Subset Convolution

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.12

Related Version *Full Version*: <https://arxiv.org/abs/2209.01623> [22]

Funding Research supported by the European Research Council (ERC) consolidator grant No. 725978 SYSTEMATICGRAPH and the project TIPEA (grant No. 850979).

Acknowledgements We would like to thank Karl Bringmann and Jesper Nederlof for useful discussions. Barış Can Esmer and Philipp Schepper are part of Saarbrücken Graduate School of Computer Science, Germany.



© Barış Can Esmer, Ariel Kulik, Dániel Marx, Philipp Schepper, and Karol Węgrzycki; licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 12; pp. 12:1–12:22

Leibniz International Proceedings in Informatics



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



1 Introduction

Convolutions occur naturally in many algorithmic applications, especially in the exact and parameterized algorithms. The most prominent example is a subset convolution procedure [23, 37], for which an efficient $\tilde{O}(2^n \cdot \text{polylog}(M))$ time algorithm for subset convolution dates back to Yates [40] but in the context of exact algorithms it was first used by Björklund et al. [6].¹ Researchers considered a plethora of other variants of convolutions, such as: Cover Product, XOR Product, Packing Product, Generalized Subset Convolution, or Discriminantal Subset Convolution [6, 8, 7, 10, 35, 21, 11]. These subroutines are crucial ingredients in the design of efficient algorithms for many exact and parameterized algorithms such as Hamiltonian Cycle, Feedback Vertex Set, Steiner Tree, Connected Vertex Cover, Chromatic Number, Max k -Cut or Bin Packing [20, 10, 41, 28, 5, 39]. These convolutions are especially useful for dynamic programming algorithms on tree decompositions and occur naturally during join operations (e.g., [35, 20, 34]). Usually, in the process of algorithm design, the researcher needs to design a different type of convolution from scratch to solve each of these problems. Often this is a highly technical and laborious task. Ideally, we would like to have a single tool that can be used as a blackbox in all of these cases. This motivates the following ambitious goal in this paper:

Goal: Unify convolution procedures under one general umbrella.

Towards this goal, we consider the problem of computing f -Generalized Convolution (f -CONVOLUTION) introduced by van Rooij [34]. Let D be a finite domain and let D^n be the n length vectors (tuples) of D . Let $f: D \times D \rightarrow D$ be an arbitrary function and let \oplus_f be a coordinate-wise application of the function f .² For two functions $g, h: D^n \rightarrow \mathbb{Z}$ the f -CONVOLUTION, denoted by $(g \otimes_f h): D^n \rightarrow \mathbb{Z}$, is defined for all $\mathbf{v} \in D^n$ as

$$(g \otimes_f h)(\mathbf{v}) := \sum_{\substack{\mathbf{v}_g, \mathbf{v}_h \in D^n \\ \text{s.t. } \mathbf{v} = \mathbf{v}_g \oplus_f \mathbf{v}_h}} g(\mathbf{v}_g) \cdot h(\mathbf{v}_h).$$

Here we consider a standard $\mathbb{Z}(+, \cdot)$ ring. Through the paper we assume that M is the absolute value of the maximum integer given on the input.

In the f -CONVOLUTION problem the functions $g, h: D^n \rightarrow \{-M, \dots, M\}$ are given as an input and the output is the function $(g \otimes_f h)$. Note, that the input and output of the f -CONVOLUTION problem consist of $3 \cdot |D|^n$ integers. Hence it is conceivable that f -CONVOLUTION could be solved in $\tilde{O}(|D|^n \cdot \text{polylog}(M))$. Such a result for arbitrary f would be a real breakthrough in how we design parameterized algorithms. So far, however, researchers have focused on characterizing functions f for which f -CONVOLUTION can be solved in $\tilde{O}(|D|^n \cdot \text{polylog}(M))$ time. In [34] van Rooij considered specific instances of this setting, where for some constant $r \in \mathbb{N}$ the function f is defined as either (i) standard addition: $f(x, y) := x + y$, or (ii) addition with a maximum: $f(x, y) := \min(x + y, r - 1)$, or (iii) addition modulo r , or (iv) maximum: $f(x, y) := \max(x, y)$. Van Rooij [34] showed that for these special cases the f -CONVOLUTION can be solved in $\tilde{O}(|D|^n \cdot \text{polylog}(M))$ time. His results allow the function f to differ between coordinates. A recent result regarding generalized

¹ We use $\tilde{O}(\cdot)$ notation to hide polylogarithmic factors. We assume that M is the maximum absolute value of the integers on the input.

² We provide a formal definition of \oplus_f in Section 2.

Discrete Fourier Transform [32] can be used in conjunction with Yates algorithm [40] to compute f -CONVOLUTION in $\tilde{O}(|D|^{\omega \cdot n/2} \cdot \text{polylog}(M))$ time when f is a finite-group operation and ω is the exponent of the currently fastest matrix-multiplication algorithms.³ To the best of our knowledge these are the most general settings where convolution has been considered so far.

Nevertheless, for an arbitrary function f , to the best of our knowledge the state-of-the-art for f -CONVOLUTION is a straightforward quadratic time enumeration.

Question 1: Is the naive $\tilde{O}(|D|^{2n} \cdot \text{polylog}(M))$ algorithm for f -CONVOLUTION optimal?

Similar questions were studied from the point of view of the Fine-Grained Complexity. In that setting the focus is on convolutions with sparse representations, where the input size is only the size of the support of the functions g and h . It is conjectured that even subquadratic algorithms are highly unlikely for these representations [19, 25]. However, these lower bounds do not answer Question 1, because they are highly dependent on the sparsity of the input.

1.1 Our Results

We provide a positive answer to Question 1 and show an exponential improvement (in n) over a naive $\tilde{O}(|D|^{2n} \cdot \text{polylog}(M))$ algorithm for every function f .

► **Theorem 1.1 (Generalized Convolution).** *Let D be a finite set and $f: D \times D \rightarrow D$. There is an algorithm for f -CONVOLUTION with the following running time $\tilde{O}((\frac{5}{6} \cdot |D|^2)^n \cdot \text{polylog}(M))$ when $|D|$ is even, or $\tilde{O}((\frac{5}{6} \cdot |D|^2 + \frac{1}{6} \cdot |D|)^n \cdot \text{polylog}(M))$ when $|D|$ is odd.*

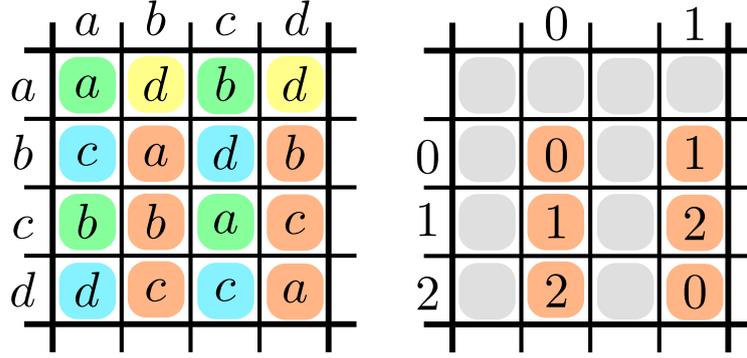
Observe that the running time obtained by Theorem 1.1 improves upon the brute-force for every $|D| \geq 2$. Our technique works in a more general setting when $g: L^n \rightarrow \mathbb{Z}$ and $h: R^n \rightarrow \mathbb{Z}$ and $f: L \times R \rightarrow T$ for arbitrary domains L, R and T (see Section 2 for the exact running time dependence).

Our Technique: Cyclic Partition. Now, we briefly sketch the idea behind the proof of Theorem 1.1. We say that a function is k -cyclic if it can be represented as an addition modulo k (after relabeling the entries of the domain and image). These functions are somehow simple, because as observed in [34, 33] f -CONVOLUTION can be computed in $\tilde{O}(k^n \cdot \text{polylog}(M))$ time if f is k -cyclic. In a nutshell, our idea is to partition the function $f: D \times D \rightarrow D$ into cyclic functions and compute the convolution on these parts independently.

More formally, a cyclic minor of the function $f: D \times D \rightarrow D$ is a (combinatorial) rectangle $A \times B$ with $A, B \subseteq D$ and a number $k \in \mathbb{N}$ such that f restricted to A, B is a k -cyclic function. The cost of the cyclic minor (A, B, k) is $\text{cost}(A, B) := k$. A cyclic partition is a set $\{(A_1, B_1, k_1), \dots, (A_m, B_m, k_m)\}$ of cyclic minors such that for every $(a, b) \in D \times D$ there exists a unique $i \in [m]$ with $(a, b) \in A_i \times B_i$. The cost of the cyclic partition $\mathcal{P} = \{(A_1, B_1, k_1), \dots, (A_m, B_m, k_m)\}$ is $\text{cost}(\mathcal{P}) := \sum_{i=1}^m k_i$. See Figure 1.1 for an example of a cyclic partition.

Our first technical contribution is an algorithm to compute f -CONVOLUTION when the cost of a cyclic partition is small.

³ This observation was brought to our attention by Jesper Nederlof [27].



■ **Figure 1.1** Left figure illustrates exemplar function $f: D \times D \rightarrow D$ over domain $D := \{a, b, c, d\}$. We highlighted a cyclic partition with red, blue, yellow and blue colors. Each color represents a different minor of f . On the right figure we demonstrate that the red-highlighted minor can be represented as addition modulo 3 (after relabeling $a \mapsto 0, b \mapsto 1$ and $c \mapsto 2$). Hence the red minor has cost 3. The reader can further verify that green and blue minors have cost 2 and yellow minor has cost 1, hence the cost of that particular partition is $3 + 2 + 2 + 1 = 8$.

► **Lemma 1.2** (Algorithm for f -CONVOLUTION). *Let D be an arbitrary finite set, $f: D \times D \rightarrow D$ and let \mathcal{P} be the cyclic partition of f . Then there exists an algorithm which given $g, h: D^n \rightarrow \mathbb{Z}$ computes $(g \otimes_f h)$ in $\tilde{O}((\text{cost}(\mathcal{P})^n + |D|^n) \cdot \text{polylog}(M))$ time.*

The idea behind the proof of Lemma 1.2 is as follows. Based on the partition \mathcal{P} , for any pair of vectors $\mathbf{u}, \mathbf{w} \in D^n$, we can define a type $\bar{p} \in [m]^n$ such that $(\mathbf{u}_i, \mathbf{w}_i) \in A_{\bar{p}_i} \times B_{\bar{p}_i}$ for every $i \in [n]$. Our main idea is to go over each type \bar{p} and compute the sum in the definition of f -CONVOLUTION only for pairs $(\mathbf{v}_g, \mathbf{v}_h)$ that have type \bar{p} . In order to do this, first we select the vectors \mathbf{v}_g and \mathbf{v}_h that are compatible with this type \bar{p} . For instance, consider the example in Figure 1.1. Whenever \bar{p}_i refers to, say, the red-colored minor, then we consider \mathbf{v}_g only if its i -th coordinate is in $\{b, c, d\}$ and consider \mathbf{v}_h only if its i -th coordinate is in $\{b, d\}$. After computing all these vectors \mathbf{v}_g and \mathbf{v}_h , we can transform them according to the cyclic minor at each coordinate. Continuing our example, as the red-colored minor is 3-cyclic, we can represent the i -th coordinate of \mathbf{v}_g and \mathbf{v}_h as $\{0, 1, 2\}$ and then the problem reduces to addition modulo 3 at that coordinate. Therefore, using the algorithm of van Rooij [34] for cyclic convolution we can handle all pairs of type \bar{p} in $\tilde{O}((\prod_{i=1}^n k_{\bar{p}_i}) \cdot \text{polylog}(M))$ time. As we go over all m^n types \bar{p} the sum of m^n terms is

$$\sum_{\bar{p} \in [m]^n} \left(\prod_{i=1}^n k_{\bar{p}_i} \right) = \left(\sum_{i=1}^m k_i \right)^n = \text{cost}(\mathcal{P})^n.$$

Hence, the overall running time is $\tilde{O}(\text{cost}(\mathcal{P})^n \cdot \text{polylog}(M))$. This running time evaluation ignores the generation of the vectors given as input for the cyclic convolution algorithm. The efficient computation of these vectors is nontrivial and requires further techniques that we explain in Section 3.

It remains to provide the low-cost cyclic partition of an arbitrary function f .

► **Lemma 1.3.** *For any finite set D and any function $f: D \times D \rightarrow D$ there is a cyclic partition \mathcal{P} of f such that $\text{cost}(\mathcal{P}) \leq \frac{5}{6}|D|^2$ when $|D|$ is even, or $\text{cost}(\mathcal{P}) \leq \frac{5}{6}|D|^2 + \frac{1}{6}|D|$ when $|D|$ is odd.*

For the sake of presentation let us assume that $|D|$ is even. In order to show Lemma 1.3, we partition D into pairs A_1, \dots, A_k where $k := |D|/2$ and consider the restrictions of f to $A_j \times D$ one by one. This allows us to encode f on $A_j \times D$ as a directed graph G with $|D|$

edges and $|D|$ vertices. We observe that for certain classes of subgraphs (i.e., paths, out-stars, in-stars, and cycles) there is a corresponding cyclic minor. Our goal is to partition this graph G into such subgraphs in a way that the total cost of the resulting cyclic partition is small. Following this argument, the proof of Lemma 1.3 becomes a graph theoretic analysis. The proof of Lemma 1.3 is included in Section 4. Our method applies for more general functions $f: L \times R \rightarrow T$, where domains L, R, T can be different and have arbitrary cardinality. We note that a weaker variant of Lemma 1.3 in which the guarantee is $\text{cost}(\mathcal{P}_f) \leq \frac{7}{8}|D|^2$ is easier to attain.

Efficient Algorithm for Convolution Query. Our next contribution is an efficient algorithm to query a single value of f -CONVOLUTION. In the f -QUERY problem, the input is $g, h: D^n \rightarrow \mathbb{Z}$ and a single vector $\mathbf{v} \in D^n$. The task is to compute a value $(g \otimes_f h)(\mathbf{v})$. Observe that this task generalizes⁴ the fundamental problem of Orthogonal Vectors. We show that computing f -QUERY is much faster than computing the full output of f -CONVOLUTION.

► **Theorem 1.4** (Convolution Query). *For any finite set D and function $f: D \times D \rightarrow D$ there is a $\tilde{O}(|D|^{\omega \cdot n/2} \cdot \text{polylog}(M))$ time algorithm for the f -QUERY problem.*

Here $\tilde{O}(n^\omega \cdot \text{polylog}(M))$ is the time needed to multiply two $n \times n$ integer matrices with values in $\{-M, \dots, M\}$ and currently $\omega \in [2, 2.373]$ [2]. Note, that under the assumption that two matrices can be multiplied in the linear in the input time (i.e., $\omega = 2$) then Theorem 1.4 runs in the nearly-optimal $\tilde{O}(|D|^n \cdot \text{polylog}(M))$ time. Theorem 1.4 is significantly faster than Theorem 1.1 even when we plug-in the naive algorithm for matrix multiplication (i.e., $\omega = 3$). The proof of Theorem 1.4 is inspired by an interpretation of the f -QUERY problem as counting length-4 cycles in a graph.

1.2 Related Work

Arguably, the problem of computing the Discrete Fourier Transform (DFT) is the prime example of convolution-type problems in computer science. Cooley and Tukey [18] proposed the fast algorithm to compute DFT. Later, Beth [4] and Clausen [17] initiated the study of generalized DFTs whose goal has been to obtain a fast algorithm for DFT where the underlying group is arbitrary. After a long line of works (see [31] for the survey), the currently best algorithm for generalized DFT concerning group G runs in $\mathcal{O}(|G|^{\omega/2+\epsilon})$ operations for every $\epsilon > 0$ [32].

A similar technique to ours was introduced by Björklund et al. [9]. The paper gave a characterization of lattices that admit a fast zeta transform and a fast Möbius transform. Their paper used the notion of *covering pairs*, which is similar to cyclic partitions used in this paper but with a completely different goal.

From the lower-bounds perspective to the best of our knowledge only a naive $\Omega(|D|^n)$ lower bound is known for f -CONVOLUTION (as this is the output size). We note that known lower bounds for different convolution-type problems, such as (min, +)-convolution [19, 25], (min, max)-convolution [13], min-witness convolution [26], convolution-3SUM [14] or even skew-convolution [12] cannot be easily adapted to f -CONVOLUTION as the hardness of these problems comes primarily from the ring operations.

The Orthogonal Vector problem is related to the f -QUERY problem. In the Orthogonal Vector problem we are given two sets of n vectors $A, B \subseteq \{0, 1\}^d$ and the task is to decide if there is a pair $a \in A, b \in B$ such that $a \cdot b = 0$. In [38] it was shown that no $n^{2-\epsilon} \cdot 2^{o(d)}$

⁴ It is a special case with $D = \{0, 1\}$, $\mathbf{v} = 0^n$ and $f(x, y) = x \cdot y$

algorithm for Orthogonal Vectors is possible for any $\epsilon > 0$ assuming SETH [36]. The currently best algorithm for Orthogonal Vectors run in $n^{2-1/\mathcal{O}(\log(d)/\log(n))}$ time [1, 15], $\mathcal{O}(n \cdot 2^{cd})$ for some constant $c < 0.5$ [30], or $\mathcal{O}(|\downarrow A| + |\downarrow B|)$ [7] (where $|\downarrow F|$ is the total number of vectors whose support is a subset of the support of input vectors).

1.3 Organization

In Section 2 we provide the formal definitions of the problems alongside the general statements of our results. In Section 3 we give an algorithm for f -CONVOLUTION that uses a given cyclic partition. In Section 4 we show that for every function $f: D \times D \rightarrow D$ there exists a cyclic partition of low cost. In Section 5 we conclude the paper and discuss future work.

In Appendix A we give an algorithm for f -QUERY and prove Theorem 1.4. In Appendix C and Appendix B we include the missing proofs.

2 Preliminaries

Throughout the paper, we use Iverson bracket notation, where for the logic expression P , the value of $\llbracket P \rrbracket$ is 1 when P is true and 0 otherwise. For $n \in \mathbb{N}$ we use $[n]$ to denote $\{1, \dots, n\}$. Through the paper we denote vectors in bold, for example, $\mathbf{q} \in \mathbb{Z}^k$ denotes a k -dimensional vector of integers. We use subscripts to denote the entries of the vectors, e.g., $\mathbf{q} := (\mathbf{q}_1, \dots, \mathbf{q}_k)$.

Let L, R and T be arbitrary sets and let $f: L \times R \rightarrow T$ be an arbitrary function. We extend the definition of such an arbitrary function f to vectors as follows. For two vectors $\mathbf{u} \in L^n$ and $\mathbf{w} \in R^n$ we define

$$\mathbf{u} \oplus_f \mathbf{w} := (f(\mathbf{u}_1, \mathbf{w}_1), \dots, f(\mathbf{u}_n, \mathbf{w}_n)).$$

In this paper, we consider the f -CONVOLUTION problem with a more general domain and image. We define it formally as follows:

► **Definition 2.1** (f -Convolution). *Let L, R and T be arbitrary sets and let $f: L \times R \rightarrow T$ be an arbitrary function. The f -CONVOLUTION of two functions $g: L^n \rightarrow \mathbb{Z}$ and $h: R^n \rightarrow \mathbb{Z}$, where $n \in \mathbb{N}$, is the function $(g \otimes_f h): T^n \rightarrow \mathbb{Z}$ defined by*

$$(g \otimes_f h)(\mathbf{v}) := \sum_{\mathbf{u} \in L^n, \mathbf{w} \in R^n} \llbracket \mathbf{v} = \mathbf{u} \oplus_f \mathbf{w} \rrbracket \cdot g(\mathbf{u}) \cdot h(\mathbf{w})$$

for every $\mathbf{v} \in T^n$.

As before the operations are taken in the standard $\mathbb{Z}(+, \cdot)$ ring and M is the maximum absolute value of the integers given on the input.

Now, we formally define the input and output to the f -CONVOLUTION problem.

► **Definition 2.2** (f -CONVOLUTION PROBLEM (f -CONVOLUTION)). *Let L, R and T be arbitrary finite sets and let $f: L \times R \rightarrow T$ be an arbitrary function. The f -CONVOLUTION PROBLEM is the following.*

Input: Two functions $g: R^n \rightarrow \{-M, \dots, M\}$ and $h: L^n \rightarrow \{-M, \dots, M\}$.

Task: Compute $g \otimes_f h$.

Our main result stated in the most general form is the following.

► **Theorem 2.3.** *Let $f: L \times R \rightarrow T$ such that L, R and T are finite. There is an algorithm for the f -CONVOLUTION problem with $\tilde{O}(c^n \cdot \text{polylog}(M))$ time, where*

$$c := \begin{cases} \frac{|L|}{2} \cdot \frac{4 \cdot |R| + |T|}{3} & \text{if } |L| \text{ is even} \\ \frac{|L|-1}{2} \cdot \frac{4 \cdot |R| + |T|}{3} + |R| & \text{otherwise.} \end{cases}$$

Theorem 1.1 is a corollary of Theorem 2.3 by setting $L = R = T = D$.

The proof of Theorem 2.3 utilizes the notion of *cyclic partition*. For any $k \in \mathbb{N}$, let $\mathbb{Z}_k = \{0, 1, \dots, k-1\}$. We say a function $f: A \times B \rightarrow C$ is *k-cyclic* if, up to a relabeling of the sets A, B and C , it is an addition modulo k . Formally, $f: A \times B \rightarrow C$ is *k-cyclic* if there are $\sigma_A: A \rightarrow \mathbb{Z}_k$, $\sigma_B: B \rightarrow \mathbb{Z}_k$, and $\sigma_C: \mathbb{Z}_k \rightarrow C$ such that

$$\forall a \in A, b \in B: f(a, b) = \sigma_C(\sigma_A(a) + \sigma_B(b) \pmod k).$$

We refer to the functions σ_A, σ_B and σ_C as the *relabeling* functions of f .

The *restriction* of $f: L \times R \rightarrow T$ to $A \subseteq L$ and $B \subseteq R$ is the function $g: A \times B \rightarrow T$ defined by $g(a, b) = f(a, b)$ for all $a \in A$ and $b \in B$. We say (A, B, k) is a *cyclic minor* of $f: L \times R \rightarrow T$ if the restriction of f to A and B is a k -cyclic function.

A *cyclic partition* of $f: L \times R \rightarrow T$ is a set of minors $\mathcal{P} = \{(A_1, B_1, k_1), \dots, (A_m, B_m, k_m)\}$ such that (A_i, B_i, k_i) is a cyclic minor of f and for every $(a, b) \in L \times R$ there is a unique $1 \leq i \leq m$ such that $(a, b) \in A_i \times B_i$. The cost of the cyclic partition is $\text{cost}(\mathcal{P}) = \sum_{i=1}^m k_i$.

Theorem 2.3 follows from the following lemmas.

► **Lemma 3.1** (Algorithm for Generalized Convolution). *Let L, R and T be finite sets. Also, let $f: L \times R \rightarrow T$ be a function and let \mathcal{P} be a cyclic partition of f . Then there is an $\tilde{O}((\text{cost}(\mathcal{P})^n + |L|^n + |R|^n + |T|^n) \cdot \text{polylog}(M))$ time algorithm for f -CONVOLUTION.*

► **Lemma 4.1.** *Let $f: L \times R \rightarrow T$ where L, R and T are finite sets. Then there is a cyclic partition \mathcal{P} of f such that $\text{cost}(\mathcal{P}) \leq \frac{|L|}{2} \cdot \frac{4 \cdot |R| + |T|}{3}$ when $|L|$ is even, and $\text{cost}(\mathcal{P}) \leq |R| + \frac{|L|-1}{2} \cdot \frac{4 \cdot |R| + |T|}{3}$ when $|L|$ is odd.*

The proof of Lemma 3.1 is included in Section 3 and proof of Lemma 4.1 is included in Section 4. The proof of Lemma 3.1 uses an algorithm for CYCLIC CONVOLUTION.

► **Definition 2.4** (CYCLIC CONVOLUTION). *The CYCLIC CONVOLUTION problem is the following.*

Input: Vector $\mathbf{r} \in \mathbb{N}^k$ and functions $g, h: Z \rightarrow \{-M, \dots, M\}$ where $Z = \mathbb{Z}_{\mathbf{r}_1} \times \dots \times \mathbb{Z}_{\mathbf{r}_k}$.

Task: Compute the function $g \odot h: Z \rightarrow \mathbb{Z}$ defined by

$$(g \odot h)(\mathbf{v}) = \sum_{\mathbf{u}, \mathbf{w} \in Z} \left(\prod_{i=1}^k [\mathbf{u}_i + \mathbf{w}_i = \mathbf{v}_i \pmod{\mathbf{r}_i}] \right) \cdot g(\mathbf{u}) \cdot h(\mathbf{w}).$$

Van Rooij [33] showed that CYCLIC CONVOLUTION can be solved in $\tilde{O}\left(\left(\prod_{i=1}^k \mathbf{r}_i\right) \cdot \text{polylog}(M)\right)$ time. However, his algorithm relies on finding an appropriate large prime p . In order to circumvent the discussion on how such a prime can be found efficiently and deterministically, we can use multiple smaller primes and the Chinese Remainder Theorem. We include the details in Appendix B.

► **Theorem 2.5** (CYCLIC CONVOLUTION). *There is an $\tilde{O}\left(\left(\prod_{i=1}^k \mathbf{r}_i\right) \cdot \text{polylog}(M)\right)$ algorithm for the CYCLIC CONVOLUTION problem.*

3 Generalized Convolution

In this section we prove Lemma 3.1.

► **Lemma 3.1** (Algorithm for Generalized Convolution). *Let L , R and T be finite sets. Also, let $f: L \times R \rightarrow T$ be a function and let \mathcal{P} be a cyclic partition of f . Then there is an $\tilde{\mathcal{O}}((\text{cost}(\mathcal{P})^n + |L|^n + |R|^n + |T|^n) \cdot \text{polylog}(M))$ time algorithm for f -CONVOLUTION.*

Throughout the section we fix L , R and T , and $f: L \times R \rightarrow T$ to be as in the statement of Lemma 3.1. Additionally, fix a cyclic partition $\mathcal{P} = \{(A_1, B_1, k_1), \dots, (A_m, B_m, k_m)\}$. Furthermore, let $\sigma_{A,i}$, $\sigma_{B,i}$ and $\sigma_{C,i}$ be the relabeling functions of the cyclic minor (A_i, B_i, k_i) for every $i \in [m]$. We assume the labeling functions are also fixed throughout this section.

In order to describe our algorithm for Lemma 3.1, we first need to establish several technical definitions.

► **Definition 3.2** (Type). *The type of two vectors $\mathbf{u} \in L^n$ and $\mathbf{w} \in R^n$ is the unique vector $\bar{p} \in [m]^n$ for which $\mathbf{u}_i \in A_{\bar{p}_i}$ and $\mathbf{w}_i \in B_{\bar{p}_i}$ for all $i \in [n]$.*

Observe that the type of two vectors is well defined as \mathcal{P} is a cyclic partition. For any type $\bar{p} \in \{1, \dots, m\}^n$ we define

$$L_{\bar{p}} := A_{\bar{p}_1} \times \dots \times A_{\bar{p}_n}, \quad R_{\bar{p}} := B_{\bar{p}_1} \times \dots \times B_{\bar{p}_n}, \quad Z_{\bar{p}} := \mathbb{Z}_{k_{\bar{p}_1}} \times \dots \times \mathbb{Z}_{k_{\bar{p}_n}}$$

to be vector domains restricted to type \bar{p} . For any type \bar{p} we introduce relabeling functions on its restricted domains. The relabeling functions of \bar{p} are the functions $\sigma_{\bar{p}}^L: L_{\bar{p}} \rightarrow Z_{\bar{p}}$, $\sigma_{\bar{p}}^R: R_{\bar{p}} \rightarrow Z_{\bar{p}}$, and $\sigma_{\bar{p}}^T: Z_{\bar{p}} \rightarrow T^n$ defined as follows:

$$\begin{aligned} \sigma_{\bar{p}}^L(\mathbf{v}) &:= (\sigma_{A,\bar{p}_1}(\mathbf{v}_1), \dots, \sigma_{A,\bar{p}_n}(\mathbf{v}_n)) && \forall \mathbf{v} \in L_{\bar{p}}, \\ \sigma_{\bar{p}}^R(\mathbf{v}) &:= (\sigma_{B,\bar{p}_1}(\mathbf{v}_1), \dots, \sigma_{B,\bar{p}_n}(\mathbf{v}_n)) && \forall \mathbf{v} \in R_{\bar{p}}, \\ \sigma_{\bar{p}}^T(\mathbf{q}) &:= (\sigma_{C,\bar{p}_1}(\mathbf{q}_1), \dots, \sigma_{C,\bar{p}_n}(\mathbf{q}_n)) && \forall \mathbf{q} \in Z_{\bar{p}}. \end{aligned}$$

Our algorithm heavily depends on constructing the following projections.

► **Definition 3.3** (Projection of a Function). *The projection of a function $g: L^n \rightarrow \mathbb{Z}$ with respect to the type $\bar{p} \in [m]^n$, is the function $g_{\bar{p}}: Z_{\bar{p}} \rightarrow \mathbb{Z}$ defined as*

$$g_{\bar{p}}(\mathbf{q}) := \sum_{\mathbf{u} \in L_{\bar{p}}} \llbracket \sigma_{\bar{p}}^L(\mathbf{u}) = \mathbf{q} \rrbracket \cdot g(\mathbf{u}) \quad \text{for every } \mathbf{q} \in Z_{\bar{p}}.$$

Similarly, the projection $h_{\bar{p}}: Z_{\bar{p}} \rightarrow \mathbb{Z}$ of a function $h: R^n \rightarrow \mathbb{Z}$ with respect to the type $\bar{p} \in [m]^n$ is defined as

$$h_{\bar{p}}(\mathbf{q}) := \sum_{\mathbf{w} \in R_{\bar{p}}} \llbracket \sigma_{\bar{p}}^R(\mathbf{w}) = \mathbf{q} \rrbracket \cdot h(\mathbf{w}) \quad \text{for every } \mathbf{q} \in Z_{\bar{p}}.$$

The projections are useful due to the following connection with $g \otimes_f h$.

► **Lemma 3.4.** *Let $g: L^n \rightarrow \mathbb{Z}$ and $h: R^n \rightarrow \mathbb{Z}$, then for every $\mathbf{v} \in T^n$ it holds that:*

$$(g \otimes_f h)(\mathbf{v}) = \sum_{\bar{p} \in [m]^n} \sum_{\mathbf{q} \in Z_{\bar{p}}} \llbracket \sigma_{\bar{p}}^T(\mathbf{q}) = \mathbf{v} \rrbracket \cdot (g_{\bar{p}} \odot h_{\bar{p}})(\mathbf{q}),$$

where $g_{\bar{p}} \odot h_{\bar{p}}$ is the cyclic convolution of $g_{\bar{p}}$ and $h_{\bar{p}}$.

We give the proof of Lemma 3.4 in Section 3.1. It should be noted that the naive computation of the projection functions of g and h with respect to all types \bar{p} is significantly slower than the running time stated in Lemma 3.1. To adhere to the stated running time we use a dynamic programming procedure for the computations, as stated in the following lemma.

► **Lemma 3.5.** *There exists an algorithm which given a function $g: L^n \rightarrow \{-M, \dots, M\}$ returns the set of its projections, $\{g_{\bar{p}} \mid \bar{p} \in [m]^n\}$, in time $\tilde{O}((\text{cost}(\mathcal{P})^n + |L|^n) \cdot \text{polylog}(M))$.*

► **Remark 3.6.** Analogously, we can also construct every projection of a function $h: R^n \rightarrow \{-M, \dots, M\}$ in $\tilde{O}((\text{cost}(\mathcal{P})^n + |R|^n) \cdot \text{polylog}(M))$ time.

The proof of Lemma 3.5 is given in Appendix C.

Our algorithm for f -CONVOLUTION (see Algorithm 1 for the pseudocode) is a direct implication of Lemma 3.4 and Lemma 3.5. First, the algorithm computes the projections of g and h with respect to every type \bar{p} . Subsequently, the cyclic convolution of $g_{\bar{p}}$ and $h_{\bar{p}}$ is computed efficiently as described in Theorem 2.5. Finally, the values of $(g \otimes_f h)$ are reconstructed by the formula in Lemma 3.4.

■ **Algorithm 1** Cyclic Partition Algorithm for the f -CONVOLUTION problem.

Setting: Finite sets L, R and $T, f: L \times R \rightarrow T$ and a cyclic partition \mathcal{P} of f , of size m .

Input: $g: L^n \rightarrow \{-M, \dots, M\}, h: R^n \rightarrow \{-M, \dots, M\}$

- 1 Construct the projections of g and h w.r.t \bar{p} , for all $\bar{p} \in [m]^n$ ▷ Lemma 3.5
- 2 For every $\bar{p} \in [m]^n$ compute $c_{\bar{p}} = g_{\bar{p}} \odot h_{\bar{p}}$ ▷ Cyclic convolutions (Definition 2.4)
- 3 Define $r: T^n \rightarrow \mathbb{Z}$ by

$$r(\mathbf{v}) = \sum_{\bar{p} \in [m]^n} \sum_{\mathbf{q} \in Z_{\bar{p}} \text{ s.t. } \sigma_{\bar{p}}^T(\mathbf{q}) = \mathbf{v}} c_{\bar{p}}(\mathbf{q}) \quad \text{for all } \mathbf{v} \in T^n.$$

4 **return** r

Proof of Lemma 3.1. Observe that Algorithm 1 returns $r: T^n \rightarrow \mathbb{Z}$ such that for every $\mathbf{v} \in T^n$ it holds that

$$r(\mathbf{v}) = \sum_{\bar{p} \in [m]^n} \sum_{\substack{\mathbf{q} \in Z_{\bar{p}} \\ \text{s.t. } \sigma_{\bar{p}}^T(\mathbf{q}) = \mathbf{v}}} c_{\bar{p}}(\mathbf{q}) = \sum_{\bar{p} \in [m]^n} \sum_{\mathbf{q} \in Z_{\bar{p}}} [\sigma_{\bar{p}}^T(\mathbf{q}) = \mathbf{v}] \cdot (g_{\bar{p}} \odot h_{\bar{p}})(\mathbf{q}) = (g \otimes_f h)(\mathbf{v}),$$

where the last equality is by Lemma 3.4. Thus, the algorithm returns $(g \otimes_f h)$ as required. It therefore remains to bound the running time of the algorithm.

By Lemma 3.5, Line 1 of Algorithm 1 runs in time $\tilde{O}((\text{cost}(\mathcal{P})^n + |L|^n + |R|^n) \cdot \text{polylog}(M))$. By Theorem 2.5, for any type $\bar{p} \in [m]^n$ the computation of $g_{\bar{p}} \odot h_{\bar{p}}$ in Line 2 runs in time $\tilde{O}((\prod_{i=1}^n k_{\bar{p}_i}) \cdot \text{polylog}(M))$. Thus the overall running time of Line 2 is $\tilde{O}((\sum_{\bar{p} \in [m]^n} \prod_{i=1}^n k_{\bar{p}_i}) \cdot \text{polylog}(M))$.

Finally, observe that the construction of r in Line 3 can be implemented by initializing r to be zeros and iteratively adding the value of $c_{\bar{p}}(\mathbf{q})$ to $r(\sigma_{\bar{p}}^T(\mathbf{q}))$ for every $\bar{p} \in [m]^n$ and $\mathbf{q} \in Z_{\bar{p}}$. The required running time is thus $\tilde{O}(|T|^n \cdot \text{polylog}(M))$ for the initialization and $\tilde{O}((\sum_{\bar{p} \in [m]^n} |Z_{\bar{p}}|) \cdot \text{polylog}(M)) = \tilde{O}((\sum_{\bar{p} \in [m]^n} \prod_{i=1}^n k_{\bar{p}_i}) \cdot \text{polylog}(M))$ for the addition operations. Thus, the overall running time of Line 3 is

$$\tilde{O} \left(\left(|T|^n + \sum_{\bar{p} \in [m]^n} \prod_{i=1}^n k_{\bar{p}_i} \right) \cdot \text{polylog}(M) \right).$$

12:10 Computing Generalized Convolutions Faster Than Brute Force

Combining the above, with $\sum_{\bar{p} \in [m]^n} \prod_{i=1}^n k_{\bar{p}_i} = (\sum_{i=1}^m k_i)^n = (\text{cost}(\mathcal{P}))^n$ means that the running time of Algorithm 1 is

$$\tilde{O}((|T|^n + |R|^n + |L|^n + \text{cost}(\mathcal{P})^n) \cdot \text{polylog}(M))$$

This concludes the proof of Lemma 3.1. \blacktriangleleft

3.1 Properties of Projections

In this section we provide the proof for Lemma 3.4. The proof of Lemma 3.4 uses the following definitions of coordinate-wise addition with respect to a type \bar{p} .

► **Definition 3.7** (Coordinate-wise Addition Modulo for Type). *For any $\bar{p} \in [m]^n$ we define a coordinate-wise addition modulo as*

$$\mathbf{q} +_{\bar{p}} \mathbf{r} := ((\mathbf{q}_1 + \mathbf{r}_1 \pmod{k_{\bar{p}_1}}, \dots, (\mathbf{q}_n + \mathbf{r}_n \pmod{k_{\bar{p}_n}})) \quad \text{for every } \mathbf{q}, \mathbf{r} \in Z_{\bar{p}}.$$

Proof of Lemma 3.4. By Definition 2.1 it holds that:

$$(g \oplus_f h)(\mathbf{v}) = \sum_{\mathbf{u} \in L^n, \mathbf{w} \in R^n} \llbracket \mathbf{v} = \mathbf{u} \oplus_f \mathbf{w} \rrbracket \cdot g(\mathbf{u}) \cdot h(\mathbf{w}). \quad (3.1)$$

Recall that the type of every two vectors $(\mathbf{u}, \mathbf{w}) \in L^n \times R^n$ is unique and $[m]^n$ contains all possible types and hence, we can rewrite (3.1) as

$$(g \oplus_f h)(\mathbf{v}) = \sum_{\bar{p} \in [m]^n} \sum_{\mathbf{u} \in L_{\bar{p}}, \mathbf{w} \in R_{\bar{p}}} g(\mathbf{u}) \cdot h(\mathbf{w}) \cdot \llbracket \mathbf{v} = \mathbf{u} \oplus_f \mathbf{w} \rrbracket \quad (3.2)$$

By the properties of the relabeling functions, we get

$$\begin{aligned} &= \sum_{\bar{p} \in [m]^n} \sum_{\mathbf{u} \in L_{\bar{p}}, \mathbf{w} \in R_{\bar{p}}} g(\mathbf{u}) \cdot h(\mathbf{w}) \cdot \llbracket \mathbf{v} = \sigma_{\bar{p}}^T(\sigma_{\bar{p}}^L(\mathbf{u}) +_{\bar{p}} \sigma_{\bar{p}}^R(\mathbf{w})) \rrbracket \\ &= \sum_{\bar{p} \in [m]^n} \sum_{\mathbf{q} \in Z_{\bar{p}}} \sum_{\mathbf{u} \in L_{\bar{p}}, \mathbf{w} \in R_{\bar{p}}} g(\mathbf{u}) \cdot h(\mathbf{w}) \cdot \llbracket \mathbf{v} = \sigma_{\bar{p}}^T(\mathbf{q}) \rrbracket \cdot \llbracket \mathbf{q} = \sigma_{\bar{p}}^L(\mathbf{u}) +_{\bar{p}} \sigma_{\bar{p}}^R(\mathbf{w}) \rrbracket \\ &= \sum_{\bar{p} \in [m]^n} \sum_{\substack{\mathbf{q} \in Z_{\bar{p}} \\ \text{s.t. } \sigma_{\bar{p}}^T(\mathbf{q}) = \mathbf{v}}} \sum_{\mathbf{u} \in L_{\bar{p}}, \mathbf{w} \in R_{\bar{p}}} g(\mathbf{u}) \cdot h(\mathbf{w}) \cdot \llbracket \mathbf{q} = \sigma_{\bar{p}}^L(\mathbf{u}) +_{\bar{p}} \sigma_{\bar{p}}^R(\mathbf{w}) \rrbracket. \end{aligned}$$

Observe that we can *partition* $L_{\bar{p}}$ (respectively $R_{\bar{p}}$) by considering the inverse images of $\mathbf{r} \in Z_{\bar{p}}$ under $\sigma_{\bar{p}}^L$ (respectively $\sigma_{\bar{p}}^R$), i.e. $L_{\bar{p}} = \bigsqcup_{\mathbf{r} \in Z_{\bar{p}}} \{\mathbf{u} \in L_{\bar{p}} \mid \sigma_{\bar{p}}^L(\mathbf{u}) = \mathbf{r}\}$. Hence, for every $\bar{p} \in [m]^n$ and $\mathbf{q} \in Z_{\bar{p}}$ it holds that

$$\begin{aligned} & \sum_{\mathbf{u} \in L_{\bar{p}}, \mathbf{w} \in R_{\bar{p}}} g(\mathbf{u}) \cdot h(\mathbf{w}) \cdot \llbracket \mathbf{q} = \sigma_{\bar{p}}^L(\mathbf{u}) +_{\bar{p}} \sigma_{\bar{p}}^R(\mathbf{w}) \rrbracket \\ &= \sum_{\mathbf{r}, \mathbf{s} \in Z_{\bar{p}}} \sum_{\mathbf{u} \in L_{\bar{p}}, \mathbf{w} \in R_{\bar{p}}} g(\mathbf{u}) \cdot h(\mathbf{w}) \cdot \llbracket \mathbf{q} = \mathbf{r} +_{\bar{p}} \mathbf{s} \rrbracket \cdot \llbracket \mathbf{r} = \sigma_{\bar{p}}^L(\mathbf{u}) \rrbracket \cdot \llbracket \mathbf{s} = \sigma_{\bar{p}}^R(\mathbf{w}) \rrbracket \\ &= \sum_{\mathbf{r}, \mathbf{s} \in Z_{\bar{p}}} \llbracket \mathbf{q} = \mathbf{r} +_{\bar{p}} \mathbf{s} \rrbracket \left(\sum_{\mathbf{u} \in L_{\bar{p}}} \llbracket \mathbf{r} = \sigma_{\bar{p}}^L(\mathbf{u}) \rrbracket \cdot g(\mathbf{u}) \right) \cdot \left(\sum_{\mathbf{w} \in R_{\bar{p}}} \llbracket \mathbf{s} = \sigma_{\bar{p}}^R(\mathbf{w}) \rrbracket \cdot h(\mathbf{w}) \right) \\ &= \sum_{\mathbf{r}, \mathbf{s} \in Z_{\bar{p}}} \llbracket \mathbf{q} = \mathbf{r} +_{\bar{p}} \mathbf{s} \rrbracket \cdot g_{\bar{p}}(\mathbf{r}) \cdot h_{\bar{p}}(\mathbf{s}) \\ &= (g_{\bar{p}} \odot h_{\bar{p}})(\mathbf{q}). \end{aligned} \quad (3.3)$$

By plugging (3.3) into (3.2) we get

$$(g \circledast_f h)(\mathbf{v}) = \sum_{\bar{p} \in [m]^n} \sum_{\substack{\mathbf{q} \in Z_{\bar{p}} \\ \text{s.t. } \sigma_{\bar{p}}^T(\mathbf{q}) = \mathbf{v}}} (g_{\bar{p}} \odot h_{\bar{p}})(\mathbf{q}) = \sum_{\bar{p} \in [m]^n} \sum_{\mathbf{q} \in Z_{\bar{p}}} [\sigma_{\bar{p}}^T(\mathbf{q}) = \mathbf{v}] \cdot (g_{\bar{p}} \odot h_{\bar{p}})(\mathbf{q}),$$

as required. ◀

4 Existence of Low-Cost Cyclic Partition

In this section we prove Lemma 4.1.

► **Lemma 4.1.** *Let $f: L \times R \rightarrow T$ where L , R and T are finite sets. Then there is a cyclic partition \mathcal{P} of f such that $\text{cost}(\mathcal{P}) \leq \frac{|L|}{2} \cdot \frac{4 \cdot |R| + |T|}{3}$ when $|L|$ is even, and $\text{cost}(\mathcal{P}) \leq |R| + \frac{|L|-1}{2} \cdot \frac{4 \cdot |R| + |T|}{3}$ when $|L|$ is odd.*

We first consider the special case when $|L| = 2$. Later we reduce the general case to this scenario and use the result as a black-box.

As a warm-up we construct a cyclic partition of cost at most $\frac{7}{8}|D|^2$ assuming that $L = R = T = D$ and that $|D|$ is even. For this, we first partition D into pairs $d_1^{(i)}, d_2^{(i)}$ where $i \in [|D|/2]$ and show for each such pair that f restricted to $\{d_1^{(i)}, d_2^{(i)}\}$ and D has a cyclic partition of cost at most $\frac{7}{4}|D|$. The union of these cyclic partitions forms a cyclic partition of f with cost at most $\frac{|D|}{2} \cdot \frac{7}{4}|D|$.

To construct the cyclic partition for a fixed $i \in [|D|/2]$, we find a maximal number r of pairwise disjoint pairs $e_1^{(j)}, e_2^{(j)} \in D$ such that $|\{f(d_a^{(i)}, e_b^{(j)}) \mid a, b = 1, 2\}| \leq 3$ for each $j \in [r]$, i.e. for each j at least one of the four values repeats. With this assumption, f restricted to $\{d_1^{(i)}, d_2^{(i)}\}$ and $\{e_1^{(j)}, e_2^{(j)}\}$ is either a cyclic minor of cost at most 3 or can be decomposed into 3 trivial cyclic minors of the total cost at most 3. We claim that $r \geq |D|/4$. Indeed, assume that there are fewer than $|D|/4$ such pairs, i.e. $r < |D|/4$. Let \bar{D} denote the $|D| - 2 \cdot r > |D|/2$ remaining values in D . As the set $\{f(d_a^{(i)}, d) \mid d \in \bar{D}, a = 1, 2\}$ can only contain at most $|D|$ values, we can find another pair $e_1^{(r+1)}, e_2^{(r+1)}$ with the above constraints. Note that f restricted to $\{d_1^{(i)}, d_2^{(i)}\}$ and \bar{D} can be decomposed into at most $2|\bar{D}|$ trivial minors. Hence, the cyclic partition for f restricted to $\{d_1^{(i)}, d_2^{(i)}\}$ and D has cost at most

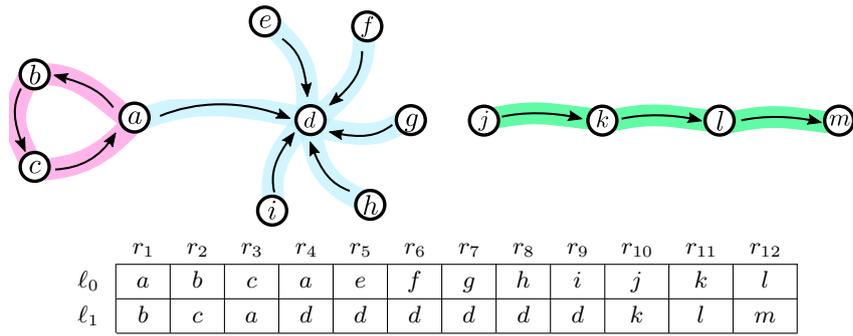
$$3r + 2 \cdot |\bar{D}| \leq 3 \cdot \frac{|D|}{4} + 2 \cdot \frac{|D|}{2} \leq \frac{7}{4}|D|.$$

4.1 Special Case

In this section, we prove the following lemma that is a special case of Lemma 4.1.

► **Lemma 4.2.** *If $f: L \times R \rightarrow T$ with $|L| = 2$, then there is a cyclic partition \mathcal{P} of f such that $\text{cost}(\mathcal{P}) \leq (4|R| + |T|)/3$.*

To construct the cyclic partition we proceed as follows. First we define, for a function f , the representation graph G_f . Next we show that if this graph has a special structure, which we later call *nice*, then we can easily find a cyclic partition for the function f . Afterwards we decompose (the edges of) an arbitrary representation graph G_f into nice structures and then combine the cyclic partitions coming from these parts to a cyclic partition for the original function f .



■ **Figure 4.1** Example of the construction of a representation graph from the function f to obtain a cyclic partition. We put an edge between vertices u and v if there is an r_i with $u = f(\ell_0, r_i)$ and $v = f(\ell_1, r_i)$. We highlight the partition of the graph into cycle (red), in-star (blue) and path (green). The cost of this cyclic partition is $3 + 7 + 4 = 14$.

► **Definition 4.3.** Let $f: L \times R \rightarrow T$ be a such that $|L| = 2$ with $L = \{\ell_0, \ell_1\}$.

We define a function $\lambda_f: R \rightarrow T \times T$ with $\lambda_f: r \mapsto (f(\ell_0, r), f(\ell_1, r))$ as the edge mapping of f .

We define a directed graph G_f (which might have loops) with vertex set $V(G_f) := T$ and edge set $E(G_f) := \{\lambda_f(r) \mid r \in R\}$. We say that G_f is the representation graph of f .

We say that a representation graph G_f is nice if it is a cycle, a path (potentially with only one edge), an in-star, or an out-star.⁵

Let $E' \subseteq E(G_f)$ be a subset of edges inducing the subgraph G' of G_f . With $T' := V(G')$ and $R' := \{r \in R \mid \lambda_f(r) \in E'\}$, we define $f': L \times R' \rightarrow T'$ as the restriction of f such that the representation graph of f' is G' . Formally, $f'(\ell, r) = f(\ell, r)$ for all $\ell \in L$ and $r \in R'$. We say that f' is the function represented by G' or E' , respectively.

A decomposition of a directed graph G is a set C of edge-disjoint subgraphs of G , such that each edge belongs to exactly one set in C . For ease of notation, we identify the set of edges E' with the induced graph G' . For a graph G , the *line graph* $L(G)$ is a graph where the set of edges $E(G)$ is the vertex set of $L(G)$ and $e_1, e_2 \in V(L(G))$ are adjacent in $L(G)$ if edges e_1, e_2 share an endpoint in graph G .

The following observation follows directly from the previous definition.

► **Observation 4.4.** Let G_1, \dots, G_p be decomposition of the graph G_f into p subgraphs, let f_i be the function represented by G_i , and let \mathcal{P}_i be a cyclic partition of f_i . Then $\mathcal{P} = \bigcup_{i \in [p]} \mathcal{P}_i$ is a cyclic partition of f with cost $\text{cost}(\mathcal{P}) = \sum_{i \in [p]} \text{cost}(\mathcal{P}_i)$.

Cyclic Partitions Given Nice Representation Graphs. As a next step, we show that functions admit cyclic partitions if the representation graph is nice. Afterwards we show how to decompose (the representation) graphs into nice (representation) graphs. Finally, we combine these results to obtain a cyclic partition for the original function f . See Figure 4.1 for an example.

► **Lemma 4.5.** Let $f: L \times R \rightarrow T$ be a function such that G_f is nice. Then f has a cyclic partition of cost at most $|T|$.

⁵ A star graph where either all edges are directed to the central vertex or all edges are directed away from it, respectively.

Proof. By definition, a nice graph is either a path, a cycle or an in-star or out-star. We handle each case separately in the following. Let $L = \{\ell_0, \ell_1\}$.

G_f is a cycle. We first define the relabeling functions of f to show that f is $|T|$ -cyclic.

For the elements in L , let $\sigma_L: L \rightarrow \mathbb{Z}_2$ with $\sigma_L(\ell_i) = i$. To define σ_R and σ_T , fix an arbitrary $t_0 \in T$. Let $t_1, \dots, t_{|T|}$ be the elements in T with $t_{|T|} = t_0$ such that, for all $i \in \mathbb{Z}_{|T|}$, there is some $r_i \in R$ with $\lambda_f(r_i) = (t_i, t_{i+1})$.⁶ Note that these r_i exist since G_f is a cycle. Using this notation, we define $\sigma_T: \mathbb{Z}_{|T|} \rightarrow T$ with $\sigma_T(i) = t_i$, for all $i \in \mathbb{Z}_{|T|}$. For the elements in R we define $\sigma_R: R \rightarrow \mathbb{Z}_{|R|}$ with $\sigma_R(r) = i$ whenever $\lambda_f(r) = (t_i, t_{i+1})$ for some i .

It is easy to check that f can be seen as addition modulo $|T|$. Indeed, let $j \in \{0, 1\}$ and $r \in R$ with $\lambda_f(r) = (t_i, t_{i+1})$. Then we get

$$\sigma_T(\sigma_L(\ell_j) + \sigma_R(r) \pmod{|T|}) = \sigma_T(j + i \pmod{|T|}) = t_{j+i \pmod{|T|}} = f(\ell_j, r_i) = f(\ell_j, r).$$

Thus, f is $|T|$ -cyclic and $\{(L, R, |T|)\}$ is a cyclic partition of f .

G_f is a path. Similarly to the previous case, f can be represented as addition modulo $|T|$.

As the proof is essentially identical to the cyclic case, we omit the details here.

G_f is a star. We only consider the out-star as the in-star follows symmetrically by swapping the roles of ℓ_0 and ℓ_1 . We define the following cyclic partition \mathcal{P} as

$$\mathcal{P} := \{(\{\ell_0\}, R, 1)\} \cup \{(\{\ell_1\}, \{r\}, 1) \mid r \in R\}.$$

Note that every $(\ell, r) \in L \times R$ appears in exactly one minor of \mathcal{P} . Hence, \mathcal{P} is indeed a cyclic partition. Next, we observe that each minor contains exactly one element of T . Thus, no addition is needed and hence f is 1-cyclic for each minor of \mathcal{P} . Thus the cost of each minor of \mathcal{P} is 1.

By the structure of G_f , the cost of cyclic partition \mathcal{P} is $|R| + 1 = |T|$. ◀

Decomposition into Nice Graphs. We first decompose the graph into cycles and acyclic components. The later parts are then decomposed further using the next two results.

▷ **Claim 4.6.** Every directed graph G can be decomposed into cycles and acyclic graphs.

Proof. We remove an arbitrary cycle from the graph and add it as a new component to the decomposition. We repeat this procedure until the graph is acyclic. ◁

Next, we decompose the acyclic graph further. First, we decompose it into pairs of edges that share at least one endpoint.

▷ **Claim 4.7.** Every directed graph $G = (V, E)$ whose undirected version is connected can be decomposed into $\lfloor |E|/2 \rfloor$ pairs of edges which share (at least) one endpoint and, if and only if $|E|$ is odd, one additional single edge.

Proof. If the graph has an odd number of edges, then we find one edge e , such that the removal of e does not disconnect the graph (except for maybe one isolated vertex). Next, we include this edge into the decomposition and apply the procedure for the case when the number of edges is even.

Chartrand et al. [16, Theorem 1] showed that if a graph G has an even number of edges, then there is a perfect matching in the line graph $L(G)$ of G . This perfect matching directly gives us a partition of the edges of G into $\lfloor |E|/2 \rfloor$ pairs which share at least one endpoint. ◁

⁶ Note that there might be multiple $r \in R$ with $\lambda_f(r) = (t_i, t_{i+1})$.

12:14 Computing Generalized Convolutions Faster Than Brute Force

Next, we present a different way to decompose the graph into nice structures.

▷ **Claim 4.8.** Every directed acyclic graph $G = (V, E)$ can be decomposed into at most $|V| - 1$ out-stars.

Proof. The sets of out-going edges from each vertex form a partition of all edges of the graph G . Moreover, each such non-empty set of edges describes an out-star. As in every directed acyclic graph, there is at least one sink vertex, there are at most $|V| - 1$ such out-stars. ◀

Combining the Results. Finally, we are ready to combine the above results and prove Lemma 4.2.

Proof of Lemma 4.2. We first use Claim 4.6 to decompose G_f into c cycles C_1, \dots, C_c and d connected, acyclic graphs G_1, \dots, G_d .

For each C_i , Lemma 4.5 gives us a cyclic partition \mathcal{P}'_i for the associated function with cost at most $|E(C_i)| = |V(C_i)|$. For the remaining components, we use the following claim.

▷ **Claim 4.9.** For each G_i , there is a cyclic partition \mathcal{P}_i for the function represented by G_i with the cost at most

$$\text{cost}(\mathcal{P}_i) \leq \frac{4|E(G_i)| + |V(G_i)|}{3}. \quad (4.1)$$

Proof. Fix some $i \in [d]$ in the following. We show the claim by considering two cases. For ease of notation, let $E_i = E(G_i)$ and $V_i = V(G_i)$.

In the case when $2|V_i| \geq |E_i| + 3$, we decompose the graph G_i via Claim 4.7. This decomposes G_i into $\lfloor |E_i|/2 \rfloor$ pairs of edges that share an endpoint (plus an extra edge when $|E_i|$ is odd). Observe that a pair of edges that share an endpoint is either a directed path, an in-star, or an out-star. Hence, by Lemma 4.5 each pair contributes a cost of 3 to the cyclic partition. Therefore, by Observation 4.4, the function represented by G_i has a cyclic partition with a cost at most $3|E_i|/2$ if $|E_i|$ is even and with cost at most $3(|E_i| - 1)/2 + 2$ if $|E_i|$ is odd. As the latter bound is the larger one, it can be easily checked that the claimed bound for the cyclic partition follows.

It remains to analyze case $2|V_i| < |E_i| + 3$. Here, we use Claim 4.8 to decompose the graph G_i into out-stars. By Observation 4.4 and as there is at least one sink vertex, there is a cyclic partition of the function represented by G_i with cost at most $|E_i| + |V_i| - 1$. By the assumption that $2|V_i| \leq |E_i| + 3$, the cost of the cyclic partition is bounded by $(4|E_i| + |V_i|)/3$ which settles (4.1). ◀

With the notation from the claim and by Observation 4.4, we define the cyclic partition \mathcal{P} for f as

$$\mathcal{P} := \bigcup_{i \in [c]} \mathcal{P}'_i \cup \bigcup_{i \in [d]} \mathcal{P}_i.$$

Because the G_i s are connected components of G_f after the removal of $C_1 \dots C_c$, they are vertex-disjoint and it holds that $\sum_{i \in [d]} |V(G_i)| \leq |V(G_f)|$. Moreover, by Lemma 4.5 the cost of each cycle C_i is $|E(C_i)|$. Hence, we get

$$\text{cost}(\mathcal{P}) \leq \sum_{i \in [c]} |E(C_i)| + \sum_{i \in [d]} \frac{4|E(G_i)| + |V(G_i)|}{3} \leq \frac{4|E(G_f)| + |V(G_f)|}{3}.$$

Because $|E(G_f)| \leq |R|$ and $|V(G_f)| = |T|$ the cost of this cyclic partition is bounded which finishes the proof. ◀

4.2 General Case

Now we have everything ready to prove the main result of this section.

Proof of Lemma 4.1. We first handle the case when $|L|$ is even. We partition L into $\lambda = |L|/2$ sets L_1, \dots, L_λ consisting of exactly two elements. We use Lemma 4.2 to find a cyclic partition \mathcal{P}_i for each $f_i: L_i \times R \rightarrow T$. By definition of the cyclic partition, $\mathcal{P} = \bigcup_{i \in [\lambda]} \mathcal{P}_i$ is a cyclic partition for f , hence it remains to analyze the cost of \mathcal{P} .

Observe that for each G_i we have that $|V_i| \leq |T|$ and $|E_i| \leq |R|$. By the definition of the cost of the cyclic partition, we immediately get that

$$\text{cost}(\mathcal{P}) \leq \sum_{i=1}^{\lambda} \text{cost}(\mathcal{P}_i) \leq \lambda \cdot \frac{4 \cdot |R| + |T|}{3}.$$

If $|L|$ is odd, then we remove one element ℓ from L and let $L_0 = \{\ell\}$. There is a trivial cyclic partition \mathcal{P}_0 for $f_0: L_0 \times R \rightarrow T$ of cost at most $|R|$. Then we use the above procedure to find a cyclic partition \mathcal{P}' for the restriction of f to $L \setminus \{\ell\}$ and R . Hence, setting $\mathcal{P} = \mathcal{P}_0 \cup \mathcal{P}'$ gives a cyclic partition for f with cost

$$\text{cost}(\mathcal{P}) \leq \text{cost}(\mathcal{P}_0) + \text{cost}(\mathcal{P}') \leq |R| + \left\lfloor \frac{|L|}{2} \right\rfloor \left(\frac{4 \cdot |R| + |T|}{3} \right). \quad \blacktriangleleft$$

► **Remark 4.10.** If $|L|$ and $|R|$ are both even, one can easily achieve the following cost

$$\min \left(\frac{L}{2} \cdot \frac{4 \cdot |R| + |T|}{3}, \frac{R}{2} \cdot \frac{4 \cdot |L| + |T|}{3} \right)$$

by swapping the role of L and R and considering the function $f': R \times L \rightarrow T$ with $f'(r, \ell) = f(\ell, r)$ for all $\ell \in L$ and $r \in R$.

5 Conclusion and Future Work

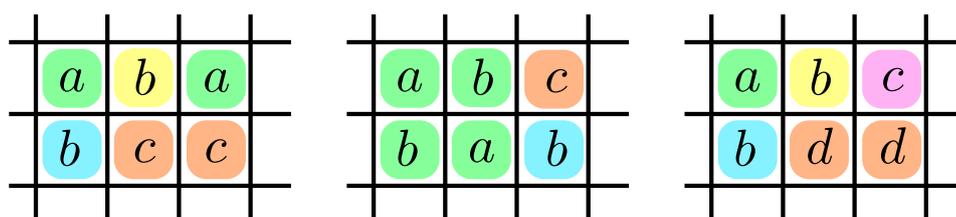
In this paper, we studied the f -CONVOLUTION problem and demonstrated that the naive brute-force algorithm can be improved for every $f: D \times D \rightarrow D$. We achieve that by introducing a *cyclic partition* of a function and showing that there always exists a cyclic partition of bounded cost. We give an $\tilde{O}((c|D|^2)^n \cdot \text{polylog}(M))$ time algorithm that computes f -CONVOLUTION for $c := 5/6$ when $|D|$ is even.

The cyclic partition is a very general tool and potentially it can be used to achieve greater improvements for certain functions f . For example, in multiple applications (e.g., [20, 34, 24, 29]) the function f has a cyclic partition with a single cyclic minor. Nevertheless, in our proof we only use cyclic minors where one dimension is at most 2. We suspect that better results can be obtained by considering larger minors.

We leave several open problems. Our algorithm offers an exponential (in n) improvement over a naive algorithm for domains D of constant size. Can we hope for an $\tilde{O}(|D|^{(2-\epsilon)n} \cdot \text{polylog}(M))$ time algorithm for f -CONVOLUTION for some $\epsilon > 0$? We are not aware of any lower bounds, so in principle even an $\tilde{O}(|D|^n \cdot \text{polylog}(M))$ time algorithm is plausible.

Ideally, we would expect that the f -CONVOLUTION problem can be solved in $\tilde{O}((|L|^n + |R|^n + |T|^n) \cdot \text{polylog}(M))$ for any function $f: L \times R \rightarrow T$. In Figure 5.1 we include three examples of functions that are especially difficult for our methods.

Finally, we gave an $\tilde{O}(|D|^{\omega \cdot n/2} \cdot \text{polylog}(M))$ time algorithm for f -QUERY problem. For $\omega = 2$ this algorithm runs in almost linear-time, however for the current bound $\omega < 2.373$ our algorithm runs in time $\tilde{O}(|D|^{1.19n} \cdot \text{polylog}(M))$. Can f -QUERY be solved in $\tilde{O}(|D|^n \cdot \text{polylog}(M))$ time without assuming $\omega = 2$?



■ **Figure 5.1** Here are three concrete examples of functions f for which we expect that the running times for f -CONVOLUTION should be $\tilde{O}(3^n \cdot \text{polylog}(M))$, $\tilde{O}(3^n \cdot \text{polylog}(M))$ and $\tilde{O}(4^n \cdot \text{polylog}(M))$. However, the best cyclic partitions for these functions have costs 4, 4 and 5 (the partitions are highlighted appropriately). This implies that the best running time, which may be attained using our techniques are $\tilde{O}(4^n \cdot \text{polylog}(M))$, $\tilde{O}(4^n \cdot \text{polylog}(M))$ and $\tilde{O}(5^n \cdot \text{polylog}(M))$.

References

- 1 Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More Applications of the Polynomial Method to Algorithm Design. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 218–230. SIAM, 2015. doi:10.1137/1.9781611973730.17.
- 2 Josh Alman and Virginia Vassilevska Williams. A Refined Laser Method and Faster Matrix Multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.
- 3 Michael A. Bennett, Greg Martin, Kevin O’Byrant, and Andrew Rechnitzer. Explicit bounds for primes in arithmetic progressions. *Illinois J. Math.*, 62(1-4):427–532, 2018. doi:10.1215/ijm/1552442669.
- 4 Thomas Beth. *Verfahren der schnellen Fourier-Transformation: die allgemeine diskrete Fourier-Transformation—ihre algebraische Beschreibung, Komplexität und Implementierung*, volume 61. Teubner, 1984.
- 5 Andreas Björklund and Thore Husfeldt. The Parity of Directed Hamiltonian Cycles. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 727–735. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.83.
- 6 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier Meets Möbius: Fast Subset Convolution. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74. ACM, 2007. doi:10.1145/1250790.1250801.
- 7 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Counting Paths and Packings in Halves. In Amos Fiat and Peter Sanders, editors, *Algorithms – ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 578–586. Springer, 2009. doi:10.1007/978-3-642-04128-0_52.
- 8 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Covering and packing in linear space. *Inf. Process. Lett.*, 111(21-22):1033–1036, 2011. doi:10.1016/j.ipl.2011.08.002.
- 9 Andreas Björklund, Thore Husfeldt, Petteri Kaski, Mikko Koivisto, Jesper Nederlof, and Pekka Parviainen. Fast Zeta Transforms for Lattices with Few Irreducibles. *ACM Trans. Algorithms*, 12(1):4:1–4:19, 2016. doi:10.1145/2629429.
- 10 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set Partitioning via Inclusion-Exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009. doi:10.1137/070683933.

- 11 Cornelius Brand. Discriminantal subset convolution: Refining exterior-algebraic methods for parameterized algorithms. *Journal of Computer and System Sciences*, 129:62–71, 2022. doi:10.1016/j.jcss.2022.05.004.
- 12 Karl Bringmann, Nick Fischer, Danny Hermelin, Dvir Shabtay, and Philip Wellnitz. Faster Minimization of Tardy Processing Time on a Single Machine. *Algorithmica*, 84(5):1341–1356, 2022. doi:10.1007/s00453-022-00928-w.
- 13 Karl Bringmann, Marvin Künnemann, and Karol Węgrzycki. Approximating APSP without scaling: equivalence of approximate min-plus and exact min-max. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 943–954, 2019.
- 14 Timothy M. Chan and Qizheng He. Reducing 3SUM to Convolution-3SUM. In Martin Farach-Colton and Inge Li Gørtz, editors, *3rd Symposium on Simplicity in Algorithms, SOSA 2020, Salt Lake City, UT, USA, January 6-7, 2020*, pages 1–7. SIAM, 2020. doi:10.1137/1.9781611976014.1.
- 15 Timothy M. Chan and R. Ryan Williams. Deterministic APSP, Orthogonal Vectors, and More: Quickly Derandomizing Razborov-Smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021. doi:10.1145/3402926.
- 16 Gary Chartrand, Albert D Polimeni, and M James Stewart. The existence of 1-factors in line graphs, squares, and total graphs. In *Indagationes Mathematicae (Proceedings)*, volume 76, pages 228–232. Elsevier, 1973.
- 17 Michael Clausen. Fast generalized Fourier transforms. *Theoretical Computer Science*, 67(1):55–63, 1989.
- 18 James W Cooley and John W Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, 19(90):297–301, 1965.
- 19 Marek Cygan, Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. On Problems Equivalent to $(\min, +)$ -Convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019. doi:10.1145/3293465.
- 20 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving Connectivity Problems Parameterized by Treewidth in Single Exponential Time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.
- 21 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.
- 22 Barış Can Esmer, Ariel Kulik, Dániel Marx, Philipp Schepper, and Karol Węgrzycki. Computing Generalized Convolutions Faster Than Brute Force, 2022. doi:10.48550/ARXIV.2209.01623.
- 23 Philip Hall. A contribution to the theory of groups of prime-power order. *Proceedings of the London Mathematical Society*, 2(1):29–95, 1934.
- 24 Falko Hegerfeld and Stefan Kratsch. Solving Connectivity Problems Parameterized by Treedepth in Single-Exponential Time and Polynomial Space. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPICs*, pages 29:1–29:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.STACS.2020.29.
- 25 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the Fine-Grained Complexity of One-Dimensional Dynamic Programming. 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 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.21.
- 26 Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic Triangles, Intermediate Matrix Products, and Convolutions. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPICs*, pages 53:1–53:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ITCS.2020.53.

- 27 Jesper Nederlof. personal communication, 2022.
- 28 Jesper Nederlof, Jakub Pawlewicz, Céline M. F. Swennenhuis, and Karol Węgrzycki. A Faster Exponential Time Algorithm for Bin Packing With a Constant Number of Bins via Additive Combinatorics. In Daniel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 1682–1701. SIAM, 2021. doi:10.1137/1.9781611976465.102.
- 29 Jesper Nederlof, Michał Pilipczuk, Céline M. F. Swennenhuis, and Karol Węgrzycki. Hamiltonian Cycle Parameterized by Treedepth in Single Exponential Time and Polynomial Space. In Isolde Adler and Haiko Müller, editors, *Graph-Theoretic Concepts in Computer Science – 46th International Workshop, WG 2020, Leeds, UK, June 24–26, 2020, Revised Selected Papers*, volume 12301 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2020. doi:10.1007/978-3-030-60440-0_3.
- 30 Jesper Nederlof and Karol Węgrzycki. Improving Schroeppeel and Shamir’s Algorithm for Subset Sum via Orthogonal Vectors. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC ’21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21–25, 2021*, pages 1670–1683. ACM, 2021. doi:10.1145/3406325.3451024.
- 31 Daniel N Rockmore. Recent progress and applications in group FFTs. In *Computational noncommutative algebra and applications*, pages 227–254. Springer, 2004.
- 32 Chris Umans. Fast Generalized DFTs for all Finite Groups. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9–12, 2019*, pages 793–805. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00052.
- 33 Johan M. M. van Rooij. Fast Algorithms for Join Operations on Tree Decompositions. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treedepth, Kernels, and Algorithms – Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 262–297. Springer, 2020. doi:10.1007/978-3-030-42071-0_18.
- 34 Johan M. M. van Rooij. A Generic Convolution Algorithm for Join Operations on Tree Decompositions. In Rahul Santhanam and Daniil Musatov, editors, *Computer Science – Theory and Applications – 16th International Computer Science Symposium in Russia, CSR 2021, Sochi, Russia, June 28 – July 2, 2021, Proceedings*, volume 12730 of *Lecture Notes in Computer Science*, pages 435–459. Springer, 2021. doi:10.1007/978-3-030-79416-3_27.
- 35 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, 17th Annual European Symposium, Copenhagen, Denmark, September 7–9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.
- 36 Virginia Vassilevska-Williams. On Some Fine-Grained Questions in Algorithms and Complexity. In *Proceedings of the International Congress of Mathematicians (ICM 2018)*, pages 3447–34, 2018.
- 37 Louis Weisner. Abstract theory of inversion of finite series. *Transactions of the American Mathematical Society*, 38(3):474–484, 1935.
- 38 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.
- 39 Michał Włodarczyk. Clifford Algebras Meet Tree Decompositions. *Algorithmica*, 81(2):497–518, 2019. doi:10.1007/s00453-018-0489-3.
- 40 Frank Yates. The design and analysis of factorial experiments. *Technical Communication No. 35.*, 1937.
- 41 Or Zamir. Breaking the 2^n Barrier for 5-Coloring and 6-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 113:1–113:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.113.

A Querying a Generalized Convolution

In this section, we prove Theorem 1.4. The main idea is to represent the f -QUERY problem as a matrix multiplication problem, inspired by a graph interpretation of f -QUERY.

Let D be an arbitrary set and $f: D \times D \rightarrow D$. We assume D and f are fixed throughout this section. Let $g, h: D^n \rightarrow \{-M, \dots, M\}$ and $\mathbf{v} \in D^n$ be a f -QUERY instance. We use $\mathbf{a} \parallel \mathbf{b}$ to denote the concatenation of $\mathbf{a} \in D^m$ and $\mathbf{b} \in D^k$. That is $(\mathbf{a}_1, \dots, \mathbf{a}_m) \parallel (\mathbf{b}_1, \dots, \mathbf{b}_k) = (\mathbf{a}_1, \dots, \mathbf{a}_m, \mathbf{b}_1, \dots, \mathbf{b}_k)$. If we assume that n is even, then, for a vector $\mathbf{v} \in D^n$, let $\mathbf{v}^{(\text{high})}, \mathbf{v}^{(\text{low})} \in D^{n/2}$ be the unique vectors such that $\mathbf{v}^{(\text{high})} \parallel \mathbf{v}^{(\text{low})} = \mathbf{v}$. Indeed, to achieve this assumption let n be odd, fix an arbitrary $d \in D$, and define $\tilde{g}, \tilde{h}: D^{n+1} \rightarrow \{-M, \dots, M\}$ as $\tilde{g}(\mathbf{u}_1, \dots, \mathbf{u}_{n+1}) = \llbracket \mathbf{u}_{n+1} = d \rrbracket \cdot g(\mathbf{u}_1, \dots, \mathbf{u}_n)$ and $\tilde{h}(\mathbf{u}_1, \dots, \mathbf{u}_{n+1}) = \llbracket \mathbf{u}_{n+1} = d \rrbracket \cdot h(\mathbf{u}_1, \dots, \mathbf{u}_n)$ for all $\mathbf{u} \in D^{n+1}$. It can be easily verified that $(g \otimes_f h)(\mathbf{v}) = (\tilde{g} \otimes_f \tilde{h})(\mathbf{v} \parallel (f(d, d)))$. Thus, we can solve the f -QUERY instance \tilde{g}, \tilde{h} and $\mathbf{v} \parallel (f(d, d))$ and obtain the correct result.

We first provide the intuition behind the algorithm and then formally show the existence.

Intuition. We define a directed multigraph G where the vertices are partitioned into four layers $L^{(\text{high})}$, $L^{(\text{low})}$, $R^{(\text{low})}$, and $R^{(\text{high})}$. Each of these sets consists of $|D|^{n/2}$ vertices representing every vector in $D^{n/2}$. For ease of notation, we use the vectors to denote the associated vertices; furthermore, the intuition assumes g and h are non-negative. The multigraph G contains the following edges:

- $g(\mathbf{w} \parallel \mathbf{x})$ parallel edges from $\mathbf{w} \in D^{n/2}$ in $L^{(\text{high})}$ to $\mathbf{x} \in D^{n/2}$ in $L^{(\text{low})}$.
- One edge from $\mathbf{x} \in D^{n/2}$ in $L^{(\text{low})}$ to $\mathbf{y} \in D^{n/2}$ in $R^{(\text{low})}$ if and only if $\mathbf{x} \oplus_f \mathbf{y} = v^{(\text{low})}$.
- $h(\mathbf{z} \parallel \mathbf{y})$ parallel edges from $\mathbf{y} \in D^{n/2}$ in $R^{(\text{low})}$ to $\mathbf{z} \in D^{n/2}$ in $R^{(\text{high})}$.
- One edge from $\mathbf{z} \in D^{n/2}$ in $R^{(\text{high})}$ to $\mathbf{w} \in D^{n/2}$ in $L^{(\text{high})}$ if and only if $\mathbf{w} \oplus_f \mathbf{z} = v^{(\text{high})}$.

In the formal proof, we denote the adjacency matrix between $L^{(\text{high})}$ and $L^{(\text{low})}$ by W , between $L^{(\text{low})}$ and $R^{(\text{low})}$ by X , between $R^{(\text{low})}$ and $R^{(\text{high})}$ by Y , and between $R^{(\text{high})}$ and $L^{(\text{high})}$ by Z . See Figure A.1 for an example of this construction.

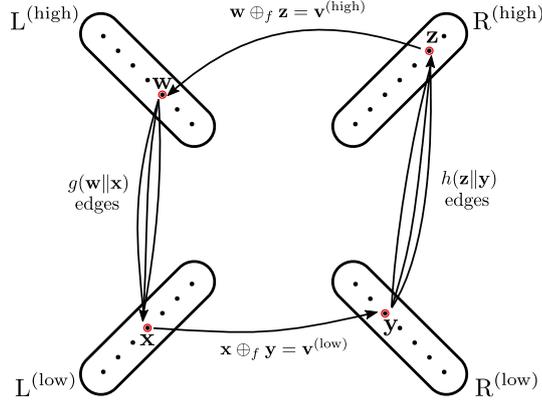
Let $\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z} \in D^{n/2}$ be vertices in $L^{(\text{high})}$, $L^{(\text{low})}$, $R^{(\text{low})}$, and $R^{(\text{high})}$. It can be observed that if $(\mathbf{w} \parallel \mathbf{x}) \oplus_f (\mathbf{y} \parallel \mathbf{z}) \neq \mathbf{v}$, then G does not contain any cycle of the form $\mathbf{w} \rightarrow \mathbf{x} \rightarrow \mathbf{y} \rightarrow \mathbf{z} \rightarrow \mathbf{w}$ as one of the edges (\mathbf{x}, \mathbf{y}) or (\mathbf{z}, \mathbf{w}) is not present in the graph. Conversely, if $(\mathbf{w} \parallel \mathbf{x}) \oplus_f (\mathbf{y} \parallel \mathbf{z}) = \mathbf{v}$, then one can verify that there are $g(\mathbf{w} \parallel \mathbf{x}) \cdot h(\mathbf{z} \parallel \mathbf{y})$ cycles of the form $\mathbf{w} \rightarrow \mathbf{x} \rightarrow \mathbf{y} \rightarrow \mathbf{z} \rightarrow \mathbf{w}$. We therefore expect that $(g \otimes_f h)(\mathbf{v})$ is the number of cycles in G that start at some $\mathbf{w} \in D^{n/2}$ in $L^{(\text{high})}$, have length four, and end at the same vertex \mathbf{w} in $L^{(\text{high})}$ again.

Formal Proof. We use the notation $\text{Mat}_{\mathbb{Z}}(D^{n/2} \times D^{n/2})$ to refer to a $|D|^{n/2} \times |D|^{n/2}$ matrix of integers where we use the values in $D^{n/2}$ as indices. The *transition matrices* of g, h and \mathbf{v} are the matrices $W, X, Y, Z \in \text{Mat}_{\mathbb{Z}}(D^{n/2} \times D^{n/2})$ defined by

$$\begin{aligned} W_{\mathbf{w}, \mathbf{x}} &:= g(\mathbf{w} \parallel \mathbf{x}) & \forall \mathbf{w}, \mathbf{x} \in D^{n/2} \\ X_{\mathbf{x}, \mathbf{y}} &:= \llbracket \mathbf{x} \oplus_f \mathbf{y} = \mathbf{v}^{(\text{low})} \rrbracket & \forall \mathbf{x}, \mathbf{y} \in D^{n/2} \\ Y_{\mathbf{y}, \mathbf{z}} &:= h(\mathbf{z} \parallel \mathbf{y}) & \forall \mathbf{y}, \mathbf{z} \in D^{n/2} \\ Z_{\mathbf{z}, \mathbf{w}} &:= \llbracket \mathbf{w} \oplus_f \mathbf{z} = \mathbf{v}^{(\text{high})} \rrbracket & \forall \mathbf{z}, \mathbf{w} \in D^{n/2} \end{aligned}$$

Recall that the *trace* $\text{tr}(A)$ of a matrix $A \in \text{Mat}_{\mathbb{Z}}(m \times m)$ is defined as $\text{tr}(A) := \sum_{i=1}^m A_{i,i}$. The next lemma formalizes the correctness of this construction.

12:20 Computing Generalized Convolutions Faster Than Brute Force



■ **Figure A.1** Construction of the directed multigraph G . Each vertex in a layer corresponds to the vector in $D^{n/2}$. We highlighted 4 vectors $\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z} \in D^{n/2}$ each in a different layer. Note that the number of 4 cycles that go through all four $\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}$ is equal to $g(\mathbf{w}||\mathbf{x}) \cdot h(\mathbf{z}||\mathbf{y})$. The total number of directed 4-cycles in this graph corresponds to the value $(g \circledast_f h)(\mathbf{v})$ and $\text{tr}(W \cdot X \cdot Y \cdot Z)$.

► **Lemma A.1.** Let $n \in \mathbb{N}$ be an even number, $g, h: D^n \rightarrow \mathbb{Z}$ and $\mathbf{v} \in D^n$. Also, let $W, X, Y, Z \in \text{Mat}_{\mathbb{Z}}(D^{n/2} \times D^{n/2})$ be the transition matrices of g, h and \mathbf{v} . Then,

$$(g \circledast_f h)(\mathbf{v}) = \text{tr}(W \cdot X \cdot Y \cdot Z).$$

Proof. For any $\mathbf{w}, \mathbf{y} \in D^{n/2}$ it holds that,

$$(W \cdot X)_{\mathbf{w}, \mathbf{y}} = \sum_{\mathbf{x} \in D^{n/2}} W_{\mathbf{w}, \mathbf{x}} \cdot X_{\mathbf{x}, \mathbf{y}} = \sum_{\mathbf{x} \in D^{n/2}} \llbracket \mathbf{x} \oplus_f \mathbf{y} = \mathbf{v}^{(\text{low})} \rrbracket \cdot g(\mathbf{w}||\mathbf{x}). \quad (\text{A.1})$$

Similarly, for any $\mathbf{y}, \mathbf{w} \in D^{n/2}$ it holds that,

$$(Y \cdot Z)_{\mathbf{y}, \mathbf{w}} = \sum_{\mathbf{z} \in D^{n/2}} Y_{\mathbf{y}, \mathbf{z}} \cdot Z_{\mathbf{z}, \mathbf{w}} = \sum_{\mathbf{z} \in D^{n/2}} \llbracket \mathbf{w} \oplus_f \mathbf{z} = \mathbf{v}^{(\text{high})} \rrbracket \cdot h(\mathbf{z}||\mathbf{y}). \quad (\text{A.2})$$

Therefore, for any $\mathbf{w} \in D^{n/2}$,

$$\begin{aligned} (W \cdot X \cdot Y \cdot Z)_{\mathbf{w}, \mathbf{w}} &= \sum_{\mathbf{y} \in D^{n/2}} (W \cdot X)_{\mathbf{w}, \mathbf{y}} \cdot (Y \cdot Z)_{\mathbf{y}, \mathbf{w}} \\ &= \sum_{\mathbf{y} \in D^{n/2}} \left(\sum_{\mathbf{x} \in D^{n/2}} \llbracket \mathbf{x} \oplus_f \mathbf{y} = \mathbf{v}^{(\text{low})} \rrbracket \cdot g(\mathbf{w}||\mathbf{x}) \right) \left(\sum_{\mathbf{z} \in D^{n/2}} \llbracket \mathbf{w} \oplus_f \mathbf{z} = \mathbf{v}^{(\text{high})} \rrbracket \cdot h(\mathbf{z}||\mathbf{y}) \right) \\ &= \sum_{\mathbf{x}, \mathbf{y}, \mathbf{z} \in D^{n/2}} \llbracket \mathbf{x} \oplus_f \mathbf{y} = \mathbf{v}^{(\text{low})} \rrbracket \cdot \llbracket \mathbf{w} \oplus_f \mathbf{z} = \mathbf{v}^{(\text{high})} \rrbracket \cdot g(\mathbf{w}||\mathbf{x}) \cdot h(\mathbf{z}||\mathbf{y}) \\ &= \sum_{\mathbf{x}, \mathbf{y}, \mathbf{z} \in D^{n/2}} \llbracket (\mathbf{w}||\mathbf{x}) \oplus_f (\mathbf{z}||\mathbf{y}) = \mathbf{v}^{(\text{high})} || \mathbf{v}^{(\text{low})} \rrbracket \cdot g(\mathbf{w}||\mathbf{x}) \cdot h(\mathbf{z}||\mathbf{y}), \end{aligned}$$

where the second equality follows by (A.1) and (A.2). Thus,

$$\begin{aligned}
\text{tr}(W \cdot X \cdot Y \cdot Z) &= \sum_{\mathbf{w} \in D^{n/2}} (W \cdot X \cdot Y \cdot Z)_{\mathbf{w}, \mathbf{w}} \\
&= \sum_{\mathbf{w} \in D^{n/2}} \sum_{\mathbf{x}, \mathbf{y}, \mathbf{z} \in D^{n/2}} \llbracket (\mathbf{w} \parallel \mathbf{x}) \oplus_f (\mathbf{z} \parallel \mathbf{y}) = \mathbf{v} \rrbracket \cdot g(\mathbf{w} \parallel \mathbf{x}) \cdot h(\mathbf{z} \parallel \mathbf{y}) \\
&= \sum_{\mathbf{u}, \mathbf{t} \in D^n} \llbracket \mathbf{u} \oplus_f \mathbf{t} = \mathbf{v} \rrbracket \cdot g(\mathbf{u}) \cdot h(\mathbf{t}) \\
&= (g \otimes_f h)(\mathbf{v}). \quad \blacktriangleleft
\end{aligned}$$

Now we have everything ready to give the algorithm for f -QUERY.

Proof of Theorem 1.4. The algorithm for solving f -QUERY works in two steps:

1. Compute the transition matrices W , X , Y , and Z of g , h and \mathbf{v} as described above.
2. Compute and return $\text{tr}(W \cdot X \cdot Y \cdot Z)$.

By Lemma A.1 this algorithm returns $(g \otimes_f h)(\mathbf{v})$. Computing the transition matrices in Step 1 requires $\tilde{O}(|D|^n \cdot \text{polylog}(M))$ time. Observe the maximal absolute values of an entry in the transition matrices is M . The computation of $W \cdot X \cdot Y \cdot Z$ in Step 2 requires three matrix multiplications of $|D|^{n/2} \times |D|^{n/2}$ matrices, which can be done in $\tilde{O}((|D|^{n/2})^\omega \cdot \text{polylog}(M))$ time. Thus, the overall running time of the algorithm is $\tilde{O}(|D|^{\omega \cdot n/2} \cdot \text{polylog}(M))$. \blacktriangleleft

B Proof of Theorem 2.5

In this section we prove Theorem 2.5. We let M be the absolute value of largest integer on the output of functions $g: L^n \rightarrow \mathbb{Z}$ and $h: R^n \rightarrow \mathbb{Z}$. We let $K := \prod_{i=1}^n r_i$. We crucially rely on the following result by van Rooij [33].

► **Theorem B.1** ([33, Lemma 3]). *Let p denote a prime such that in the field \mathbb{F}_p , the r_i -th root of unity exists for each $i \in [n]$. For two given functions $g, h: \mathbb{Z}_{r_1} \times \cdots \times \mathbb{Z}_{r_n} \rightarrow \mathbb{Z}$, we can compute their cyclic convolution modulo p (that is, return a function ϕ such that $\phi(\mathbf{q}) = (g \odot h)(\mathbf{v}) \pmod p$ for every $\mathbf{q} \in \mathbb{Z}_{r_1} \times \cdots \times \mathbb{Z}_{r_n}$) in time $\mathcal{O}(K \log(Kp))$ (assuming a r_i -th primitive root of unity ω_i in the field \mathbb{F}_p is given for all $i \in [n]$).*

The basic idea behind the proof is to compute $g \odot h$ modulo p_i for a sufficiently large number of distinct small primes p_i . If $\prod_i p_i > 2^n$, then the values of $g \odot h$ can be uniquely recovered using the Chinese Remainder Theorem.

► **Theorem B.2** (Chinese Remainder Theorem). *Let m_1, \dots, m_ℓ denote a sequence of integers that are pairwise coprime and define $M := \prod_{i \in [\ell]} m_i$. Also let $0 \leq a_i < m_i$ for all $i \in [\ell]$. Then there is a unique number $0 \leq s < M$ such that*

$$s \equiv a_i \pmod{m_i}$$

for all $i \in [\ell]$. Moreover, there is an algorithm that, given m_1, \dots, m_ℓ and a_1, \dots, a_ℓ , computes the number s in time $\mathcal{O}((\log M)^2)$.

Let $m := \lceil \log(3 \cdot |L|^n \cdot |R|^n \cdot M^2) \rceil$. We compute the list of the first m primes $p_1 < \cdots < p_m$ such that $p_i \equiv 1 \pmod K$ for all $i \in [m]$. By the Prime Number Theorem for Arithmetic Progressions (see, e.g., [3]) we get that $p_m = \mathcal{O}(\varphi(K) \cdot m \cdot \log m)$ where φ denotes Euler's totient function. In particular, $p_m = \mathcal{O}(K \cdot m \cdot \log m)$ because $\varphi(K) \leq K$. Since prime testing can be done in polynomial time, we can find the sequence p_1, \dots, p_m in time $\mathcal{O}(K \cdot m \cdot (\log m)^c)$ for some constant c .

12:22 Computing Generalized Convolutions Faster Than Brute Force

Next, for every $i \in [m]$ and $j \in [n]$, we compute a r_j -th root of unity in \mathbb{F}_{p_i} as follows. First observe that such a root of unity exists since r_j divides $p_i - 1$. For every $i \in [m]$ we first find the prime factors $q_{i,1}, \dots, q_{i,\ell_i}$ of $p_i - 1$ by iterating over every number in \mathbb{Z}_{p_i} and checking if it is both prime and divides $p_i - 1$. This can be done in time $\mathcal{O}(p_i \cdot \text{polylog } p_i)$. Next, we simply iterate over all elements $x \in \mathbb{F}_{p_i}$ and test whether a given element x is a r_j -th root of unity in time $(\log p_i)^{\mathcal{O}(1)}$. So overall, computing all roots of unity for every p_i ($i \in [m]$) can be done in time

$$\sum_{i=1}^m \left(\mathcal{O}(p_i \cdot \text{polylog } p_i) + \sum_{j=1}^n p_i \cdot (\log p_i)^{\mathcal{O}(1)} \right) = m \cdot n \cdot p_m \cdot (\log p_m)^{\mathcal{O}(1)} = K \cdot (n + m + \log K)^{\mathcal{O}(1)}.$$

Now, for every $i \in [m]$ and $\mathbf{q} \in \mathbb{Z}_{r_1} \times \dots \times \mathbb{Z}_{r_n}$, we compute

$$(g \odot h)(\mathbf{q})^{(i)} := (g \odot h)(\mathbf{q}) \pmod{p_i}$$

using Theorem B.1 in time $\mathcal{O}(m \cdot K \cdot \log(K \cdot p_m)) = K \cdot (m + \log K)^{\mathcal{O}(1)}$.

Finally, we can recover $(g \odot h)(\mathbf{q})$ for every \mathbf{q} by the Chinese Remainder Theorem in time $\mathcal{O}(K \cdot m^2)$. Note that $\prod_{i \in [m]} p_i > 2^m \geq M$ which implies that all numbers are indeed uniquely recovered. In total, this achieves the desired running time. \blacktriangleleft

C Proof of Lemma 3.5

The idea is to use a dynamic programming algorithm loosely inspired by Yates algorithm [40].

Define $X^{(\ell)} = \{(\bar{p}, \mathbf{q}) \mid \bar{p} \in [m]^\ell, \mathbf{q} \in \mathbb{Z}_{\bar{p}_1} \times \dots \times \mathbb{Z}_{\bar{p}_\ell}\}$ for every $\ell \in \{0, \dots, n\}$. We use $X^{(\ell)}$ to define a dynamic programming table $\text{DP}^{(\ell)}: X^{(\ell)} \times L^{n-\ell} \rightarrow \mathbb{Z}$ for every $\ell \in \{0, \dots, n\}$ by:

$$\text{DP}^{(\ell)}[(\bar{p}_1, \dots, \bar{p}_\ell), (\mathbf{q}_1, \dots, \mathbf{q}_\ell)][\mathbf{t}_{\ell+1}, \dots, \mathbf{t}_n] := \sum_{\substack{\mathbf{t}_1 \in A_{\bar{p}_1} \\ \vdots \\ \mathbf{t}_\ell \in A_{\bar{p}_\ell}}} \left(\prod_{i=1}^{\ell} \llbracket \sigma_{\bar{p}_i}(\mathbf{t}_i) = \mathbf{q}_i \rrbracket \right) \cdot g(\mathbf{t}_1, \dots, \mathbf{t}_n).$$

The tables $\text{DP}^{(0)}, \text{DP}^{(1)}, \dots, \text{DP}^{(n)}$ are computed consecutively where the computation of $\text{DP}^{(\ell)}$ relies on the values of $\text{DP}^{(\ell-1)}$ for any $\ell \in [n]$. Observe that $g_{\bar{p}}(\mathbf{q}) = \text{DP}^{(n)}[(\bar{p}_1, \dots, \bar{p}_n), (\mathbf{q}_1, \dots, \mathbf{q}_n)][\varepsilon]$ for every \bar{p} and \mathbf{q} , which means that computing $\text{DP}^{(n)}$ is equivalent to computing the projection functions $g_{\bar{p}}$ of g for every type \bar{p} .⁷

It holds that $\text{DP}^{(0)}[\varepsilon, \varepsilon][\mathbf{t}] = g(\mathbf{t})$. Hence, $\text{DP}^{(0)}$ can be trivially computed in $|L|^n$ time. We use the following straightforward recurrence to compute $\text{DP}^{(\ell)}$:

$$\begin{aligned} \text{DP}^{(\ell)}[(\bar{p}_1, \dots, \bar{p}_\ell), (\mathbf{q}_1, \dots, \mathbf{q}_\ell)][\mathbf{t}_{\ell+1}, \dots, \mathbf{t}_n] = \\ \sum_{\mathbf{t}_\ell \in A_{\bar{p}_\ell}} \llbracket \sigma_{\bar{p}_\ell}(\mathbf{t}_\ell) = \mathbf{q}_\ell \rrbracket \cdot \text{DP}^{(\ell-1)}[(\bar{p}_1, \dots, \bar{p}_{\ell-1}), (\mathbf{q}_1, \dots, \mathbf{q}_{\ell-1})][\mathbf{t}_\ell, \dots, \mathbf{t}_n]. \end{aligned} \quad (\text{C.1})$$

A dynamic programming algorithm which computes $\text{DP}^{(n)}$ can be easily derived from (C.1) and the formula for $\text{DP}^{(0)}$. The total number of states in the dynamic programming table $\text{DP}^{(\ell)}$ is

$$\left(\sum_{\bar{p} \in [m]^\ell} (k_{\bar{p}_1} \cdot \dots \cdot k_{\bar{p}_\ell}) \right) \cdot |L|^{n-\ell} = (k_1 + \dots + k_m)^\ell \cdot |L|^{n-\ell} = \text{cost}(\mathcal{P})^\ell \cdot |L|^{n-\ell}.$$

This is bounded by $\text{cost}(\mathcal{P})^n + |L|^n$ for every $\ell \in [n]$. To transition between states we spend polynomial time per entry because we assume that $|L| = \mathcal{O}(1)$. Hence, we can compute $g_{\bar{p}}$ for every \bar{p} in $\tilde{\mathcal{O}}((\text{cost}(\mathcal{P})^n + |L|^n) \cdot \text{polylog}(M))$ time. \blacktriangleleft

⁷ We use ε to denote the vector of length 0.

Exact Exponential Algorithms for Clustering Problems

Fedor V. Fomin  

Department of Informatics, University of Bergen, Norway

Petr A. Golovach  

Department of Informatics, University of Bergen, Norway

Tanmay Inamdar¹  

Department of Informatics, University of Bergen, Norway

Nidhi Purohit¹ 

Department of Informatics, University of Bergen, Norway

Saket Saurabh 

The Institute of Mathematical Sciences, HBNI, Chennai, India

Department of Informatics, University of Bergen, Norway

Abstract

In this paper we initiate a systematic study of exact algorithms for some of the well known clustering problems, namely k -MEDIAN and k -MEANS. In k -MEDIAN, the input consists of a set X of n points belonging to a metric space, and the task is to select a subset $C \subseteq X$ of k points as *centers*, such that the sum of the distances of every point to its nearest center is minimized. In k -MEANS, the objective is to minimize the sum of *squares* of the distances instead. It is easy to design an algorithm running in time $\max_{k \leq n} \binom{n}{k} n^{\mathcal{O}(1)} = \mathcal{O}^*(2^n)$ (here, $\mathcal{O}^*(\cdot)$ notation hides polynomial factors in n). In this paper we design first non-trivial exact algorithms for these problems. In particular, we obtain an $\mathcal{O}^*((1.89)^n)$ time *exact* algorithm for k -MEDIAN that works for any value of k . Our algorithm is quite general in that it does not use any properties of the underlying (metric) space – it does not even require the distances to satisfy the triangle inequality. In particular, the same algorithm also works for k -MEANS. We complement this result by showing that the running time of our algorithm is asymptotically optimal, up to the base of the exponent. That is, unless the Exponential Time Hypothesis fails, there is no algorithm for these problems running in time $2^{\epsilon n} \cdot n^{\mathcal{O}(1)}$.

Finally, we consider the “facility location” or “supplier” versions of these clustering problems, where, in addition to the set X we are additionally given a set of m candidate centers (or facilities) F , and objective is to find a subset of k centers from F . The goal is still to minimize the k -MEDIAN/ k -MEANS/ k -CENTER objective. For these versions we give a $\mathcal{O}(2^n (mn)^{\mathcal{O}(1)})$ time algorithms using subset convolution. We complement this result by showing that, under the Set Cover Conjecture, the “supplier” versions of these problems do not admit an exact algorithm running in time $2^{(1-\epsilon)n} (mn)^{\mathcal{O}(1)}$.

2012 ACM Subject Classification Theory of computation \rightarrow Facility location and clustering; Theory of computation \rightarrow Parameterized complexity and exact algorithms

Keywords and phrases clustering, k -median, k -means, exact algorithms

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.13

Funding The research leading to these results has received funding from the Research Council of Norway via the project BWCA (grant no. 314528) and the European Research Council (ERC) via grant LOPPRE, reference 819416.

¹ Part of this work was done when the two authors were visiting IMSc, Chennai.



1 Introduction

Clustering is a fundamental area in the domain of optimization problems with numerous applications. In this paper, we focus on some of the most fundamental problems in the clustering literature, namely k -MEDIAN, k -MEANS, and k -CENTER. We formally define the optimization version k -MEDIAN.

k -MEDIAN

Input: Given a metric space (X, d) , where $X = \{x_1, \dots, x_n\}$ is a collection of n points, with distance function d on X and a positive integer k .

Task: Find a pair (C, P) , where $C = \{c_1, \dots, c_k\} \subseteq X$ is a set of *centers* and P is a partition of X into k subsets $\{X_1, \dots, X_k\}$ (clusters). Here, X_i is the cluster corresponding to the center $c_i \in C$. The goal is to minimize the following cost, over all pairs (C, P) .

$$\text{cost}(C, P) = \sum_{i=1}^k \sum_{x \in X_i} d(c_i, x)$$

k -MEANS is a variant of k -MEDIAN, where the only difference is that we want to minimize the sum of squares of the distances, i.e., $\sum_{i=1}^k \sum_{x \in X_i} (d(c_i, x))^2$. In k -CENTER, the objective is to minimize the maximum distance of a point and its nearest center, i.e., $\max_{i=1}^k \max_{x \in X_i} d(c_i, x)$.

The special cases of k -MEDIAN have a long history, and they are known in the literature as Fermat-Weber problem [23, 24]. A recent formulation of k -MEANS can be traced back to Steinhaus [20] and MacQueen [19]. Lloyd proposed a heuristic algorithm [17] for k -MEANS that is extremely simple to implement for euclidean spaces, and it remains popular even today. k -CENTER was proved to be NP-complete by Hsu and Nemhauser [10]. All three problems have been studied from the perspective of approximation algorithms for last several decades. These three problems – as well as several of their generalizations – are known to admit constant factor approximations in polynomial time. More recently, these problems have also been studied from the perspective of Fixed-Parameter Tractable (FPT) algorithms, where one allows the running times of the form $f(k) \cdot n^{\mathcal{O}(1)}$ for some computable function f . k -MEDIAN and k -MEANS are known to admit improved approximation guarantees using FPT algorithms [4], and these approximation guarantees are tight up to certain complexity-theoretic assumptions.

A result that initiated this study is an *exact* algorithm for k -CENTER¹ by Agarwal and Procopiuc [1], who give an $n^{\mathcal{O}(k^{1-\frac{1}{d}})}$ time algorithm in \mathbb{R}^d . In particular, in two dimensional space, their algorithm runs in $2^{\mathcal{O}(\sqrt{n} \log n)}$ time for any value of k , i.e., in *sub-exponential* time. This led us towards a natural question, namely, studying the complexity of k -MEDIAN, k -MEANS, k -CENTER in general metrics.

Note that it is easy to design an exact algorithm that runs in time $\binom{n}{k} \cdot n^{\mathcal{O}(1)}$ – it simply enumerates all sets of centers of size k , and the corresponding partition of X into clusters is obtained by assigning each point to its nearest center. Then, we simply return the solution with the minimum cost. However, note that when k belongs to the range $n/2 \pm o(n)$, $\binom{n}{k} \simeq 2^n$. Thus, the naïve algorithm has running time $\mathcal{O}^*(2^n)$ in the worst case.

¹ We note that the result of [1] holds for a slightly different variant, where the centers can be placed anywhere in \mathbb{R}^d . This formulation is more natural and standard in euclidean spaces.

For many problems, the running time of $\mathcal{O}^*(2^n)$ is often achievable by a brute-force enumeration of all the solutions. However, for many NP-hard problems, it is often possible to obtain improved running times. The field of exact algorithms for NP-hard problems is several decades old. In 2003, Woeginger wrote a survey [25] on this topic, which revived the field. This eventually led to a plethora of new results and techniques, such as subset convolution [2], measure and conquer [8], and monotone local search [7]. A detailed survey on this topic can be found in a textbook by Kratsch and Fomin [16]. We study the aforementioned classical clustering problems from this perspective. In other words, we ask whether the classical clustering problems such as k -MEDIAN and k -MEANS admit moderately exponential-time algorithms, i.e., algorithms with running time $c^n \cdot n^{\mathcal{O}(1)}$ for a constant $c < 2$ that is as small as possible. We indeed answer this question in the affirmative, leading to the following theorem.

► **Theorem 1.** *There is an exact algorithm for k -MEDIAN (k -MEANS) in time $(1.89)^n n^{\mathcal{O}(1)}$, where n is the number of points in X .*

To explain the idea behind this result, consider the following fortuitous scenario. Suppose that the optimal solution only contains clusters of size exactly 2. In this case, it is easy to solve the problem optimally by reducing the problem to finding a minimum-weight matching in the complete graph defining the metric ². Note that the problem of finding Minimum-Weight Perfect Matching is known to be polynomial-time solvable by the classical result of Edmonds [13]. This idea can also be extended if the optimal solution only contains clusters of size 1 and 2, by finding matching in an auxiliary graph. However, the idea does not generalize to clusters of size 3 and more, since we need to solve a problem that has a flavor similar to the 3-dimensional matching problem or the “star partition” problem, which are known to be NP-hard [9, 3, 15]. Nevertheless, if the number of points belonging to the clusters of size at least 3 is *small*, one can “guess” these points, and solve the remaining points using matching. However, the number of points belonging to the clusters of size at least 3 can be quite large – it can be as high as n . But note that the number of *centers* corresponding to clusters of size at least 3 can be at most $n/3$. We show that “guessing” the subset of centers of such clusters is sufficient (as opposed to guessing *all* the points in such clusters), in the sense that an optimal clustering of the “residual” instance can be found – again – by finding a minimum-weight matching in an appropriately constructed auxiliary graph.

We briefly explain the idea behind the construction of this auxiliary graph. Note that in order to find an optimal clustering in the “residual” instance, we need to figure out the following things: (1) the set of points that are involved in clusters of size 1, i.e., *singleton* clusters, (2) the pairs of points that become clusters of size 2, and (3) for each center c_i of a cluster of size at least 3, the set of at least two additional points that are connected to c_i . We find the set of points of type (1) by matching them to a set of *dummy* points with zero-weight edges. The pairs of points involved in clusters of size 2 naturally correspond to a matching, such that the weight of each edge corresponds to the distance between the corresponding pair of points. Finally, to find points of type (3), we make an appropriate number of *copies* of each guessed center c_i that will be matched to the corresponding points. Although the high-level idea behind the construction of the graph is very natural, it is non-trivial to construct the graph such that a minimum-weight perfect matching in the auxiliary graph exactly corresponds to an optimal clustering (assuming we guess the centers

² Note that the cluster-center always belongs to its own cluster, which implies that a cluster of size 2 contains one *additional* point. This immediately suggests the connection to minimum-weight matching.

correctly). Thus, this construction pushes the boundary of applicability of matching in order to find an optimal clustering. Since the minimum-weight perfect matching problem can be solved in polynomial time, the running time of our algorithm is dominated by guessing the set of centers of clusters of size at least 3. As mentioned previously, the number of such centers is at most $n/3$, which implies that the number of guesses is at most $\binom{n}{n/3} \leq (1.89)^n$, which dominates the running time of our algorithm. We describe this result in Section 3. We complement these moderately exponential algorithms by showing that these running times are asymptotically optimal. Formally, assuming the Exponential Time Hypothesis (ETH), as formulated by Impagliazzo and Paturi [11], we show that these problems do not admit an algorithms running in time $2^{o(n)} \cdot n^{\mathcal{O}(1)}$. A formal definition of ETH is given in Section 2, and we prove the ETH-hardness result in Section 4.

We note that our algorithm as well as the hardness result also holds for k -CENTER. However, it is folklore that the *exact* versions of k -CENTER and DOMINATING SET are equivalent. Thus, using the currently best known algorithm for DOMINATING SET by Iwata [12], it is possible to obtain an $\mathcal{O}^*((1.4689)^n)$ time algorithm for k -CENTER.

We also consider a “facility location” or “supplier” version, which is a generalization of the clustering problems defined above. In this setting, we are given a set of clients (or points) X , and a set of facilities (or centers) F . In general the sets X and F may be different, or even disjoint. In these versions, the set of k centers C must be chosen from F , i.e., $C \subseteq F$. We formally state the “supplier” version of k -MEDIAN, which we call k -MEDIAN FACILITY LOCATION³.

k-MEDIAN FACILITY LOCATION

Input: Given a metric space $(X \cup F, d)$, where $X = \{x_1, \dots, x_n\}$ of n points, called clients, F is a set of m centers, and a positive integer k .

Task: Find a pair (C, P) , where $C = \{c_1, \dots, c_k\} \subseteq F$ of size at most k and P is a partition of X into k subsets $\{X_1, \dots, X_k\}$ (clusters) such that each client in cluster X_i is assigned to center c_i so as to minimize the k -median cost of clustering, defined as follows:

$$\text{cost}(C, P) = \sum_{i=1}^k \sum_{x \in X_i} d(c_i, x)$$

It is also possible to define the analogous versions of k -MEANS and k -CENTER— the latter has been studied in the approximation literature under the name of k -SUPPLIER. In this paper, we show that these “facility location” versions of k -MEDIAN/ k -MEANS/ k -CENTER are computationally harder, as compared to the normal versions, in the following sense. Consider the concrete example of k -MEDIAN and k -MEDIAN FACILITY LOCATION. As mentioned earlier, we beat the “trivial” bound of $\mathcal{O}(2^n)$, by giving a $\mathcal{O}((1.89)^n)$ time algorithm for k -MEDIAN. On the other hand, we show that for k -MEDIAN FACILITY LOCATION, it is not possible to obtain a $2^{(1-\epsilon)n} \cdot (mn)^{\mathcal{O}(1)}$ time algorithm for any fixed $\epsilon > 0$ (note that $m = |F|$ is the number of facilities and $n = |X|$ is the number of clients). For showing this result, we use the SET COVER CONJECTURE, which is a complexity theoretic hypothesis proposed by Cygan et al. [5]. We match this lower bound by designing an algorithm with running time $2^n \cdot (mn)^{\mathcal{O}(1)}$

³ We note that a slight generalization of this problem has been considered by Jain and Vazirani [14], who called it “a common generalization of k -median and Facility Location”, and gave a constant approximation in polynomial time.

under some mild assumptions. The details are in Section 6. While this algorithm is not obvious, it is a relatively straightforward application of the subset convolution technique. This algorithm also works for the supplier versions of k -MEANS and k -CENTER; however, again there is a much simpler algorithm for k -SUPPLIER with a similar running time.

Finally, note that designing an algorithm for the supplier versions with running time $2^m \cdot (mn)^{O(1)}$ is trivial by simple enumeration. It is not known whether the base of the exponent can be improved by showing an algorithm with running time $(2 - \epsilon)^m (mn)^{O(1)}$ for some fixed $\epsilon > 0$, or whether this is not possible assuming a similar complexity-theoretic hypothesis, such as SET COVER CONJECTURE, or Strong Exponential Time Hypothesis (SETH). We leave this open for a future work.

2 Preliminaries

We denote by $G = (V(G), E(G))$ a graph with vertex set $V(G)$ and edge set $E(G)$. Cardinality of a set S denoted by $|S|$ is the number of elements of the set. We denote an (undirected) edge between vertices u and v as uv . We denote by $N(v) = \{u \in V(G) \mid (u, v) \in E(G)\}$ be the *open neighbourhood* (or simply neighbourhood) of v , and let $N[v] = N(v) \cup \{v\}$ be the *closed neighbourhood* of v .

A *matching* M of a graph G is a set of edges such that no two edges have common vertices. A vertex $v \in V(G)$ is said to be saturated by M if there is an edge in M incident to v , otherwise it is said to be unsaturated. We also say that M saturates v . We say that a vertex u is matched to a vertex v in M if there is an edge $e \in M$ such that $e = (u, v)$. A perfect matching in a graph G is a matching which saturates every vertex in G . Given a weight function $w: E(G) \rightarrow \mathbb{R}_{\geq 0}$, the minimum weight perfect matching problem is to find a perfect matching M (if it exists) of minimum weight $w(M) = \sum_{e \in M} w(e)$. It is well known to be solvable in polynomial time by the Blossom algorithm of Edmonds [13].

A q -CNF formula $\phi = C_1 \wedge \dots \wedge C_m$ is a boolean formula over n variables $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$, such that each clause C_i is a disjunction of at most q literals of the form x_i or $\neg x_i$, for some $1 \leq i \leq n$. In a q -SAT instance we are given a q -CNF formula ϕ , and the question is to decide whether ϕ is satisfiable. Impagliazzo and Paturi [11] formulated the following hypothesis, called Exponential Time Hypothesis. Note that this ETH is a stronger assumption than $P \neq NP$.

Exponential Time Hypothesis (ETH) states that q -SAT, $q \geq 3$ cannot be solved within a running time of $2^{o(n)}$ or $2^{o(m)}$, where n is the number of variables and m is the number of clauses in the input q -CNF formula.

3 Proof of Theorem 1

Before delving into the proof of Theorem 1, we discuss the approach at a high level. We begin by “guessing” a subset of centers from an (unknown) optimal solution. For each guess, the problem of finding the best (i.e., minimum-cost) clustering that is “compatible” with the guess is reduced to finding a minimum weight perfect matching in an auxiliary graph G . The graph G is constructed in such a way that this clustering can be extracted by essentially looking at the minimum-weight perfect matching. Note that MINIMUM WEIGHT PERFECT MATCHING problem is well known to be solvable in polynomial time by the Blossom algorithm of Edmonds [13]. Finally, we simply return a minimum-cost clustering found over all guesses.

Let us fix some optimal k -median solution and let k_1^* , k_2^* and k_3^* be a partition of k , where k_1^* : the number of clusters of size exactly 1, call *Type1*; k_2^* : the number of clusters of size exactly 2, call *Type2*; and k_3^* : the number of clusters of size at least 3, call *Type3*. Let

13:6 Exact Exponential Algorithms for Clustering Problems

$C_3^* \subseteq X$ be *Type3* centers, and say $C_3^* = \{c_1, \dots, c_{k_3^*}\}$. Observe that number of clusters with *Type3* centers is at most $\frac{n}{3}$. Suppose not, then the number of clusters with *Type3* centers is greater than $\frac{n}{3}$. Each *Type3* cluster contains at least three points. This contradicts that the number of input points is n .

Algorithm. First, we guess the partition of k into k_1, k_2, k_3 as well as a subset $C_3 \subseteq X$ of size at most $n/3$. For each such guess (k_1, k_2, k_3, C_3) , we construct the auxiliary graph G (as defined subsequently) corresponding to this guess, and compute a minimum weight perfect matching M in G . Let M^* be a minimum weight perfect matching over *all* the guesses. We extract the corresponding clustering (C^*, P^*) from M^* (also explained subsequently), and return as an optimal solution of the given instance.

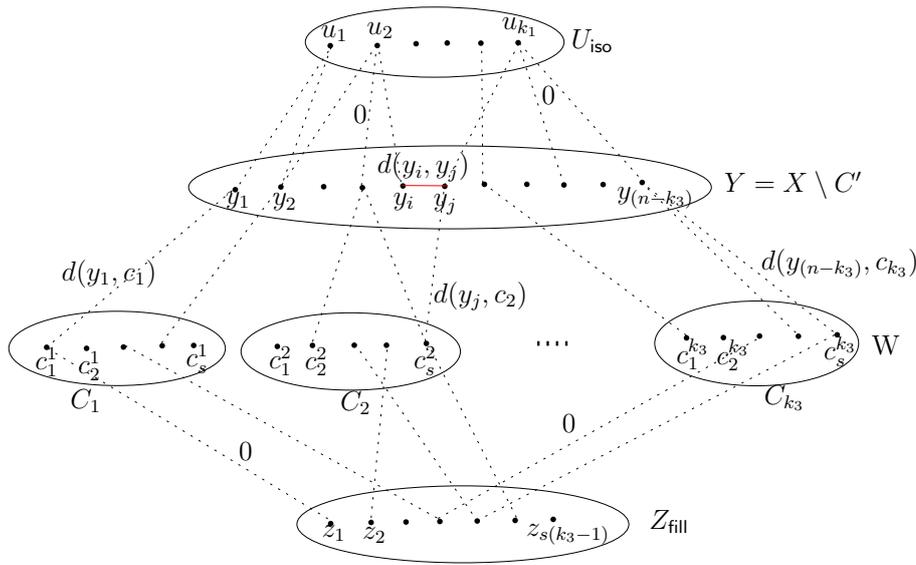
Running time. Note that there are at most $\mathcal{O}(k^2)$ tuples (k_1, k_2, k_3) such that $k_1 + k_2 + k_3 \leq k$ (note that k_i 's are non-negative integers). Furthermore, there are at most $\sum_{i=0}^{n/3} \binom{n}{i} \leq (1.89)^n$ subsets of X of size at most $n/3$. Finally, constructing the auxiliary graph, and finding a minimum-weight perfect matching takes polynomial time. Thus, the running time is dominated by the number of guesses for C_3 , which implies that we can bound the running time of our algorithm by $\mathcal{O}^*((1.89)^n)$.

Construction of Auxiliary Graph. From now on assume that our algorithm made the right guesses, i.e., suppose that $(k_1, k_2, k_3) = (k_1^*, k_2^*, k_3^*)$ and $C_3^* = C'$. Then, we initialize the *Type3* centers by placing each center from C' into a separate cluster. At this point, to achieve this, we reduce the problem to the classical MINIMUM WEIGHT PERFECT MATCHING on an auxiliary graph G , which we define as follows. (See Figure 1 for an illustration of the construction).

- For each $i \in \{1, \dots, k_3\}$, construct a set of $s = n - k_3 - 2k_2 - k_1$ vertices $C_i = \{c_1^i, \dots, c_s^i\}$. Denote $W = \cup_{i=1}^{k_3} C_i$; the block of vertices C_i corresponds to center c_i .
- Let $Y = X \setminus C'$, that is, a set consisting of unclustered points in X . Observe $|Y| = n - k_3$. Denote $Y = \{y_1, \dots, y_{(n-k_3)}\}$. For simplicity, we slightly abuse the notation by keeping the vertices in G same as points in Y . That is, for each $i \in \{1, \dots, (n - k_3)\}$, place a vertex y_i in the set Y . Make each y_i adjacent to all vertices of W .
- For each $i \in \{1, \dots, k_1\}$, construct an auxiliary vertex u_i . Denote $U_{\text{iso}} = \{u_1, \dots, u_{k_1}\}$. Make each u_i adjacent to every vertex of Y .
- Construct a set of $s(k_3 - 1)$ vertices, $Z_{\text{fill}} = \{z_1, \dots, z_{s(k_3-1)}\}$, that we call fillers and make vertices of Z_{fill} adjacent to the vertices of W .

We define edge weights. For an edge $(u, v) \in E(G)$, we will use $w(u, v)$ to denote $w((u, v))$ to avoid clutter.

- For every $i \in \{1, \dots, (n - k_3)\}$ and every $j \in \{1, \dots, k_3\}$ set $w(y_i, c_h^j) = d(y_i, c_j)$ for $h \in \{1, \dots, s\}$, i.e, weight of all edges joining y_i in Y with the vertices of C_i corresponding to center c_j .
- For every $i, j \in \{1, \dots, n - k_3\}$, $i \neq j$, set $w(y_i, y_j) = d(y_i, y_j)$, i.e, the weight of edges between vertices of Y .
- For every $i \in \{1, \dots, k_1\}$ and $j \in \{1, \dots, (n - k_3)\}$, set $w(u_i, y_j) = 0$, i.e., the edges incident to the vertices of U_{iso} have zero weights.
- For every $i \in \{1, \dots, s(k_3 - 1)\}$ and $j \in \{1, \dots, k_3\}$, $w(z_i c_h^j) = 0$, for $h \in \{1, \dots, s\}$, i.e., the edges incident to the fillers have zero weights.



■ **Figure 1** Illustration of the graph G produced in the reduction from k -MEDIAN to MINIMUM WEIGHT PERFECT MATCHING. To avoid clutter, we only show some representative edges. Recall that we guess the set of k_3 centers of type 3, and corresponding to each such center c_i , we add a set C_i consisting of s copies corresponding to that center. Next, we have the set Y corresponding to $n - k_3$ unclustered points. Finally, U_{iso} and Z_{fill} consist of auxiliary vertices in order to ensure a perfect matching. The weights of vertices among Y correspond to the corresponding original distance; whereas the weight of an edge between $y_\ell \in Y$, and a copy c_i^j corresponding to a type 3 center c_i is defined to be $d(y_\ell, c_i)$. The weights of all other edges are equal to zero.

► **Lemma 2.** *The graph G has a perfect matching.*

Proof. We construct a set $M \subseteq E(G)$ that saturates every vertex in G .

Note that $|U_{\text{iso}}| < |Y|$ and every vertex of U_{iso} is adjacent to every vertex of Y . Therefore, we can construct $M_1 \subseteq E(G)$ by arbitrarily mapping each vertex of U_{iso} to a distinct vertex of Y . Clearly, M_1 is matching saturating vertices of U_{iso} . Since $|U_{\text{iso}}| = k_1$, M_1 saturates k_1 vertices of Y . Denote by Y' the set of vertices of Y that are not saturated by M_1 . Observe $|Y'| = s + 2k_2$.

Every vertex of Z_{fill} is adjacent to every vertex of W and $|Z_{\text{fill}}| < |W|$. Construct $M_2 \subseteq E(G)$ by arbitrarily mapping each vertex of Z_{fill} to a distinct vertex of W . Thus, M_2 is a matching which saturates every vertex of Z_{fill} and since $|Z_{\text{fill}}| = s(k_3 - 1)$, it also saturates $s(k_3 - 1)$ vertices of W . Denote by W' the set of vertices of W that is not saturated by M_2 . Observe $|W'| = s$. Recall, every vertex of W' is adjacent to every vertex of Y' and note that $|W'| < |Y'|$. Therefore, construct $M_3 \subseteq E(G)$ by arbitrarily matching each vertex of W' with a distinct vertex of Y' .

Thus, the matching M_3 saturates s vertices in both the sets W' and Y' . Denote $M' = M_1 \cup M_2 \cup M_3$. Clearly, the vertices of U_{iso} , W and Z_{fill} are saturated by M' .

Denote by $Y'' = Y \setminus Y'$ the set of vertices of Y that are not saturated by M' . Note that $|Y''| = 2k_2$. Consider $M_4 \subseteq E(G)$ which maps these $2k_2$ vertices to each other. We set $M = M' \cup M_4$. It is easy to see that M is a perfect matching. ◀

We next show one-to-one correspondence between perfect matchings of G and k -median clusterings of X .

► **Lemma 3.** *Let $\text{OPT}_{\text{mm}}(G)$ = weight of minimum weight perfect matching, and $\text{OPT}_{k\text{med}}(X)$ = optimal clustering cost of k -median clustering of X . Then, $\text{OPT}_{\text{mm}}(G) = \text{OPT}_{k\text{med}}(X)$.*

Proof. In the forward direction, let M denote a minimum weight perfect matching $M \subseteq E(G)$. We construct a k -median clustering of X of same cost.

Observe that each vertex of Z_{fill} is only adjacent to the vertices of W and $|Z_{\text{fill}}| < |W|$. Let $W_1 \subseteq W$ be a set of vertices matched to vertices of Z_{fill} . Since G has a perfect matching, it saturates Z_{fill} , where $|Z_{\text{fill}}| = s(k_3 - 1)$. Then, $|W_1| = s(k_3 - 1)$. Let $W_2 = W \setminus W_1$ be set of vertices matched to vertices of Y . Clearly, $|W_2| = s$.

For every $i \in \{1, \dots, (n - k_3)\}$, vertex $y_i \in Y$ is saturated by M . Therefore, we construct the k -median clustering $\{X_1, \dots, X_k\}$ of X , where each $X_i \in \{\text{Type1}, \text{Type2}, \text{Type3}\}$, for $i \in \{1, \dots, k\}$ as follows.

Let $Y' \subseteq Y$ be the set of vertices that are matched to vertices of U_{iso} in M , where $|U_{\text{iso}}| = k_1 < |Y|$. Corresponding to each such vertex in Y' , select a center in the solution C , call $C_{\text{Type1}} = \{c_{\text{Type1}}^1, \dots, c_{\text{Type1}}^{k_1}\}$. Correspondingly, also construct a singleton cluster $X_i = \{c_{\text{Type1}}^i\}$, for $i \in \{1, \dots, k_1\}$. Let X_{Type1} denote set of all *Type1* clusters.

We now construct *Type3* clusters: Let $Y'_i \subseteq Y$ be the set of vertices matched to set C_i , for $i \in \{1, \dots, k_3\}$ in M . Consider $X_i = Y'_i \cup \{c_i\}$. Clearly, X_i , for $i \in \{1, \dots, k_3\}$ corresponds to *Type3* clusters in X . Let X_{Type3} denote set of all *Type3* clusters. Recall, we already guess set $C' = \{c_1, \dots, c_{k_3}\}$, that is, *Type3* centers correctly.

Lastly, we construct clusters of *Type2*. Denote by Y'' set of unclustered points in Y . Observe these points form a set of k_2 disjoint edges in M . Arbitrarily, select one of the endpoint of each edge as a center in the solution C , call $C_{\text{Type2}} = \{c_{\text{Type2}}^1, \dots, c_{\text{Type2}}^{k_2}\}$. That is, for an edge $(y_1, y_2) \in M$, where $y_1, y_2 \in Y''$, select center as y_1 or y_2 . Then construct a cluster X_i , for $i \in \{1, \dots, k_2\}$ by placing both the endpoints of the edge in the same cluster. Denote by X_{Type2} the set of all *Type2* clusters.

Clearly, $X_i \in \{\text{Type1}, \text{Type2}, \text{Type3}\}$, for $i \in \{1, \dots, k\}$ is a partition of X . Note, since *Type1* clusters are isolated points, therefore, they contribute zero to the total cost of clustering. Now we upper bound the cost of the obtained k -median clustering:

$$\sum_{i=1}^k \sum_{x \in X_i} d(c_i, x) = \sum_{i=1}^{k_2} \sum_{y \in X_{\text{Type2}}} d(c_{\text{Type2}}^i, y) + \sum_{i=1}^{k_3} \sum_{y \in X_{\text{Type3}}} d(c_i, y) = \text{OPT}_{\text{mm}}(G).$$

For the reverse direction, consider a k -median clustering $\{X_1, \dots, X_k\}$ of X into $\{\text{Type1}, \text{Type2}, \text{Type3}\}$ clusters of X such that $|\text{Type1}| = k_1$, $|\text{Type2}| = k_2$ and $|\text{Type3}| = k_3$ and $C' = \{c_1, \dots, c_{k_3}\}$, that is, centers of *Type3* clusters with $\text{OPT}_{k\text{med}}(X)$. We construct a perfect matching $M \subseteq E(G)$ of G as follows.

Observe that each *Type1* cluster is a singleton cluster. Construct $M_1 \subseteq E(G)$ by iterating over each singleton vertex in Y correspond to each cluster and matched it to a distinct vertex in U_{iso} . Since $|\text{Type1}| = |U_{\text{iso}}| = k_1$, M_1 is a matching saturating set U_{iso} . Also, M_1 saturates k_1 vertices in Y .

Corresponding to each *Type2* cluster, construct $M_2 \subseteq E(G)$ by adding an edge between both the end vertices in Y . Clearly, M_2 is a disjoint set of k_2 edges in G and saturates $2k_2$ vertices in Y .

Denote $Y' \subseteq Y$ be the set of vertices matched by $M_1 \cup M_2$. Clearly, $|Y'| = k_1 + 2k_2$. Let $Y'' = Y \setminus Y'$ be the set of remaining unmatched vertices in Y . Then, $|Y''| = |Y| - |Y'| = n - k_3 - 2k_2 - k_1 = s$.

Note, we already guessed $C' = \{c_1, \dots, c_{k_3}\}$ and we have a cluster X_i corresponding to each C_i , for $i \in \{1, \dots, k_3\}$. Construct $M_3 \subseteq E(G)$ by matching each vertex of $X_i \setminus \{c_i\}$ in Y'' to a distinct copy of c_i in W . Since $|Y''| < |W|$, M_3 saturates Y'' . Let $W_1 \subseteq W$ be the set of vertices saturated by M_3 . Note that $|Y''| = s$, then $|W_1| = s$. Let $W_2 = W \setminus W_1$ be the set of vertices not saturated by M_3 , where $|W| = sk_3$. Then, $|W_2| = s(k_3 - 1)$. Every vertex of Z_{fill} is only adjacent to every vertex of W (in particular of W_2). We construct $M_4 \subseteq E(G)$ by matching each vertex of Z_{fill} to a distinct vertex of W_2 . Since $|Z_{\text{fill}}| = |W_2| = s(k_3 - 1)$, M_4 saturates Z_{fill} and W_2 .

To evaluate the weight of M , recall that the edges of G incident to set U_{iso} and filler vertices Z_{fill} have zero weights, that is, $w(M_1) = w(M_4) = 0$. Then

$$\begin{aligned} w(M) &= w(M_2) + w(M_3) = \sum_{e \in M_2} w(e) + \sum_{e \in M_3} w(e) \\ &= \sum_{c_i: X_i \in X_{\text{Type2}}} \sum_{y \in C_i} d(y, c_i) + \sum_{c_i: X_i \in X_{\text{Type3}}} \sum_{y \in C_i} d(y, c_i) \\ &= \text{OPT}_{\text{kmed}}(X). \end{aligned}$$

It is straightforward to see that the construction of the graph G from an instance (X, d) of k -MEDIAN can be done in polynomial time. Then, because a perfect matching of minimum weight of the graph G can be found in polynomial time [13] and the total number of guesses is at most $(1.89)^n n^{\mathcal{O}(1)}$, k -MEDIAN can be solved exactly in $(1.89)^n n^{\mathcal{O}(1)}$ time. This completes the proof of the theorem. \blacktriangleleft

► **Remark 4.** Note that even if the distances satisfy the triangle inequality, the sum of *squares* of distances do not. Nevertheless, our algorithm also works for k -MEANS, where we want to minimize the sum of squares of distances; or even more generally, if we want to minimize the sum of z -th powers of distances, for some fixed $z \geq 1$. In fact, our algorithm works for non-metric distance functions – it is easy to modify construction of graph G so that it works with asymmetric distance functions, which are quite popular in the context of asymmetric traveling salesman problem [21, 22]. Finally, we note that it may be possible to improve the running time (i.e., the base of the exponent) using the metric properties of distances, and we leave this open for a future work. However, in the next section, we show the running time of an exact algorithm cannot be substantially improved, i.e., to $\mathcal{O}^*(2^{o(n)})$.

4 ETH Hardness

In this section, we establish result around the (im)possibility of solving k -MEDIAN problem in subexponential time in the number of points. For this, we use the result of Lokshtanov et al. [18] which states that, assuming ETH, DOMINATING SET problem cannot be solved in time $2^{o(n)}$ time, where n is the number of vertices of graph.

Given an unweighted, undirected graph $G = (V, E)$, a dominating set S is a subset of V such that each $v \in V$ is dominated by S , that is, we either have $v \in S$ or there exists an edge $(uv) \in E(G)$ such that $u \in S$. The decision version of DOMINATING SET is defined as follows.

DOMINATING SET

Input: Given an unweighted, undirected graph $G(V, E)$, positive integer k .
Task: Determine whether G has a dominating set of size at most k .

Lokshtanov et. al [18] proved the following result.

13:10 Exact Exponential Algorithms for Clustering Problems

► **Proposition 5** ([18]). *Assuming ETH, there is no $2^{o(n)}$ time algorithm for DOMINATING SET problem, where n is the number of vertices of G .*

We use this known fact about DOMINATING SET to prove the following.

► **Theorem 6.** *k -MEDIAN cannot be solved in time $2^{o(n)}$ time unless the exponential-time hypothesis fails, where n is the number of points in X .*

Proof. We give a reduction from DOMINATING SET problem to k -MEDIAN problem. Let $(G = (V, E), k)$ be the given instance of DOMINATING SET. We assume that there is no dominating set in G of size at most $k - 1$. This assumption is without loss of generality, since we can use the following reduction iteratively for $k' = 1, 2, \dots, k$, which only incurs a polynomial overhead.

Now we construct an instance (X, d) of k -MEDIAN as follows. First, let $X = V(G)$, i.e., we treat each vertex of the graph as a point in the metric space, and we use the terms vertex and point interchangeably. Recall that the graph $G = (V, E)$ is unweighted, but we suppose that the weight of every edge in $E(G)$ is 1. Then, we let d be the shortest path metric in G . The following observations are immediate.

► **Observation 7.**

- For all $u \in V(G)$, $d(u, u) = 0$.
- For all distinct $u, v \in V(G)$, $d(u, v) = 1 \iff (u, v) \in E(G)$, and $d(u, v) \geq 2 \iff (u, v) \notin E(G)$.

We now show that there is a dominating set of size k iff there is a k -median clustering of cost exactly $n - k$.

In the forward direction, let $S \subseteq V(G)$ be a dominating set of size k . We obtain the corresponding k -median clustering as follows. We let $S = \{c_1, c_2, \dots, c_k\}$ to be the set of centers. For a center $c_i \in S$, we define $X'_i = N[c_i]$. Since S is a dominating set, every vertex in $V(G) \setminus S$ has a neighbor in S . Therefore, $\bigcup_{1 \leq i \leq k} X'_i = V(G)$. Now, we remove all *other* centers except for c_i from the set X'_i . Furthermore, if a vertex belongs to multiple X'_i 's, we arbitrarily keep it only a single X'_i . Let $\{X_1, X_2, \dots, X_k\}$ be the resulting partition of $V(G)$. Observe that in the resulting clustering, centers pay a cost of zero, whereas every other vertex has a center at distance 1. Therefore, the cost of the clustering is exactly $n - k$.

In the other direction, let $(S, \{X_1, X_2, \dots, X_k\})$ be a given k -median clustering of cost $n - k$. We claim that S is a dominating set of size k . Consider any vertex $u \in V(G) \setminus S$, and suppose $u \in X_i$ corresponding to the center c_i . Since $u \notin S$, $d(u, S) \geq d(u, c_i) \geq 1$. This holds for all $n - k$ points of $V(G) \setminus S$. Now, if $u \in X_i$, and $d(u, c_i) > 1$ for some vertex $u \in V(G) \setminus S$, then this contradicts the assumption that the given clustering has cost $n - k$. This implies that every $u \in V(G) \setminus S$ has a center in S at distance exactly 1, i.e., u has a neighbor in S . This concludes the proof.

This reduction takes polynomial time. Observe that the number of points in the resulting instance is equal to n , the number of vertices in G . Therefore, if there is an algorithm for k -MEDIAN with running time subexponential in the number of points n then it would give a $2^{o(n)}$ time algorithm for DOMINATING SET, which would refute ETH, via Proposition 5. ◀

5 SeCoCo Hardness

In this section, we consider the variant of k -MEDIAN, which we call k -MEDIAN FACILITY LOCATION. Recall that in this problem, we are given a metric space $(X \cup F, d)$, where X is a set of n clients, F is a set of m centers and integer $k > 0$. The goal is to select a set $C \subseteq F$ of k centers and assign each client in X to a center in C , such that the k -median cost of clustering is minimized.

We show that there is no algorithm solves k -MEDIAN FACILITY LOCATION problem in time $\mathcal{O}(2^{(1-\epsilon)n} \text{poly}(m))$, for every fixed $\epsilon > 0$. For this, we use the SET COVER CONJECTURE by Cygan et al. [5].

The decision version of SET COVER problem is defined as follows.

SET COVER

Input: Given a universe $\mathcal{U} = \{u_1, \dots, u_n\}$ of n elements and a family $\mathcal{S} = \{S_1, \dots, S_m\}$ of m subsets of \mathcal{U} and an integer k

Task: Determine whether there is a set cover of size at most k .

To state SET COVER CONJECTURE [5] more formally, let Δ -SET COVER denote the SET COVER problem where all the sets have size at most $\Delta > 0$.

► **Conjecture 8.** SET COVER CONJECTURE (SeCoCo)[5]. For every fixed $\epsilon > 0$ there is $\Delta(\epsilon) > 0$, such that no algorithm (even randomized) solves Δ -SET COVER in time $\mathcal{O}(2^{(1-\epsilon)n} \cdot \text{poly}(m))$.

Using this result, we show the following.

► **Theorem 9.** Assuming SET COVER CONJECTURE, for any fixed $\epsilon > 0$, there is no $\mathcal{O}(2^{(1-\epsilon)n} \cdot \text{poly}(m))$ time algorithm for k -MEDIAN FACILITY LOCATION, where n is the number of clients.

Proof. We give a reduction from SET COVER to k -MEDIAN FACILITY LOCATION problem.

Given an instance $(\mathcal{U}, \mathcal{S})$ of SET COVER problem, where $\mathcal{U} = \{u_1, \dots, u_n\}$ and $\mathcal{S} = \{S_1, \dots, S_m\}$, such that $S_i \subseteq \mathcal{U}$, we create an instance of k -MEDIAN FACILITY LOCATION by building a bipartite graph $G = ((X \cup F), E)$ as follows.

- For each element $u_i \in \mathcal{U}$, we create a client, say x_i , for $i \in \{1, \dots, n\}$. Denote $X = \{x_1, \dots, x_n\}$.
 - For each set $S_i \in \mathcal{S}$, we create a center, say c_i , for $i \in \{1, \dots, m\}$. Denote $F = \{c_1, \dots, c_m\}$.
 - For every $i \in \{1, \dots, n\}$ and every $j \in \{1, \dots, m\}$, if $u_i \in S_j$, then connect corresponding x_i and c_j with an edge of weight 1, i.e., client x_i pays cost 1 when assigned to facility c_j .
- This finishes the construction of G . Now, let d be the shortest path metric in graph G .

We show that there is set cover of size at most k if and only if there is k -median clustering of cost n .

In the forward direction, assume there is a set cover $\mathcal{S}' \subseteq \mathcal{S}$ of size at most k . Assume $\mathcal{S}' = \{S_1, \dots, S_k\}$. For a set S_i , we make the corresponding vertex $c_i \in F$ a center. Then, we create its corresponding cluster X_i as follows. We add all the points x_j such that $(c_i x_j) \in E$. Finally, we make the clusters X_i pairwise disjoint, by arbitrarily choosing exactly one cluster for every client, if the client is present in multiple clusters. Clearly, $\{X_1, \dots, X_k\}$ is a partition of X . We now calculate the cost of the obtained k -median clustering.

$$\sum_{i=1}^k \sum_{x \in X_i} d(c_i, x) = \sum_{i=1}^k |X_i| = n.$$

13:12 Exact Exponential Algorithms for Clustering Problems

In the reverse direction, suppose there is a k -median clustering $\{X_1, \dots, X_k\}$ of X of cost n . Let $C = \{c_1, \dots, c_k\} \subseteq F$ be a set of centers. Every client must be at distance at least 1 from its corresponding center. We claim that each client in a cluster is at distance exactly 1 from its corresponding center. Suppose not, then there exists a client with distance strictly greater than 1 from its center. The total number of clients is n . This contradicts that the cost of k -median clustering is n . Thus, every element is chosen in some set corresponding to set C . Therefore, a subfamily $\mathcal{S}' \subseteq \mathcal{S}$ corresponding to set C forms a cover of \mathcal{U} . Since $|C| = k$, \mathcal{S}' is a cover of \mathcal{U} of size at most k .

Clearly, this reduction takes polynomial time. Furthermore, observe that the number of clients in the resulting instance is same as the number of elements in \mathcal{U} . Therefore, if there is an $\mathcal{O}(2^{(1-\epsilon)n} \cdot \text{poly}(m))$ time algorithm for k -MEDIAN FACILITY LOCATION then it would give a $\mathcal{O}(2^{(1-\epsilon)n} \cdot \text{poly}(m))$ time algorithm for SET COVER, which, in turn, refutes SET COVER CONJECTURE. \blacktriangleleft

We briefly note that the same hardness construction also shows a similar hardness result for the “supplier” versions of k -MEANS and k -CENTER.

6 A $2^n \cdot \text{poly}(m, n)$ time Algorithm for k -Median Facility Location

Let $(X \cup F, d)$ be a given instance of k -MEDIAN FACILITY LOCATION, where $n = |X|$ denotes the number of clients, and $m = |F|$ denotes the number of centers. In this section, we give a $2^n \cdot \text{poly}(m, n)$ -time exact algorithm, under a mild assumption that any distance in the input is a non-negative integer that is bounded by a polynomial in the input size.⁴ Let $M := n \cdot D$, where D denotes the maximum inter-point distance in the input. Note that $M = \text{poly}(m, n)$.

We define k functions $\text{cost}_1, \text{cost}_2, \dots, \text{cost}_k : 2^X \rightarrow M$, where $\text{cost}_i(Y)$ denotes the minimum cost of clustering the clients of Y into at most i clusters. In other words, $\text{cost}_i(Y)$ is the optimal i -MEDIAN FACILITY LOCATION cost, restricted to the instance $(Y \cup F, d)$. First, notice that $\text{cost}_1(Y)$ is simply the minimum cost of clustering all points of Y into a single cluster. This value can be computed in $\mathcal{O}(mn)$ time by iterating over all centers in F , and selecting the center c that minimizes the cost $\sum_{p \in Y} d(p, c)$. Thus, the values $\text{cost}_1(Y)$ for all subsets $Y \subseteq X$ can be computed in $\mathcal{O}(2^n mn)$ time. Next, we have the following observation.

► **Observation 10.** For any $Y \subseteq X$ and for any $1 \leq i \leq k$,

$$\text{cost}_i(Y) = \min_{\substack{A \cup B = Y \\ A \cap B = \emptyset}} \text{cost}_{i-1}(A) + \text{cost}_1(B).$$

Note that since we are interested in clustering of Y into *at most* i clusters, we do not need to “remember” the set of facilities realizing $\text{cost}_{i-1}(A)$ and $\text{cost}_1(B)$ in Observation 10. Next, we discuss the notion of subset convolution that will be used to compute $\text{cost}_i(\cdot)$ values that is faster than the naïve computation.

Subset Convolutions. Given two functions $f, g : 2^X \rightarrow \mathbb{Z}$, the *subset convolution* of f and g is the function $(f * g) : 2^X \rightarrow \mathbb{Z}$, defined as follows.

$$\forall Y \subseteq X : \quad (f * g)(Y) = \sum_{\substack{A \cup B = Y \\ A \cap B = \emptyset}} f(A) \cdot g(B) \tag{1}$$

⁴ Since the integers are encoded in binary, this implies that the length of the encoding of any distance is $\mathcal{O}(\log(m) + \log(n))$.

It is known that, given all the 2^n values of f and g in the input, all the 2^n values of $f * g$ can be computed in $\mathcal{O}(2^n \cdot n^3)$ arithmetic operations, see e.g., Theorem 10.15 in the Parameterized Algorithms book [6]. This is known as *fast subset convolution*. Now, let $(f \oplus g)(Y) = \min_{\substack{A \cup B = Y \\ A \cap B = \emptyset}} f(A) + g(B)$. We observe that $f \oplus g$ is equal to the subset convolution $f * g$ in the integer min-sum semiring $(\mathbb{Z} \cup \{\infty\}, \min, +)$, i.e., in Equation (1), we use the mapping $+$ \mapsto \min , and \cdot \mapsto $+$. This, combined with a simple “embedding trick” enables one to compute all values of $f \oplus g : 2^X \rightarrow \{-N, \dots, N\}$ in time $2^n n^{\mathcal{O}(1)} \cdot \mathcal{O}(N \log N \log \log N)$ using fast subset convolution – see Theorem 10.17 of [6]. Finally, Observation 10 implies that cost_i is exactly $\text{cost}_{i-1} \oplus \text{cost}_1$, and we observe that the function values are upper bounded by $n \cdot D = M$. We summarize this discussion in the following proposition.

► **Proposition 11.** *Given all the 2^n values of cost_{i-1} and cost_1 in the input, all the 2^n values of cost_i can be computed in time $2^n n^{\mathcal{O}(1)} \cdot \mathcal{O}(M \log M \log \log M)$.*

Using Proposition 11, we can compute all the 2^n values of $\text{cost}_2(\cdot)$, using the pre-computed values $\text{cost}_1(\cdot)$. Then, we can use the values of $\text{cost}_2(\cdot)$ and $\text{cost}_1(\cdot)$ to compute the values of $\text{cost}_3(\cdot)$. By iterating in this manner $k - 1 \leq n$ times, we compute the values of $\text{cost}_k(\cdot)$ for all 2^n subsets of k , and the overall time is upper bounded by $2^n mn^{\mathcal{O}(1)} \cdot \mathcal{O}(M \log M \log \log M)$, which is $2^n \cdot \text{poly}(m, n)$, if $M = \text{poly}(m, n)$. Note that $\text{cost}_k(X)$ corresponds to the optimal cost of k -MEDIAN FACILITY LOCATION. Finally, the computed values of the functions $\text{cost}_i(\cdot)$ can be used to also compute a clustering $\{X_1, X_2, \dots, X_k\}$ of X , and the corresponding centers $\{c_1, c_2, \dots, c_k\}$. We omit the straightforward details.

► **Theorem 12.** *k -MEDIAN FACILITY LOCATION can be solved optimally in $2^n \cdot \text{poly}(m, n)$ time, assuming the distances are integers that are bounded by polynomial in the input size.*

Note that the algorithm does not require the underlying distance function to satisfy the triangle inequality. In particular, we obtain an analogous result the “facility location” version of the k -MEANS objective. Finally, the algorithm works for k -SUPPLIER, which is a similar variant of k -CENTER. However, in this case there is a much simpler reduction to SET COVER which gives an $2^n \cdot \text{poly}(m, n)$ time algorithm. For this, we first “guess” the optimal radius r , and define a set system that consists of balls of radius r around the given centers. We omit the details.

References

- 1 Pankaj K Agarwal and Cecilia Magdalena Procopiu. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002.
- 2 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009.
- 3 Jérémie Chalopin and Daniël Paulusma. Packing bipartite graphs with covers of complete bipartite graphs. *Discret. Appl. Math.*, 168:40–50, 2014. doi:10.1016/j.dam.2012.08.026.
- 4 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight fpt approximations for k -median and k -means. *arXiv preprint*, 2019. arXiv:1904.12334.
- 5 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as cnf-sat. *ACM Trans. Algorithms*, 12, 2016.
- 6 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5(4). Springer, 2015.
- 7 Fedor V Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. *Journal of the ACM (JACM)*, 66(2):1–23, 2019.

13:14 Exact Exponential Algorithms for Clustering Problems

- 8 Fedor V Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM (JACM)*, 56(5):1–32, 2009.
- 9 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 10 Wen-Lian Hsu and George L Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979.
- 11 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 9:367–375, 2001.
- 12 Yoichi Iwata. A faster algorithm for dominating set analyzed by the potential method. In *International Symposium on Parameterized and Exact Computation*, pages 41–54. Springer, 2011.
- 13 Edmonds Jack. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.
- 14 Kamal Jain and Vijay V Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *Journal of the ACM (JACM)*, 48(2):274–296, 2001.
- 15 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.
- 16 Dieter Kratsch and FV Fomin. *Exact exponential algorithms*. Springer-Verlag Berlin Heidelberg, 2010.
- 17 Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.
- 18 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bull. EATCS*, 105:41–72, 2011.
- 19 James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1(14), pages 281–297. Oakland, CA, USA, 1967.
- 20 Hugo Steinhaus et al. Sur la division des corps matériels en parties. *Bull. Acad. Polon. Sci.*, 1(804):801, 1956.
- 21 Ola Svensson, Jakub Tarnawski, and László A Végh. A constant-factor approximation algorithm for the asymmetric traveling salesman problem. *Journal of the ACM (JACM)*, 67(6):1–53, 2020.
- 22 Vera Traub and Jens Vygen. An improved approximation algorithm for atsp. In *Proceedings of the 52nd annual ACM SIGACT symposium on theory of computing*, pages 1–13, 2020.
- 23 Wikipedia. Geometric median – Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Geometric%20median&oldid=1061179886>, 2022. [Online; accessed 21-April-2022].
- 24 Wikipedia. Weber problem – Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Weber%20problem&oldid=916663348>, 2022. [Online; accessed 21-April-2022].
- 25 Gerhard J Woeginger. Exact algorithms for np-hard problems: A survey. In *Combinatorial optimization – eureka, you shrink!*, pages 185–207. Springer, 2003.

Domination and Cut Problems on Chordal Graphs with Bounded Leafage

Esther Galby ✉

TU Hamburg, Germany

Dániel Marx ✉ 

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Philipp Schepper ✉ 

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Roohani Sharma ✉ 

Max Planck Institute for Informatics, SIC, Saarbrücken, Germany

Prafullkumar Tale ✉ 

Indian Institute of Science Education and Research, Pune, India

Abstract

The leafage of a chordal graph G is the minimum integer ℓ such that G can be realized as an intersection graph of subtrees of a tree with ℓ leaves. We consider structural parameterization by the leafage of classical domination and cut problems on chordal graphs. Fomin, Golovach, and Raymond [ESA 2018, Algorithmica 2020] proved, among other things, that DOMINATING SET on chordal graphs admits an algorithm running in time $2^{\mathcal{O}(\ell^2)} \cdot n^{\mathcal{O}(1)}$. We present a conceptually much simpler algorithm that runs in time $2^{\mathcal{O}(\ell)} \cdot n^{\mathcal{O}(1)}$. We extend our approach to obtain similar results for CONNECTED DOMINATING SET and STEINER TREE. We then consider the two classical cut problems MULTICUT WITH UNDELETABLE TERMINALS and MULTIWAY CUT WITH UNDELETABLE TERMINALS. We prove that the former is W[1]-hard when parameterized by the leafage and complement this result by presenting a simple $n^{\mathcal{O}(\ell)}$ -time algorithm. To our surprise, we find that MULTIWAY CUT WITH UNDELETABLE TERMINALS on chordal graphs can be solved, in contrast, in $n^{\mathcal{O}(1)}$ -time.

2012 ACM Subject Classification Theory of computation \rightarrow Fixed parameter tractability

Keywords and phrases Chordal Graphs, Leafage, FPT Algorithms, Dominating Set, MultiCut with Undeleteable Terminals, Multiway Cut with Undeleteable Terminals

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.14

Related Version *Full Version:* <https://arxiv.org/abs/2208.02850> [23]

Funding Research supported by the European Research Council (ERC) consolidator grant No. 725978 SYSTEMATICGRAPH.

Esther Galby and Prafullkumar Tale: Part of the work was carried out when the authors were Post-Doctoral Researchers at CISPA Helmholtz Center for Information Security, Germany.

Philipp Schepper: Part of Saarbrücken Graduate School of Computer Science, Germany.

1 Introduction

The intersection graph of a family \mathcal{F} of nonempty sets is the graph whose vertices are the elements of \mathcal{F} with two vertices being adjacent if and only if their corresponding sets intersect. The most natural and famous example of such intersection graphs are *interval graphs* where \mathcal{F} is a collection of subpaths of a path. Due to their applicability in scheduling, interval graphs have received a considerable attention in the realm of algorithmic graph theory. One useful characterization of an interval graph is that its maximal cliques can be linearly ordered such



© Esther Galby, Dániel Marx, Philipp Schepper, Roohani Sharma, and Prafullkumar Tale; licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 14; pp. 14:1–14:24

Leibniz International Proceedings in Informatics



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

that for every vertex, the maximal cliques containing that vertex occur consecutively [25]. This property proves very useful for the design of polynomial-time dynamic programming based or greedy algorithms on interval graphs.

Consider the generalization where \mathcal{F} is a collection of subtrees of a tree instead of subpaths of a path. In this case, the corresponding class of intersection graphs is exactly that of *chordal graphs* [47, 24, 11]. Recall that a graph is chordal if every cycle of length at least 4 has a chord. Often, the algorithms of the types mentioned in the previous paragraph fail to generalize to this superclass as witnessed by the following problems that admit polynomial-time algorithms on interval graphs but are NP-complete on chordal graphs: DOMINATING SET [13, 8], CONNECTED DOMINATING SET [3, 48], STEINER TREE [3, 48], MULTICUT WITH UNDELETABLE TERMINALS [28, 44], SUBSET FEEDBACK VERTEX SET (SUBSET FVS) [45, 21], LONGEST CYCLE [34, 27]¹, LONGEST PATH [32], COMPONENT ORDER CONNECTIVITY [19], s -CLUB CONTRACTION [26], INDEPENDENT SET RECONFIGURATION [5], BANDWIDTH [36], CLUSTER VERTEX DELETION [35]. Also, GRAPH ISOMORPHISM on chordal graphs is polynomial-time equivalent to the problem on general graphs whereas it admits a linear-time algorithm on interval graphs [40].

The problems above remain hard even on *split graphs*, another well-studied subclass of chordal graphs. A graph is a split graph if its vertex set can be partitioned into a clique and an independent set. The collection of split graphs is a (proper) subset of the class of intersection graphs where \mathcal{F} is a collection of substars of a star. As interval graphs are intersection graphs of subpaths of a path (a tree with two leaves) and split graphs are intersection graphs of substars of a star (a tree with arbitrary number of leaves), a natural question to consider is what happens to these problems on subclasses of chordal graphs that are intersection graphs of subtrees of a tree with a bounded number of leaves. Motivated by such questions, we consider the notion of *leafage* introduced by Lin et al. [39]: the leafage of a chordal graph G is the minimum integer ℓ such that G can be realized as an intersection graph of a collection \mathcal{F} of subtrees of a tree that has ℓ leaves. Note that the leafage of interval graphs is at most 2 while split graphs have unbounded leafage. Thus the leafage measures, in some sense, how close a chordal graph is to an interval graph. Alternately, an FPT or XP algorithm parameterized by the leafage can be seen as a generalization of the algorithm on interval graphs.

Related Work. Habib and Stacho [29] showed that we can compute the leafage of a connected chordal graph in polynomial time. Their algorithm also constructs a corresponding *representation tree*² T with the minimum number of leaves. In recent years, researchers have studied the structural parameterization of various graph problems on chordal graphs parameterized by the leafage. Fomin et al. [20] and Arvind et al. [2] proved, respectively, that the DOMINATING SET and GRAPH ISOMORPHISM problems on chordal graphs are FPT parameterized by the leafage. Barnetson et al. [4] and Papadopoulos and Tzimas [46] presented XP-algorithms running in time $n^{\mathcal{O}(\ell)}$ for FIRE BREAK and SUBSET FVS on chordal graphs, respectively. Papadopoulos and Tzimas [46] also proved that SUBSET FVS is W[1]-hard when parameterized by the leafage. Hochstättler et al [31] showed that we can compute the neighborhood polynomial of a chordal graph in $n^{\mathcal{O}(\ell)}$ -time.

It is known that the size of *asteroidal set* in a chordal graph is upper bounded by its leafage [39]. See [30, 1] for the relationship between leafage and other structural properties of chordal graphs. Kratsch and Stewart [37] proved that we can effectively 2ℓ -approximate

¹ See Exercise 2 in Chapter 6 in [27].

² We present formal definitions of the terms used in this section in Section 2.

bandwidth of chordal graphs of leafage ℓ . Chaplick and Stacho [14] generalized the notion of leafage to *vertex leafage* and proved that, unlike leafage, it is hard to determine the optimal vertex leafage of a given chordal graph. Figueiredo et al. [18] proved that DOMINATING SET, CONNECTED DOMINATING SET and STEINER TREE are FPT on chordal graphs when parameterized by the size of the solution plus the vertex leafage, provided that a tree representation with optimal vertex leafage is given as part of the input.

Our Results. We consider well-studied domination and cut problems on chordal graphs. As our first result, we prove that DOMINATING SET on chordal graphs of leafage at most ℓ admits an algorithm running in time $2^{\mathcal{O}(\ell)} \cdot n^{\mathcal{O}(1)}$. This improves upon the existing algorithm by Fomin et al. [20, Theorem 9] which runs in time $2^{\mathcal{O}(\ell^2)} \cdot n^{\mathcal{O}(1)}$. Despite being significantly simpler than the algorithm in [20], our algorithm in fact solves the RED-BLUE DOMINATING SET problem, a well-known generalization of DOMINATING SET. In this generalized version, an input is a graph G with a partition (R, B) of its vertex set and an integer k , and the objective is to find a subset D of R that dominates every vertex in B , i.e., $B \subseteq N(D)$. We further use this algorithm to solve other related domination problems.

► **Theorem 1.** DOMINATING SET, CONNECTED DOMINATING SET, and STEINER TREE can be solved in $2^{\mathcal{O}(\ell)} \cdot n^{\mathcal{O}(1)}$ on chordal graphs of leafage at most ℓ .

The reductions in [8] and [48] used to prove that these problems are NP-complete on chordal graphs imply that these problems do not admit $2^{o(n)}$, and hence $2^{o(\ell)} \cdot n^{\mathcal{O}(1)}$, algorithms unless the ETH fails.

Arguably, the two most studied cut problems are MULTICUT and MULTIWAY CUT. In the MULTICUT problem, an input is graph G , a set of terminal pairs $P \subseteq V(G) \times V(G)$ and an integer k , and the objective is to find a subset $S \subseteq V(G)$ of size at most k such that no pair of vertices in P is connected in $G - S$. In the MULTIWAY CUT problem, instead of terminal pairs, we are given a terminal set P and the objective is to find a subset $S \subseteq V(G)$ of size at most k such that no two vertices in P are connected in $G - S$. These problems and variations of them have received a considerable attention which lead to the development of new techniques [41, 42, 9, 16, 15]. Misra et al. [43] studied the parameterized complexity of these problems on chordal graphs. Guo et al. [28] proved that MULTICUT WITH DELETABLE TERMINALS is NP-complete on interval graphs, thereby implying that this problem is paraNP-hard when parameterized by the leafage. We consider the MULTICUT WITH UNDELETABLE TERMINALS problem and prove the following result.

► **Theorem 2.** MULTICUT WITH UNDELETABLE TERMINALS on chordal graphs is $W[1]$ -hard when parameterized by the leafage ℓ and assuming the ETH, does not admit an algorithm running in time $f(\ell) \cdot n^{o(\ell)}$ for any computable function f . However, it admits an XP-algorithm running in time $n^{\mathcal{O}(\ell)}$.

Next, we focus on the MULTIWAY CUT WITH UNDELETABLE TERMINALS problem. We find it somewhat surprising that the classical complexity of this problem on chordal graphs was not known. Bergougnoux et al. [7], using the result in [20], proved that the problem admits an XP-algorithm when parameterized by the leafage³. Our next result significantly improves upon this and [43, Theorem 2] which states that the problem admits a polynomial kernel when parameterized by the solution size.

³ See the discussion after Corollary 2 on page 1388 in [7].

► **Theorem 3.** MULTIWAY CUT WITH UNDELETABLE TERMINALS can be solved in $n^{\mathcal{O}(1)}$ -time on chordal graphs.

A well-known trick to convert an instance of MULTIWAY CUT WITH DELETABLE TERMINALS into an instance of MULTIWAY CUT WITH UNDELETABLE TERMINALS is to add a pendant vertex to each terminal, remove that vertex from the set of terminals, and make the newly added vertex a terminal. As this reduction converts a chordal graph into another chordal graph, Theorem 3 implies that MULTIWAY CUT WITH DELETABLE TERMINALS is also polynomial-time solvable on chordal graphs. Another closely related problem is SUBSET FVS which is NP-complete on split graphs [45]. To the best of our knowledge, this is the first graph class on which the classical complexity of these two problems differ.

Next, we revisit the problems on chordal graphs with bounded leafage and examine how far we can generalize this class. An *asteroidal triple* of a graph G is a set of three vertices such that each pair is connected by some path that avoids the closed neighborhood of the third vertex. Lekkerkerker and Boland [38] showed that a graph is an interval graph if and only if it is chordal and does not contain an asteroidal triple. They also listed all minimal chordal graphs that contain an asteroidal triple (see, for instance, [12, Figure 1]). Among this list, we found the *net graph* to be the most natural to generalize. For a positive integer $\ell \geq 3$, we define H_ℓ as a split graph on 2ℓ vertices with split partition (C, I) such that the only edges across C, I are a perfect matching. Note that H_3 is the net graph. As interval graphs are a proper subset of the collection of chordal graphs that do not contain a net graph as an induced subgraph, the collection of the chordal graph of leafage ℓ is a proper subset of the collection of chordal graphs that do not contain $H_{\ell+1}$ as an induced subgraph (see the full version [23]). We show that, although the considered domination problems are polynomial-time solvable for constant ℓ , the fixed-parameter tractability results are unlikely to extend to this larger class. Let us mention that the core reason these problems admit XP-algorithms parameterized by ℓ lies in the fact that H_ℓ -induced-subgraph-free chordal graphs have mim-width at most $\ell - 1$ [33] (all three problems are indeed known to be solvable in $n^{\mathcal{O}(m)}$ on graphs of mim-width at most m [6, 10]). Nonetheless, we present alternative algorithms which we believe to be simpler and more insightful. In fact, we give a $n^{\mathcal{O}(\ell)}$ algorithm for the more general RED-BLUE DOMINATING SET problem and obtain the other results by simple reductions.

► **Theorem 4.** DOMINATING SET, CONNECTED DOMINATING SET and STEINER TREE on H_ℓ -induced-subgraph-free chordal graphs are W[1]-hard when parameterized by ℓ and assuming the ETH, do not admit an algorithm running in time $f(\ell) \cdot n^{\mathcal{O}(\ell)}$ for any computable function f . However, they all admit XP-algorithms running in time $n^{\mathcal{O}(\ell)}$.

We observe a similar trend with respect to MULTICUT WITH UNDELETABLE TERMINALS as its parameterized complexity jumps from W[1]-hard on chordal graph of leafage ℓ to paraNP-hard on H_ℓ -induced-subgraph-free chordal graphs when parameterized by ℓ .

► **Theorem 5.** MULTICUT WITH UNDELETABLE TERMINALS is NP-hard even when restricted to H_3 -induced-subgraph-free chordal graphs.

Table 1 summarizes our results.

Our Methods. We briefly discuss the methods used in our two main algorithms, namely the algorithm for DOMINATING SET and the one for MULTIWAY CUT.

■ **Table 1** Overview of the known results and our contributions. Every graph class mentioned in the first column is a proper subset of the graph class mentioned below.

Input graph	DOM SET, CONNDOM SET, STEINER TREE	MULTICUT WITH UNDEL TERM	MULTIWAYCUT
Interval Graphs	Poly-time [13, 3]	Poly-time [28]	Poly-time [7]
Chordal graphs of leafage ℓ	$2^{\mathcal{O}(\ell^2)} \cdot n^{\mathcal{O}(1)}$ algo [20] $2^{\mathcal{O}(\ell)} \cdot n^{\mathcal{O}(1)}$ algo (Thm 1)	W[1]-hard $n^{\mathcal{O}(\ell)}$ algo (Thm 2)	$n^{\mathcal{O}(\ell)}$ algo [7] Poly-time (Thm 3)
H_ℓ -induced subgraph-free chordal	W[1]-hard (Thm 4); $n^{\mathcal{O}(\ell)}$ algo (Thm 4, [33] + [6, 10])	NP-hard for $\ell \geq 3$ (Thm 5)	Poly-time (Thm 3)
Chordal graphs	NP-complete [8]	NP-complete [48]	Poly-time (Thm 3)

Red-Blue Dominating Set in Chordal Graphs. As mentioned earlier, the linear ordering of cliques in interval graphs is particularly useful for the design of polynomial-time algorithms. Such an ordering is not possible even if G is a chordal graph whose representation tree T is a star. Consider the case where the model of every red vertex in G includes the center of the star T (and possibly some leaves) and the model of every blue vertex is (only) a leaf. We can solve this instance by converting it to an instance of SET COVER and solving it using the FPT algorithm parameterized by the size of the universe. In this case, the size of the universe is at most the number of leaves which is upper bounded by the leafage. In the other case where the properties of red vertices and blue vertices are reversed, we obtain a similar result by creating an equivalent instance of HITTING SET.

These ideas can be used in a more general setting as long as the following two properties are satisfied: (1) the model of each vertex is *local*, that is, it contains at most one branching node, and (2) each branching node is contained only in models of either red vertices or blue vertices. Based on this observation, we introduce a restricted version of the problem in which the input graph is required to satisfy these two conditions. We then show that the general case reduces to this restricted version: indeed, we prove that there is a branching algorithm that constructs $2^{\mathcal{O}(\ell)}$ many instances (where ℓ is the leafage of the input graph) of the restricted version of the problem such that the input instance is a YES-instance if and only if one of these newly created instances is a YES-instance. These two properties ensure that the graph induced by the red and blue vertices whose model intersect the subtree rooted at a farthest branching node (from some fixed root) satisfies the premise of at least one of the cases mentioned in the previous paragraph. We then present a greedy procedure, based on solving the SET COVER and HITTING SET problems, that identifies some part of an optimum solution. Apart from this greedy selection procedure, all other steps of the algorithm run in polynomial time.

Multiway Cut in Chordal Graphs. We give a polynomial-time algorithm for MULTIWAY CUT on chordal graphs by solving several instances of the (s, t) -CUT problem (not necessarily with unit capacities). Our strategy is based on a bottom-up dynamic programming (DP) on a tree representation of a chordal graph. An interesting aspect of our DP is that we need to look-up *all* DP table values that are already computed to compute a new entry. This is in contrast to typical DP-based algorithms that do computations only based on *local* entries.

We remark that we do not expect to design an algorithm for MULTIWAY CUT on chordal graphs using much simpler arguments (like a simple dynamic programming procedure etc.) as the problem generalizes some well-studied cut-flow based problems. As an example, recall the VERTEX COVER problem on bipartite graphs where given a bipartite graph G with bipartition (A, B) , the goal is to find $A' \subseteq A$ and $B' \subseteq B$ such that $|A' \cup B'|$ is minimum and $N(A \setminus A') \subseteq B'$. The set $A' \cup B'$ is called a vertex cover of G . The VERTEX COVER problem on bipartite graphs reduces to the MULTIWAY CUT problem on chordal graphs: indeed, let G' be the graph obtained from G by making B a clique, adding new pendant vertex t_a to each vertex $a \in A$, and further adding another new vertex t that is adjacent to all vertices of B . Then G' is a chordal graph and letting $T = t \cup \{t_a \mid a \in A\}$, it is easy to see that $S \subseteq V(G)$ is a vertex cover of G if and only if S is a T -multiway-cut in G' . As mentioned earlier, our algorithm solves several instances of the (s, t) -CUT problem, which also sits at the heart of some algorithms for VERTEX COVER on bipartite graphs. The above reduction suggests that an algorithm for MULTIWAY CUT on chordal graphs using much simpler techniques, would imply an algorithm for VERTEX COVER on bipartite graphs that uses much simpler techniques as well.

Note that a similar reduction would work from the weighted variant of the VERTEX COVER problem on bipartite graphs. This can be achieved by further replacing each vertex of the graph G by a clique of size proportional to the weight of this vertex and making each vertex of the clique adjacent to all the neighbors of this vertex. This reduction still preserves the chordality of the resulting graph.

2 Preliminaries

For a directed graph H , we denote, for all $v \in V(H)$, by $N_H^+(v)$ the out-neighbors of v and by $N_H^-(v)$ the in-neighbors of v . If H is clear from the context, we omit the subscript H . Given a (directed) path P and two vertices $u, v \in V(P)$, we denote by $P[u, v]$ the subpath of P from u to v . For a tree T rooted at r , we define the function $\text{parent}(t, T) : V(T) \setminus \{r\} \mapsto V(T)$ to specify the unique parent of the nodes in T . For any node $t \in T$, we denote by T_t the subtree rooted at t .

It is well-known that a chordal graph G can be represented as intersection graphs of subtrees in a tree T . The pair (T, \mathcal{M}) is called a *tree representation* of G where for every $v \in V(G)$, we denote by $\mathcal{M}(v)$ the subtree corresponding to v and refer to $\mathcal{M}(v)$ as the *model* of v in T . The *leafage* of G , denoted by $\text{lf}(G)$, is defined as the minimum number of leaves in the tree of a tree representation of G .

For every node $\alpha \in V(T)$, we let $\text{ver}(\alpha) = \{v \in V(G) \mid \alpha \in \mathcal{M}(v)\}$ be the set of vertices in G that contain the node α as their model. A vertex $v \in V(G)$ whose model contains α may also be referred to as an α -*vertex*. Similarly, for every edge $e \in E(T)$, we define $\text{ver}(e) = \{v \in V(G) \mid e \subseteq \mathcal{M}(v)\}$. Given a subtree T' of T , we denote by $G_{|T'}$ the subgraph of G induced by those vertices $x \in V(G)$ such that $V(\mathcal{M}(x)) \subseteq V(T')$.

3 Dominating Set

For a graph G , a set $X \subseteq V(G)$ is a *dominating set* if every vertex in $V(G) \setminus X$ has at least one neighbor in X , that is, $V(G) = N[X]$. In the DOMINATING SET problem (DOMSET for short), the input is a graph G and an integer k , and the objective is to decide whether G has a dominating set of size at most k . We assume that the leafage of the input graph is given as part of the input. If not, recall that it can be computed in polynomial time [29]. We consider a generalized version of this problem as defined below.

RED-BLUE DOMINATING SET (RED-BLUE-DOMSET)

Input: A graph G , a partition (R, B) of $V(G)$, and an integer k .

Question: Does there exist a set $X \subseteq R$ of size at most k such that $B \subseteq N(X)$?

To solve DOMSET, it is sufficient to solve RED-BLUE-DOMSET even when the input is restricted to chordal graphs of leafage ℓ . A simple reduction, which is given in the full version [23], suffices to prove the following result.

► **Lemma 6.** *There is a polynomial-time algorithm that given an instance (G, k) of DOMSET constructs an equivalent instance $(G', (R', B'), k)$ of RED-BLUE-DOMSET such that if G has leafage at most ℓ , then so does G' .*

In the remainder of this section, we present an FPT algorithm for RED-BLUE-DOMSET when parameterized by the leafage ℓ of the input graph. The algorithm consists of two parts. In the first part, the algorithm constructs $2^{\mathcal{O}(\ell)}$ many instances of a “restricted version” of the problem such that the input instance is a YES-instance if and only if one of these newly created instances is a YES-instance. Moreover, the graphs in the newly created instances satisfy certain properties that allow us to design a fast algorithm. See Lemma 7 for the formal statement. In the second part (cf. Lemma 8), the algorithm solves the restricted version of RED-BLUE-DOMSET which is defined as follows.

RESTRICTED-RED-BLUE DOMINATING SET (REST-RED-BLUE-DOMSET)

Input: A chordal graph G , a partition (R, B) of $V(G)$, an integer k and tree representation (T, \mathcal{M}) of G such that

- for every vertex in G , its model contains at most one branching node of T , and
- for all branching nodes $\gamma \in V(T)$, there are either only red γ -vertices or only blue γ -vertices.

Question: Does there exist a set $D \subseteq R$ of size at most k such that $B \subseteq N(D)$?

The first step of the algorithm is summarized in the following lemma which is proven in Appendix A.

► **Lemma 7.** *Let $\mathcal{I} = (G, (R, B), k)$ be an instance of RED-BLUE-DOMSET where G is a chordal graph of leafage at most ℓ . We can construct, in time $2^{\mathcal{O}(\ell)} \cdot n^{\mathcal{O}(1)}$, a collection $\{\mathcal{I}_i = (G_i, (R_i, B_i), k) \mid i \in [2^{\mathcal{O}(\ell)}]\}$ of REST-RED-BLUE-DOMSET instances such that*

- for every $i \in [2^{\mathcal{O}(\ell)}]$, G_i is a chordal graph of leafage at most 2ℓ , and
- \mathcal{I} is a YES-instance of RED-BLUE-DOMSET if and only if at least one of the instances in the collection is a YES-instance of REST-RED-BLUE-DOMSET.

The second step of the algorithm solves REST-RED-BLUE-DOMSET. Formally, we prove the following lemma.

► **Lemma 8.** REST-RED-BLUE-DOMSET admits an algorithm running in time $2^{\mathcal{O}(\ell)} \cdot n^{\mathcal{O}(1)}$.

We first state some easy reduction rules before we handle two cases based on whether the farthest branching node⁴ is contained only in the models of red vertices or blue vertices. We present Greedy Select 14 and Greedy Select 16 to handle these cases. The proof of the lemma follows from the correctness of the Greedy Select 14 and 16 and the fact that each application of the greedy selection procedure deletes some vertices in the graph.

⁴ We assume that the tree in the tree representation is rooted and thus, by farthest branching node, we mean farthest from the root.

We first introduce some notations. Recall that an instance of REST-RED-BLUE-DOMSET contains a chordal graph G , a partition (R, B) of $V(G)$, an integer k and tree representation (T, \mathcal{M}) of G such that for every vertex in G , its model contains at most one branching node of T , and for all branching nodes $\gamma \in V(T)$, there are either only red γ -vertices or only blue γ -vertices. We assume, without loss of generality, that the tree T is rooted at node r . Unless mentioned otherwise, α denotes the farthest branching node in T from the root, that is, each proper subtree of T_α is a path. If there are more than one branching node that satisfy the property, we arbitrarily select one of them. Let β be the closest branching ancestor of α , that is, no internal node in the unique path from α to β is a branching node in T .⁵ Recall that for a vertex $v \in V(G)$, we define $\text{top}_{\mathcal{M}}(v)$ as the node $\eta \in \mathcal{M}(v)$ that is closest to the root. Likewise if a leaf λ is fixed, we define $\text{bot}_{\mathcal{M}}^\lambda(v)$ as the node $\eta \in \mathcal{M}(v)$ that is closest to λ . For ease of notation, we omit λ as it is always clear from the context.

- **Definition 9.** Let γ be a node of the tree T . We define the following sets of vertices in G .
- $B_\gamma^\cap, R_\gamma^\cap, V_\gamma^\cap$ are the sets of, respectively, blue, red, all vertices $v \in V(G)$ whose models intersect the tree rooted at γ , i.e., $\mathcal{M}(v) \cap V(T_\gamma) \neq \emptyset$.
 - $B_\gamma^\subseteq, R_\gamma^\subseteq, V_\gamma^\subseteq$ are the sets of, respectively, blue, red, all vertices $v \in V(G)$ whose models are completely contained inside the tree rooted at γ , i.e., $\mathcal{M}(v) \subseteq V(T_\gamma)$.
 - $B_\gamma^{\subseteq\ddagger}, R_\gamma^{\subseteq\ddagger}, V_\gamma^{\subseteq\ddagger}$ are the sets of blue, red, all vertices $v \in V(G)$ where the model is completely contained inside the tree rooted at γ but does not contain γ , respectively, i.e. $\mathcal{M}(v) \subseteq V(T_\gamma^\ddagger) = V(T_\gamma) \setminus \{\gamma\}$.
 - $B_\gamma^\in, R_\gamma^\in, V_\gamma^\in$ are the sets of, respectively, blue, red, all vertices $v \in V(G)$ whose models contains γ , i.e., $\gamma \in \mathcal{M}(v)$.

Simplifications. We first apply the following easy reduction rules whose correctness readily follows from the definition of the problem. It is also easy to see that the reduction rules can be applied in polynomial time and the reduced instance is also a valid instance of REST-RED-BLUE-DOMSET.

► **Reduction Rule 10.** If there is a blue vertex, which is not adjacent to a red vertex, or if $k < 0$, then return a trivial NO-instance.

► **Reduction Rule 11.**

- If there are two blue vertices u, v such that $\mathcal{M}(u) \subseteq \mathcal{M}(v)$, then delete v .
- If there are two red vertices u, v such that $\mathcal{M}(u) \subseteq \mathcal{M}(v)$, then delete u .

Consider a blue vertex v in G whose model is contained in the subtree rooted at α . Moreover, let v be such a vertex for which $\text{top}_{\mathcal{M}}(v)$ is farthest from the root and v is not adjacent to a red vertex whose model contains α . Hence, there is a natural ordering amongst the red neighbors of v . Note that such an ordering is not possible if some of its neighbors contain α in their models. As any solution contains a red neighbor of v , it is safe to include its neighbor v_r for which $\text{top}_{\mathcal{M}}(v_r)$ is closest to α .

► **Reduction Rule 12.** Suppose that there is a blue vertex $v \in B_\alpha^{\subseteq\ddagger}$ such that $\text{top}_{\mathcal{M}}(v)$ is farthest from the root and v is not adjacent to any red α -vertices. Moreover, amongst all the red neighbors of v , let v_r be the node such that $\text{top}_{\mathcal{M}}(v_r)$ is closest to α . Then, remove v_r and all of its blue neighbors and decrease k by 1.

⁵ If α is the root of the tree, then we can add an artificial new root β which is not contained in the model of any vertex.

We remark that the above reduction rule is applicable irrespective of the fact whether either all α -vertices are red or all α -vertices are blue.

Case-1: All the vertices that contain α in their models are red vertices. Let β be the closest branching ancestor of α . Consider the blue vertices whose model intersect the path from α to β . Note that there may not be any such blue vertex; however, we find it convenient to present an uniform argument. With a slight abuse of notation, let b_1, \dots, b_d be these blue vertices ordered according to their endpoint in the direction of α , that is, for $i < j$ we have either $\text{bot}_{\mathcal{M}}(b_i) = \text{bot}_{\mathcal{M}}(b_j)$ or $\text{bot}_{\mathcal{M}}(b_i)$ is closer to α than $\text{bot}_{\mathcal{M}}(b_j)$. For each $i \in [d]$, we compute an optimal solution for dominating the vertices whose model is in the tree rooted at α (i.e., the vertices of $B_\alpha^{\subseteq \dagger}$) and the vertex b_i while only using red α -vertices. Formally, we want to compute an optimal solution for the following instance: $\mathcal{I}_i := G[R_\alpha^\cap \cup B_\alpha^{\subseteq} \cup \{b_i\}]$. We also define instance $\mathcal{I}_0 := G[R_\alpha^\cap \cup B_\alpha^{\subseteq}]$ to handle the cases when there are no blue vertices whose model intersects the path from α to β or when b_1 (and hence, the other blue vertices mentioned above) are not dominated by red α -vertices in an optimum solution. To simplify notation we set $\text{OPT}_i := \text{OPT}(\mathcal{I}_i)$ in the following. If \mathcal{I}_i is not defined, then we set $\text{OPT}_i = \infty$. Note that the solution OPT_i also dominates the blue vertices b_1, \dots, b_{i-1} due to the ordering of the b_i s. Hence, for any $i, j \in [0, d]$ such that $i < j$, we have $|\text{OPT}_i| \leq |\text{OPT}_j|$. We use this monotonicity to prove the following structural lemma.

► **Lemma 13.** *Let $q \in [0, d]$ be the largest value such that $|\text{OPT}_q| = |\text{OPT}_0|$. If there is a solution, then there is an optimum solution containing OPT_q .*

Proof. Let OPT be an optimum solution of $(G, (R, B), k)$. Let S denote the collection of vertices in OPT whose model contains nodes in the subtree rooted at α , i.e., $S := \text{OPT} \cap R_\alpha^\cap$. We claim that we can replace S by a super-set S' of OPT_q of equal size to obtain another solution.

Let $j \in [0, d]$ be the largest integer such that b_j is dominated by some vertex in S . If $j \leq q$, then by our choice of q , $|S| = |\text{OPT}_q|$. By the definition of the \mathcal{I}_i s, we get that OPT_q is also a solution for \mathcal{I}_i . Hence, we can replace S by OPT_q to get another optimal solution. Suppose therefore that $j > q$. By our choice of q , we have $|S| > |\text{OPT}_q|$. Let r_j be the red α -vertex with $\text{top}_{\mathcal{M}}(r_j)$ closest to β such that b_j is a neighbor of r_j . Such a vertex exists, as by assumption, S contains one of these vertices which dominates b_j . Then we replace S by $S' = \text{OPT}_q \cup \{r_j\}$. As $|S| > |\text{OPT}_q|$, we have $|S'| \leq |S|$. Moreover, observe that $S' \cup \text{OPT} \setminus S$ is still a solution as all vertices in $B_\alpha^{\subseteq \dagger}$ and the vertices b_1, \dots, b_q are dominated by some vertex in OPT_q , vertex r_j dominates the vertices b_{q+1}, \dots, b_j and, by the choice of j , the vertices b_{j+1}, \dots, b_d are dominated by some vertex not contained in S . ◀

We devise a greedy selection step based on the above lemma which can be completed in time $2^{\mathcal{O}(\ell)} \cdot n^{\mathcal{O}(1)}$ (cf. full version in [23]).

► **Greedy Select 14.** *Let $q \in [0, d]$ be the largest value such that $|\text{OPT}_q| = |\text{OPT}_0|$. Include the vertices of OPT_q in the solution, i.e., delete the red vertices in OPT_q , the blue vertices that are adjacent to vertices in OPT_q , and decrease k by $|\text{OPT}_q|$.*

Case-2: All the vertices that contain α in their models are blue vertices. Let β be the closest branching ancestor of α . We consider two cases depending on whether there is a red vertex whose model intersects the path from α to β . If there is no such red vertex, then we

14:10 Domination and Cut Problems on Chordal Graphs with Bounded Leafage

consider the graph induced by all the red vertices whose model is (properly) contained in the subtree rooted at α and the blue vertices whose model intersects the subtree rooted at α . Formally, we define $\mathcal{I}_0 = G[R_\alpha^\subseteq \cup B_\alpha^\cap]$.

Consider the other case and suppose that there are $d \geq 1$ many red vertices whose model intersects the path from α to β . Let r_1, \dots, r_d be these vertices ordered according to their endpoints in the direction of α , that is, for $i < j$, we have either $\text{bot}_{\mathcal{M}}(r_i) = \text{bot}_{\mathcal{M}}(r_j)$ or $\text{bot}_{\mathcal{M}}(r_i)$ is closer to α than $\text{bot}_{\mathcal{M}}(r_j)$. For each such red vertex v_i , we compute the optimal solution to dominate the vertices in B_α^\cap by vertices in R_α^\subseteq assuming that v_i is already selected. Note that we only have to focus on the blue vertices in B_α^\cap which are not adjacent to v_i . Formally we define $\mathcal{I}_i = G[R_\alpha^\subseteq \cup (B_\alpha^\cap \setminus N[v_i])]$. It is possible that the optimum solution does not include any of the vertices in $\{r_1, r_2, \dots, r_d\}$. To handle this case, we define $\mathcal{I}_{d+1} = G[R_\alpha^\subseteq \cup B_\alpha^\cap]$. To simplify notation, we set $\text{OPT}_i := \text{OPT}(\mathcal{I}_i)$ in the following. Note that for the instance defined above, R_i is same for every instance whereas $B_i \subseteq B_{i+1}$ because of the ordering. Hence, for any $i, j \in [d+1]$ such that $i < j$, we have $|\text{OPT}_i| \leq |\text{OPT}_j|$. We use this monotonicity to prove the following structural lemma.

► **Lemma 15.** *If there is a red vertex whose model intersects the path from α to β , let $q \in [d+1]$ be the largest value such that $|\text{OPT}_q| = |\text{OPT}_1|$. Otherwise, define $\text{OPT}_q = \text{OPT}_0$. If there is a solution for the instance, then there is an optimum solution OPT such that $\text{OPT} \cap R_\alpha^\subseteq = \text{OPT}_q$.*

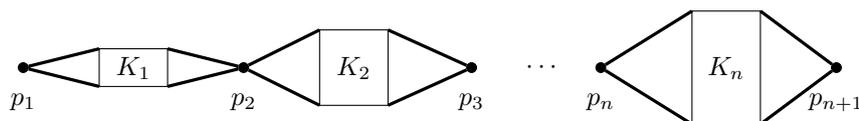
Proof. If there is no red vertices whose model intersects the path from α to β , then all the red vertices in G that are adjacent to blue vertices in \mathcal{I}_0 are the red vertices in \mathcal{I}_0 . Hence, the statement of the lemma follows.

We now consider the case where there are red vertices whose model intersects the path from α to β . Let OPT be an optimum solution of $(G, (R, B), k)$. Let S denote the collection of vertices in OPT whose model is (properly) contained in the subtree rooted at α , i.e., $S := \text{OPT} \cap R_\alpha^\subseteq$. We claim that we can replace S by a super-set S' of OPT_q of equal size to obtain another optimum solution.

Let $j \in [d]$ be the smallest index such that v_j is contained in OPT . Note that, by definition, $j \neq d+1$ as there are only d red vertices with the said property. If $j \leq q$, then by our choice of q , $|S| \geq |\text{OPT}_j|$. By the definition of \mathcal{I}_j and the fact blue vertices in \mathcal{I}_j are subset of blue vertices in \mathcal{I}_q , OPT_q is also a solution for \mathcal{I}_j . Hence, we can replace S by OPT_q to get another optimal solution. Suppose therefore that $j > q$. By our choice of q , we have $|\text{OPT}_j| > |\text{OPT}_q|$. As OPT is a solution, all vertices in B_α^\cap must be covered by OPT . Hence, we can replace S by $S' = \text{OPT}_q \cup \{r_q\}$ and get a solution of not larger size which still dominates all vertices in B_α^\cap . Indeed, the vertices which are not dominated by OPT_q are dominated by r_q . ◀

We devise a greedy selection step based on the above lemma which can be completed in time $2^{\mathcal{O}(\ell)} \cdot n^{\mathcal{O}(1)}$ (cf. full version in [23]).

► **Greedy Select 16.** *If there is a red vertex whose model intersects the path from α to β , let $q \in [d+1]$ be the largest value such that $|\text{OPT}_q| = |\text{OPT}_1|$. Otherwise, define $\text{OPT}_q = \text{OPT}_0$. Include OPT_q in the solution, i.e., delete the red vertices in OPT_q , the blue vertices that are adjacent to vertices in OPT_q , and decrease k by $|\text{OPT}_q|$.*



■ **Figure 1** The auxiliary graph B . Rectangles represent cliques and thick edges indicate that the corresponding vertex is complete to the corresponding cliques.

4 Multicut with Undeletable Terminals

This section considers the MULTICUT WITH UNDELETABLE TERMINALS problem formally defined as follows.

MULTICUT WITH UNDELETABLE TERMINALS (MULTICUT WITH UNDEL TERM)

Input: An undirected graph G , a set $P \subseteq V(G) \times V(G)$, and an integer k .

Question: Is there a set $S \subseteq V(G) \setminus V(P)$ such that $|S| \leq k$ and for all $(p, p') \in P$, there is no path between p and p' in $G - S$?

In the following, a set $S \subseteq V(G) \setminus V(P)$ such that for all $(p, p') \in P$, there is no path between p and p' in $G - S$ is called a P -multicut in G . We first prove that when the input is restricted to chordal graphs, the problem is unlikely to admit an FPT algorithm when parameterized by the leafage. We then complement this result with an XP-algorithm parameterized by the leafage. We restate the theorem with the precise statement for the reader's convenience.

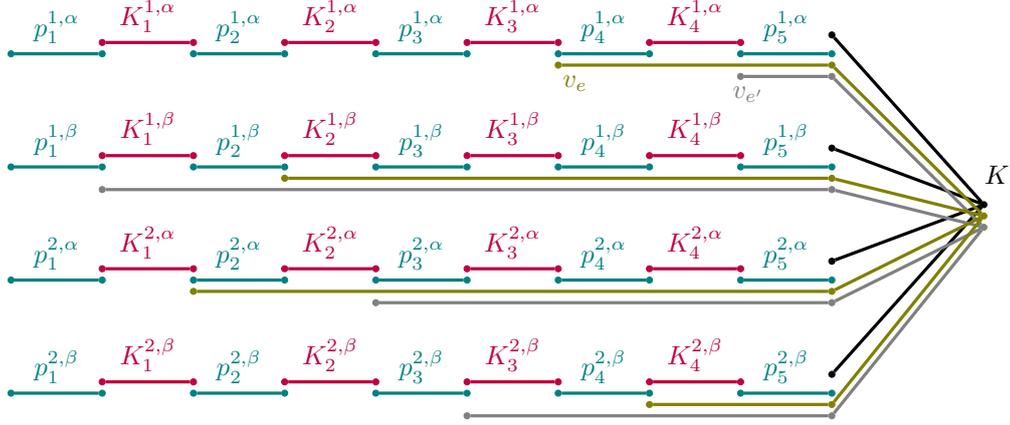
► **Theorem 2.** MULTICUT WITH UNDELETABLE TERMINALS on chordal graphs is $W[1]$ -hard when parameterized by the leafage ℓ and assuming the ETH, does not admit an algorithm running in time $f(\ell) \cdot n^{o(\ell)}$ for any computable function f . However, it admits an XP-algorithm running in time $n^{\mathcal{O}(\ell)}$.

To prove that the problem is $W[1]$ -hard, we present a parameter preserving reduction from MULTICOLORED CLIQUE. An instance of this problem consists of a simple graph G , an integer q , and a partition (V_1, V_2, \dots, V_q) of $V(G)$. The objective is to determine whether there is a clique in G that contains exactly one vertex from each part V_i . Such a clique is called a *multicolored clique*. We assume, without loss of generality, that each V_i is an independent set and that $|V_1| = \dots = |V_q| = n$.⁶ This implies, in particular, that $|E(G)| < n^2 \cdot q^2$. For every $i \in [q]$, we denote by v_1^i, \dots, v_n^i the vertex set of V_i and for every $i \neq j \in [q]$, we denote by $E_{i,j} \subseteq E(G)$ the set of edges between V_i and V_j . We define $M := (n+1)^2 \cdot q^2$.

Reduction. The reduction takes as input an instance $(G, q, (V_1, \dots, V_q))$ of MULTICOLORED CLIQUE and outputs an instance (H, P, k) of MULTICUT WITH UNDEL TERM which is constructed as follows.

- The reduction starts by constructing an auxiliary graph B . The vertex set of B consists of $n+1$ vertices p_1, \dots, p_{n+1} and n vertex-disjoint cliques K_1, \dots, K_n such that $|K_a| = a \cdot M$ for every $a \in [n]$. Then, it adds edges so that p_1 is complete to K_1 , p_{n+1} complete to K_n , and p_a complete to $K_{a-1} \cup K_a$ for every $a \in [n] \setminus \{1\}$. This completes the construction of B (see Figure 1).

⁶ Unlike in the rest of the article, we *do not* use n to denote the total number of vertices in G to keep notation simple while presenting the reduction.



■ **Figure 2** A tree representation of the graph H restricted to the gadgets representing V_1, V_2 and $E_{1,2}$ where $n = 4$ and $E_{1,2} = \{e = v_3^1 v_1^2, e' = v_4^1 v_2^2\}$.

- For each $i \in [q]$, the reduction introduces two vertex-disjoint copies $B^{i,\alpha}$ and $B^{i,\beta}$ of B . For every $i \in [q]$, let $p_1^{i,\alpha}, \dots, p_{n+1}^{i,\alpha}$ denote the copies of p_1, \dots, p_{n+1} in $B^{i,\alpha}$ and $K_1^{i,\alpha}, \dots, K_n^{i,\alpha}$ denote the copies of K_1, \dots, K_n in $B^{i,\alpha}$. Moreover, for every $1 \leq a_1 \leq a_2 \leq n+1$, we define, for notational convenience,

$$p^{i,\alpha}[a_1, a_2] := \{p_a^{i,\alpha} \mid a_1 \leq a \leq a_2\} \quad \text{and} \quad K^{i,\alpha}[a_1, a_2] := \bigcup_{a_1 \leq a \leq a_2} K_a^{i,\alpha}.$$

We define $p_a^{i,\beta}, K_a^{i,\beta}, p^{i,\beta}[a_1, a_2]$, and $K^{i,\beta}[a_1, a_2]$ in a similar way.

- For $i \in [q]$ and $a \in [n]$, the reduction uses $p_a^{i,\alpha}, p_{n+1-a}^{i,\beta}, K_a^{i,\alpha}$, and $K_{n+1-a}^{i,\beta}$ to encode vertex v_a^i .
- For every edge $e = v_{a_i}^i v_{a_j}^j \in E(G)$, the reduction introduces an *edge-vertex* v_e and adds edges so that v_e is complete to the following sets.
 - $p^{i,\alpha}[a_i + 1, n + 1]$ and $K^{i,\alpha}[a_i, n + 1]$ in $V(B^{i,\alpha})$.
 - $p^{j,\alpha}[a_j + 1, n + 1]$ and $K^{j,\alpha}[a_j, n + 1]$ in $V(B^{j,\alpha})$.
 - $p^{i,\beta}[n + 1 - a_i + 1, n + 1]$ and $K^{i,\beta}[n + 1 - a_i + 1, n + 1]$ in $V(B^{i,\beta})$.
 - $p^{j,\beta}[n + 1 - a_j + 1, n + 1]$ and $K^{j,\beta}[n + 1 - a_j + 1, n + 1]$ in $V(B^{j,\beta})$.
 Note that v_e is adjacent to vertices in $K^{i,\alpha}[a_i] \cup K^{j,\alpha}[a_j]$ but not to any vertex in $K^{i,\beta}[n + 1 - a_i] \cup K^{j,\beta}[n + 1 - a_j]$.
- The reduction introduces a central clique K of size $2M^2$ and makes it complete to $\{p_{n+1}^{i,\alpha}, p_{n+1}^{i,\beta} \mid i \in [q]\}$ and V_E where $V_E = \{v_e \mid e \in E(G)\}$ is the set of edge-vertices. This completes the construction of H .
- The reduction further defines

$$P := \{(p_a^{i,\alpha}, p_{n+2-a}^{i,\beta}) \mid a \in [n] \text{ and } i \in [q]\}, \quad \text{and} \quad k := q(n+1)M + |E(G)| - q(q-1)/2.$$

The reduction returns (H, P, k) as the instance of MULTICUT WITH UNDEL TERM. This completes the reduction. It is easy to see that H is chordal and has leafage at most $2q$. See Figure 2 for a tree representation of H .

Intuition. We first provide the intuition behind the reduction. Recall that the reduction uses $p_a^{i,\alpha}, p_{n+1-a}^{i,\beta}, K_a^{i,\alpha}$, and $K_{n+1-a}^{i,\beta}$ to encode vertex v_a^i where $i \in [q]$ and $a \in [n]$. Hence, for $a, b \in [n]$, if $a + b = n + 1$, then $p_a^{i,\alpha}$ and $p_b^{i,\beta}$ correspond to the same vertex. Note that the

pairs in P do not correspond to the vertices associated with v_a^i . Rather, $p_{a+1}^{i,\alpha}$ is paired with $p_{n+1-a}^{i,\beta}$. Conversely, for $a, b \in [n]$, if $a + b = n + 2$, then $(p_a^{i,\alpha}, p_b^{i,\beta}) \in P$. By the construction of H and P , for a P -multicut S of H , if there is a path from $p_a^{i,\alpha}$ to $p_b^{i,\beta}$ in $H - S$, then $a + b \geq n + 3$.

Now, consider the terminal pairs $(p_1^{i,\alpha}, p_{n+1}^{i,\beta})$ in P for some $i \in [q]$. Because of the size constraints, S cannot contain all the vertices of the central clique K . Since S cannot contain a terminal, it needs to include one clique from $B^{i,\alpha}$. Let $a_i \in [n]$ be the largest index such that $K_{a_i}^{i,\alpha} \subseteq S$. Using similar arguments, there must also exist $b_i \in [n]$ such that $K_{b_i}^{i,\beta} \subseteq S$ and b_i is largest such index. By definition of a_i, b_i and construction of H , there is a path from $p_{a_i+1}^{i,\alpha}$ to $p_{b_i+1}^{i,\beta}$ in $H - S$. The discussion in the previous paragraph implies that $a_i + 1 + b_i + 1 \geq n + 3$, i.e., $a_i + b_i \geq n + 1$. However, by definition of the solution size k and the size of the cliques, we have $a_i + b_i \leq n + 1$. Hence, the structure of the auxiliary graphs and the terminal pairs ensure that the selected cliques in $S \cap V(B^{i,\alpha})$ and $S \cap V(B^{i,\beta})$ encode selecting a vertex in V_i in G .

Suppose that $\{v_{a_1}^1, v_{a_2}^2, \dots, v_{a_q}^q\}$ are the vertices in G that are selected by S . Recall that V_E is the collection of edge-vertices in H . Considering the remaining budget, a solution S can include at most $|E(G)| - q(q-1)/2$ many vertices in V_E . We argue that $q(q-1)/2$ edges in G corresponding to vertices in $V_E \setminus S$ should have their endpoints in $\{v_{a_1}^1, v_{a_2}^2, \dots, v_{a_q}^q\}$ as otherwise some terminal pair is connected in $H - S$. Hence, a P -multicut S of H corresponds to a multicolored clique in G . We give the formal proof in the full version [23].

Finally, it is known that, assuming the ETH, there is no algorithm that can solve MULTICOLORED CLIQUE on instance $(G, q, (V_1, V_2, \dots, V_q))$ in time $f(q) \cdot |V(G)|^{o(q)}$ for any computable function f (see, e.g., [17, Corollary 14.23]). Thus, together with the fact that the reduction takes polynomial time in the size of the input, the proof of correctness, and arguments that are standard for parameter preserving reductions, we conclude that the following holds.

► **Lemma 17.** *MULTICUT WITH UNDELETABLE TERMINALS on chordal graphs is $W[1]$ -hard when parameterized by leafage ℓ and assuming the ETH, does not admit an algorithm running in time $f(\ell) \cdot n^{o(\ell)}$ for any computable function f .*

We defer the XP-algorithm for MULTICUT WITH UNDEL TERM on chordal graphs to the full version of this paper [23]. Together with Lemma 17 this proves Theorem 2.

5 Multiway Cut with Undeletable Terminals on Chordal Graphs

In this section, we consider the MULTIWAY CUT WITH UNDELETABLE TERMINALS problem formally defined below. Given a graph G and a set $P \subseteq V(G)$, a set $S \subseteq V(G) \setminus P$ is called a P -multiway-cut in G if $G - S$ has no (p, p') -path for any two distinct $p, p' \in P$.

MULTIWAY CUT WITH UNDELETABLE TERMINALS (MWC)

Input: An undirected graph G and a set $P \subseteq V(G)$ of terminals.

Question: Find the size of a minimum P -multiway-cut in G .

The aim of this section is to prove Theorem 3 which states that MULTIWAY CUT WITH UNDELETABLE TERMINALS can be solved in $n^{O(1)}$ -time on chordal graphs. Before turning to the proof, we first start with a few definitions. Let (T, \mathcal{M}) a tree representation of a chordal graph G where T is rooted at an arbitrary node $r \in V(T)$. Given a subtree T' of T and a set $Q \subseteq V(G)$, we let $Q|_{T'} \subseteq Q$ be the set of vertices $x \in Q$ such that $\mathcal{M}(x) \subseteq V(T')$. Now let

$Q \subseteq V(G)$ be an independent set of G such that for every leaf η of T , $\mathbf{ver}(\eta) \cap Q \neq \emptyset$. Then the *truncated tree w.r.t. Q* is the tree T_Q^{trunc} obtained from T as follows. Let $\{\eta_1, \dots, \eta_q\}$ be the set of leaves of T . For each $i \in [q]$, let $Q_i \subseteq Q \setminus \mathbf{ver}(r)$ be the set of vertices $p \in Q \setminus \mathbf{ver}(r)$ such that $\mathbf{top}_{\mathcal{M}}(p)$ is on the (η_i, r) -path in T , and let $p_i \in Q_i$ be the vertex of Q_i such that $\mathbf{top}_{\mathcal{M}}(p_i)$ is closest to r . Then T_Q^{trunc} is obtained from T by deleting the subtrees rooted at the children of the nodes in $\{\mathbf{top}_{\mathcal{M}}(p_i) \mid i \in [q]\}$. Note that, by construction, the set of leaves of T_Q^{trunc} is $\{\mathbf{top}_{\mathcal{M}}(p_i) \mid i \in [q]\}$ and that, apart from the vertices in $\{p_i \mid i \in [q]\}$, there is at most one other vertex in Q whose model intersects $V(T_Q^{\text{trunc}})$, namely the potential vertex in $Q \cap \mathbf{ver}(r)$ (note that if such a vertex exists, its model is in fact fully contained in T_Q^{trunc}). Finally, given a set $P \subseteq V(G)$, a P -multiway-cut X in G is said to *destroy* an edge $e \in E(T)$ if $\mathbf{ver}(e) \subseteq X$.

We now turn to the proof of Theorem 3. Throughout the remaining of this section, we let (G, P) be an instance of MWC, where G is a n -vertex chordal graph, and further let (T, \mathcal{M}) be a tree representation of G . First, we may assume that P is an independent set: indeed, if there exist $p, p' \in P$ such that $pp' \in E(G)$, then (G, P) is a NO-instance. Furthermore, if a vertex $v \in V(G)$ does not belong to any (p, p') -path in G , where $p, p' \in P$, then it can be safely deleted as no minimal P -multiway-cut in G may contain v . Hence, we assume that every vertex in G participates in some (p, p') -path where $p, p' \in P$; in particular, we may assume that for every leaf η of T , $\mathbf{ver}(\eta) \cap P \neq \emptyset$. Note that, consequently, for every internal node $\alpha \in V(T)$, the truncation of T_α w.r.t. P_{T_α} exists.

Now let T_0 be the tree obtained by adding a new node r_0 and connecting it to an arbitrary node $r \in V(T)$. Observe that (T_0, \mathcal{M}) is also a tree representation of G . In the following, we root T_0 at r_0 . To prove Theorem 3, we design a dynamic program that computes, in a bottom-up traversal of T_0 , the entries of a table \mathbf{A} whose content is defined as follows. The table \mathbf{A} is indexed over the edges of $E(T_0)$. For each node $\alpha \in V(T)$, $\mathbf{A}[\alpha \text{ parent}_{T_0}(\alpha)]$ stores the size of a minimum P_{T_α} -multiway-cut in G_{T_α} . The size of a minimum P -multiway-cut in G may then be found in $\mathbf{A}[\mathbf{rr}_0]$. We describe below how to compute the entries of \mathbf{A} .

Update Procedure. For every leaf η of T , we set $\mathbf{A}[\eta \text{ parent}_{T_0}(\eta)] = 0$. Consider now an internal node α of T . We show how to compute $\mathbf{A}[\alpha \text{ parent}_{T_0}(\alpha)]$ assuming that for every edge $e \in E(T_\alpha)$, the entry $\mathbf{A}[e]$ is correctly filled.

Let \tilde{T} be the truncation of T_α w.r.t. P_{T_α} and let $\tilde{G} = G_{\tilde{T}}$. Denote by η_1, \dots, η_q the leaves of \tilde{T} . Recall that, by construction, for every $i \in [q]$, there exists $p_i \in P_{T_\alpha}$ such that $\eta_i = \mathbf{top}_{\mathcal{M}}(p_i)$: we let $\tilde{P} = \{p_i \mid i \in [q]\}$. Furthermore, it may be that $P_{T_\alpha} \cap \mathbf{ver}(r)$ is nonempty: we let $\tilde{P}_r = P_{T_\alpha} \cap \mathbf{ver}(r)$. Note that $|\tilde{P}_r| \leq 1$: if $\tilde{P}_r \neq \emptyset$ then we refer to the terminal in \tilde{P}_r as the *root terminal*. Observe that $V(\tilde{G}) \cap P_{T_\alpha} = V(\tilde{G}) \cap P = \tilde{P} \cup \tilde{P}_r$ by construction. To compute $\mathbf{A}[\alpha \text{ parent}_{T_0}(\alpha)]$, we distinguish two cases:

- (1) if $\tilde{P}_r \neq \emptyset$ then we construct a unique instance $(H_0, \mathbf{s}, \mathbf{t}, \mathbf{wt}_0)$ of (s, t) -CUT;
- (2) otherwise, for every $i \in [0, q]$, we construct an instance $(H_i, \mathbf{s}, \mathbf{t}, \mathbf{wt}_i)$ of (s, t) -CUT.

We describe below how such instances are constructed. First, recall that an instance of the (s, t) -CUT problem consists of a digraph D , vertices $s, t \in V(D)$, a weight function $\mathbf{wt} : E(D) \rightarrow \mathbb{N} \cup \{\infty\}$, and the goal is to find a set $X \subseteq E(D)$ such that $D - X$ has no (s, t) -path and $\mathbf{wt}(X)$ is minimum with this property, where $\mathbf{wt}(X) = \sum_{u \in X} \mathbf{wt}(u)$.

Construction of the (s, t) -Cut Instances. For every $i \in [q]$, let us denote by $\tilde{P}_i = \tilde{P} \setminus \{p_i\}$ and let $\tilde{P}_0 = \tilde{P}$. Consider $i \in [0, q]$. Before turning to the formal construction of the instance $(H_i, \mathbf{s}, \mathbf{t}, \mathbf{wt}_i)$, let us first give an intuitive idea of the construction. The digraph H_i

is obtained from \tilde{T} by orienting all edges of \tilde{T} towards its root $\tilde{r} = \alpha$ and further adding vertices and weighted arcs to encode the graph $G|_{T_\alpha}$. The arcs in H_i corresponding to the edges of \tilde{T} are called the *tree arcs* and the nodes in H_i corresponding to the nodes of \tilde{T} are called the *tree nodes*. The idea is that we separate, for each terminal $p \in \tilde{P}_i$, the node $\text{top}_{\mathcal{M}}(p)$ from the root \tilde{r} . To achieve this, we add a *source* node \mathbf{s} and *source* arcs from \mathbf{s} to $\text{top}_{\mathcal{M}}(p)$ (of infinite weight) and look for an (\mathbf{s}, \tilde{r}) -cut in H_i . Since the edges of T can presumably not be independently destroyed in a P -multiway-cut, we need some additional vertices to encode these dependencies. For each vertex $v \in V(\tilde{G}) \setminus \tilde{P}_i$, we introduce a node $\gamma(v)$ in H_i which is reachable via *connection* arcs (with infinite weight) from all the tree nodes that are contained in the model of v . This node $\gamma(v)$ is further connected via a *sink* arc (of weight one) to $\text{top}_{\mathcal{M}}(v)$ which ensures that if we want to cut a tree arc, we also have to cut all the sink arcs associated to vertices containing the corresponding edge in their model. The index i is then used to specify which root-to-leaf path of \tilde{T} is uncut: if $i = 0$ then every such path is cut, otherwise the (η_i, \tilde{r}) -path is uncut. To encode the rest of the solution, we associate with each tree arc (β, δ) a weight $\text{wt}_i((\beta, \delta))$ corresponding to the size of a minimum $P|_\beta$ -multiway-cut in $G|_\beta$.

We proceed with the formal construction of H_i . The vertex set of H_i is $V(H_i) = V(\tilde{T}) \uplus \{\mathbf{s}\} \uplus \{\Gamma\}$ where $\Gamma = \{\gamma(v) \mid v \in V(\tilde{G}) \setminus \tilde{P}\}$, that is, Γ contains a node of every non-terminal vertex in \tilde{G} . For every $z \in \Gamma$, we denote by $\gamma^{-1}(z)$ the corresponding vertex in $V(\tilde{G}) \setminus \tilde{P}$. The arc set of H_i is partitioned into four sets:

- the set $E_{\tilde{T}}$ of *tree arcs* containing all the edges of \tilde{T} oriented towards the root \tilde{r} ,
- the set $E_{\text{source}}^i = \{(\mathbf{s}, \text{top}_{\mathcal{M}}(p)) \mid p \in \tilde{P}_i\}$ of *source* arcs,
- the set $E_{\text{conn}} = \{(\alpha, \gamma(v)) \mid \gamma(v) \in \Gamma, \alpha \in \mathcal{M}(v) \cap V(\tilde{T})\}$ of *connection* arcs and
- the set $E_{\text{sink}} = \{(\gamma(v), \text{top}_{\mathcal{M}}(v)) \mid v \in V(\tilde{G}) \setminus \tilde{P}\}$ of *sink* arcs.

Furthermore, if $\tilde{P}_r \neq \emptyset$, then we let $E_{\text{rterm}} \subseteq E_{\tilde{T}}$ be the set of tree arcs $(\beta, \delta) \in E_{\tilde{T}}$ such that the edge $\beta\delta$ is contained in the model of the root terminal; otherwise, we let $E_{\text{rterm}} = \emptyset$. The weight function $\text{wt}_i : E(H_i) \rightarrow \mathbb{N} \cup \{\infty\}$ is defined as follows. For every $j \in [q]$, let ρ_j be the path in \tilde{T} from η_j to \tilde{r} and let $\vec{\rho}_j$ be the corresponding directed path in H_i (that is, $\vec{\rho}_j$ is the path in H_i from η_j to \tilde{r} consisting only of tree arcs). Then for every arc e of H_i ,

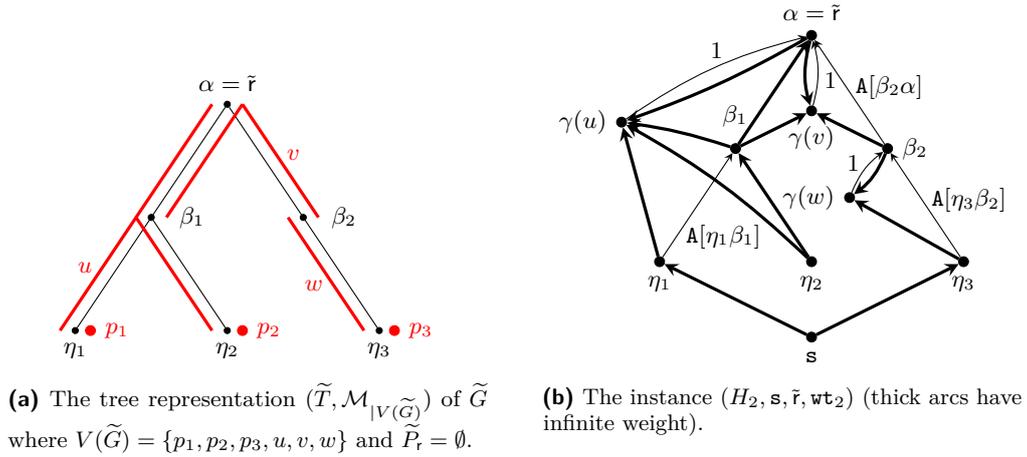
$$\text{wt}_i(e) = \begin{cases} A[e] & \text{if } i = 0 \text{ and } e \in E_{\tilde{T}} \setminus E_{\text{rterm}} \\ A[e] & \text{if } i \neq 0, e \in E_{\tilde{T}} \text{ and } e \text{ does not belong to the path } \vec{\rho}_i \\ 1 & \text{if } e \in E_{\text{sink}} \\ \infty & \text{otherwise.} \end{cases}$$

Note, in particular, that every arc in E_{rterm} (if any) has infinite weight. Similarly, if $i \neq 0$, then every arc of the path $\vec{\rho}_i$ has infinite weight. This completes the construction of the instance $(H_i, \mathbf{s}, \mathbf{t} = \tilde{r}, \text{wt}_i)$ (see Figure 3). It is easy to see that such an instance can be constructed in $\mathcal{O}(n^2)$ -time.

Now let X_0 be an (\mathbf{s}, \tilde{r}) -cut in H_0 such that $\text{wt}_0(X_0)$ is minimum; and if $\tilde{P}_r = \emptyset$, then for every $i \in [q]$, further let X_i be an (\mathbf{s}, \tilde{r}) -cut in H_i such that $\text{wt}_i(X_i)$ is minimum. For each $i \in [q]$, let us denote by $\text{cost}_i = A[\eta_i \text{ parent}_{T_0}(\eta_i)]$ and let $\text{cost}_0 = 0$. Then we set

$$A[\alpha \text{ parent}_{T_0}(\alpha)] = \begin{cases} |X_0| & \text{if } \tilde{P}_r \neq \emptyset \\ \min_{i \in [0, q]} \{|X_i| + \text{cost}_i\} & \text{otherwise} \end{cases}$$

In the following, for convenience, we let $I = [0, q]$ if $\tilde{P}_r = \emptyset$, and $I = \{0\}$ otherwise. We prove in Appendix B that the entry $A[\alpha \text{ parent}_{T_0}(\alpha)]$ is updated correctly. To this end, we show that $G|_{T_\alpha}$ has a $P|_{T_\alpha}$ -multiway-cut of size at most k if and only if there exists $i \in I$ such that H_i has an (\mathbf{s}, \tilde{r}) -cut of weight at most $k - \text{cost}_i$ w.r.t. wt_i .



■ **Figure 3** An illustration of the construction of the (s, t) -CUT instances.

From the correctness (cf. Lemmas 18 and 21) we conclude that $\mathbf{A}[\alpha \text{ parent}_{T_0}(\alpha)]$ indeed stores the size of a minimum P_{T_α} -multiway-cut in $G|_{T_\alpha}$. Since the construction of each H_i takes polynomial-time, an (s, t) -cut in H_i can be computed in polynomial time (see, for instance, [22]) and the number of H_i s is at most n , it takes polynomial-time to update $\mathbf{A}[\alpha \text{ parent}_{T_0}(\alpha)]$. Finally, since the number of edges of T is linear in n , the overall running time is polynomial in n , which proves Theorem 3. We remark that a more careful analysis of the running time of the algorithm leads to an upper bound of $\mathcal{O}(n^4)$.

6 Conclusion

In this article, we presented improved and new results regarding domination and cut problems on chordal graphs with bounded leafage. We presented an FPT algorithm running in time $2^{\mathcal{O}(\ell)} \cdot n^{\mathcal{O}(1)}$ for the DOMINATING SET problem on chordal graphs. Regarding cut problems, we proved that MULTICUT WITH UNDELETABLE TERMINALS on chordal graphs is $W[1]$ -hard when parameterized by the leafage. We also presented a polynomial-time algorithm for MULTIWAY CUT WITH UNDELETABLE TERMINALS on chordal graphs. We find it surprising that the complexity of this problem was not known before.

In the case of chordal graphs, we believe the leafage to be a more natural parameter than other popular parameters such as vertex cover, feedback vertex set or treewidth. It would be interesting to examine the structural parameterized complexity of problems such as LONGEST CYCLE, LONGEST PATH, COMPONENT ORDER CONNECTIVITY, s -CLUB CONTRACTION, INDEPENDENT SET RECONFIGURATION, BANDWIDTH, or CLUSTER VERTEX DELETION. These problems are known to be NP-complete on split graphs and admit polynomial-time algorithms on interval graphs. Hence it is plausible that they admit an FPT or XP algorithm on chordal graphs parameterized by the leafage. We believe it is a representative list, though not exhaustive, of problems that exhibit this behavior. In fact, it would be fascinating to find a natural problem that does not exhibit this behavior, i.e., a problem that is NP-complete on interval graphs but admits a polynomial-time algorithm on split graphs.

References

- 1 Liliana Alcón. On asteroidal sets in chordal graphs. *Discret. Appl. Math.*, 164:482–491, 2014. doi:10.1016/j.dam.2013.04.019.
- 2 Vikraman Arvind, Roman Nedela, Iliia Ponomarenko, and Peter Zeman. Testing isomorphism of chordal graphs of bounded leafage is fixed-parameter tractable. *CoRR*, abs/2107.10689, 2021. arXiv:2107.10689.
- 3 Hari Balakrishnan, Anand Rajaraman, and C. Pandu Rangan. Connected domination and steiner set on asteroidal triple-free graphs. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, Nicola Santoro, and Sue Whitesides, editors, *Algorithms and Data Structures, Third Workshop, WADS '93, Montréal, Canada, August 11-13, 1993, Proceedings*, volume 709 of *Lecture Notes in Computer Science*, pages 131–141. Springer, 1993. doi:10.1007/3-540-57155-8_242.
- 4 Kathleen D. Barnetson, Andrea C. Burgess, Jessica A. Enright, Jared Howell, David A. Pike, and Brady Ryan. The firebreak problem. *Networks*, 77(3):372–382, 2021. doi:10.1002/net.21975.
- 5 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. *Theory Comput. Syst.*, 65(4):662–686, 2021. doi:10.1007/s00224-020-09967-8.
- 6 Benjamin Bergougnoux and Mamadou Moustapha Kanté. More applications of the d-neighbor equivalence: Acyclicity and connectivity constraints. *SIAM J. Discret. Math.*, 35(3):1881–1926, 2021. doi:10.1137/20M1350571.
- 7 Benjamin Bergougnoux, Charis Papadopoulos, and Jan Arne Telle. Node multiway cut and subset feedback vertex set on graphs of bounded mim-width. *Algorithmica*, 84(5):1385–1417, 2022. doi:10.1007/s00453-022-00936-w.
- 8 Alan A. Bertossi. Dominating sets for split and bipartite graphs. *Inf. Process. Lett.*, 19(1):37–40, 1984. doi:10.1016/0020-0190(84)90126-1.
- 9 Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. *SIAM J. Comput.*, 47(1):166–207, 2018. doi:10.1137/140961808.
- 10 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theor. Comput. Sci.*, 511:66–76, 2013. doi:10.1016/j.tcs.2013.01.009.
- 11 Peter Buneman. A characterisation of rigid circuit graphs. *Discret. Math.*, 9(3):205–212, 1974. doi:10.1016/0012-365X(74)90002-8.
- 12 Yixin Cao. Linear recognition of almost interval graphs. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1096–1115. SIAM, 2016. doi:10.1137/1.9781611974331.ch77.
- 13 Maw-Shang Chang. Efficient algorithms for the domination problems on interval and circular-arc graphs. *SIAM J. Comput.*, 27(6):1671–1694, 1998. doi:10.1137/S0097539792238431.
- 14 Steven Chaplick and Juraj Stacho. The vertex leafage of chordal graphs. *Discret. Appl. Math.*, 168:14–25, 2014. doi:10.1016/j.dam.2012.12.006.
- 15 Rajesh Hemant Chitnis, Marek Cygan, Mohammad Taghi Hajiaghayi, and Dániel Marx. Directed subset feedback vertex set is fixed-parameter tractable. *ACM Trans. Algorithms*, 11(4):28:1–28:28, 2015. doi:10.1145/2700209.
- 16 Rajesh Hemant Chitnis, Mohammad Taghi Hajiaghayi, and Dániel Marx. Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. *SIAM J. Comput.*, 42(4):1674–1696, 2013. doi:10.1137/12086217X.
- 17 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.

- 18 Celina M. H. de Figueiredo, Raul Lopes, Aleksander Andrade de Melo, and Ana Silva. Parameterized algorithms for steiner tree and dominating set: Bounding the leafage by the vertex leafage. In Petra Mutzel, Md. Saidur Rahman, and Slamun, editors, *WALCOM: Algorithms and Computation - 16th International Conference and Workshops, WALCOM 2022, Jember, Indonesia, March 24-26, 2022, Proceedings*, volume 13174 of *Lecture Notes in Computer Science*, pages 251–262. Springer, 2022. doi:10.1007/978-3-030-96731-4_21.
- 19 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/s00453-016-0127-x.
- 20 Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the Tractability of Optimization Problems on H-graphs. *Algorithmica*, 82(9):2432–2473, 2020. doi:10.1007/s00453-020-00692-9.
- 21 Fedor V. Fomin, Pinar Heggernes, Dieter Kratsch, Charis Papadopoulos, and Yngve Villanger. Enumerating minimal subset feedback vertex sets. *Algorithmica*, 69(1):216–231, 2014. doi:10.1007/s00453-012-9731-6.
- 22 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.
- 23 Esther Galby, Dániel Marx, Philipp Schepper, Roohani Sharma, and Prafullkumar Tale. Domination and cut problems on chordal graphs with bounded leafage. *CoRR*, abs/2208.02850, 2022. doi:10.48550/arXiv.2208.02850.
- 24 Fanica Gavril. The intersection graphs of subtrees in tree are exactly the chordal graphs. *Combinatorica*, 1974.
- 25 P. C. Gilmore and A. J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canadian Journal of Mathematics*, 16:539–548, 1964. doi:10.4153/CJM-1964-055-5.
- 26 Petr A. Golovach, Pinar Heggernes, Pim van 't Hof, and Christophe Paul. Hadwiger number of graphs with small chordality. *SIAM J. Discret. Math.*, 29(3):1427–1451, 2015. doi:10.1137/140975279.
- 27 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.
- 28 Jiong Guo, Falk Hüffner, Erhan Kenar, Rolf Niedermeier, and Johannes Uhlmann. Complexity and exact algorithms for vertex multicut in interval and bounded treewidth graphs. *Eur. J. Oper. Res.*, 186(2):542–553, 2008. doi:10.1016/j.ejor.2007.02.014.
- 29 Michel Habib and Juraj Stacho. Polynomial-time algorithm for the leafage of chordal graphs. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 290–300. Springer, 2009. doi:10.1007/978-3-642-04128-0_27.
- 30 Michel Habib and Juraj Stacho. Reduced clique graphs of chordal graphs. *Eur. J. Comb.*, 33(5):712–735, 2012. doi:10.1016/j.ejc.2011.09.031.
- 31 Winfried Hochstättler, Johann L. Hurink, Bodo Manthey, Daniël Paulusma, Britta Peis, and Georg Still. In memoriam walter kern. *Discret. Appl. Math.*, 303:2–3, 2021. doi:10.1016/j.dam.2021.08.034.
- 32 Kyriaki Ioannidou, George B. Mertzios, and Stavros D. Nikolopoulos. The longest path problem has a polynomial solution on interval graphs. *Algorithmica*, 61(2):320–341, 2011. doi:10.1007/s00453-010-9411-3.
- 33 Dong Yeap Kang, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. A width parameter useful for chordal and co-comparability graphs. *Theor. Comput. Sci.*, 704:1–17, 2017. doi:10.1016/j.tcs.2017.09.006.
- 34 J. Mark Keil. Finding hamiltonian circuits in interval graphs. *Inf. Process. Lett.*, 20(4):201–206, 1985. doi:10.1016/0020-0190(85)90050-X.
- 35 Athanasios L. Konstantinidis and Charis Papadopoulos. Cluster deletion on interval graphs and split related graphs. *Algorithmica*, 83(7):2018–2046, 2021. doi:10.1007/s00453-021-00817-8.

- 36 Dieter Kratsch. Finding the minimum bandwidth of an interval graphs. *Inf. Comput.*, 74(2):140–158, 1987. doi:10.1016/0890-5401(87)90028-9.
- 37 Dieter Kratsch and Lorna Stewart. Approximating bandwidth by mixing layouts of interval graphs. *SIAM J. Discret. Math.*, 15(4):435–449, 2002. doi:10.1137/S0895480199359624.
- 38 C. Lekkekerker and J. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962. URL: <http://eudml.org/doc/213681>.
- 39 In-Jen Lin, Terry A. McKee, and Douglas B. West. The leafage of a chordal graph. *Discuss. Math. Graph Theory*, 18(1):23–48, 1998. doi:10.7151/dmgt.1061.
- 40 George S. Lueker and Kellogg S. Booth. A linear time algorithm for deciding interval graph isomorphism. *J. ACM*, 26(2):183–195, 1979. doi:10.1145/322123.322125.
- 41 Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006. doi:10.1016/j.tcs.2005.10.007.
- 42 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.
- 43 Pranabendu Misra, Fahad Panolan, Ashutosh Rai, Saket Saurabh, and Roohani Sharma. Quick separation in chordal and split graphs. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPICs*, pages 70:1–70:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.MFCS.2020.70.
- 44 Charis Papadopoulos. Restricted vertex multicut on permutation graphs. *Discret. Appl. Math.*, 160(12):1791–1797, 2012. doi:10.1016/j.dam.2012.03.021.
- 45 Charis Papadopoulos and Spyridon Tzimas. Polynomial-time algorithms for the subset feedback vertex set problem on interval graphs and permutation graphs. *Discret. Appl. Math.*, 258:204–221, 2019. doi:10.1016/j.dam.2018.11.017.
- 46 Charis Papadopoulos and Spyridon Tzimas. Computing a minimum subset feedback vertex set on chordal graphs parameterized by leafage. In Cristina Bazgan and Henning Fernau, editors, *Combinatorial Algorithms - 33rd International Workshop, IWOCA 2022, Trier, Germany, June 7-9, 2022, Proceedings*, volume 13270 of *Lecture Notes in Computer Science*, pages 466–479. Springer, 2022. doi:10.1007/978-3-031-06678-8_34.
- 47 James Richard Walter. *Representations of rigid cycle graphs*. Wayne State University, 1972.
- 48 Kevin White, Martin Farber, and William R. Pulleyblank. Steiner trees, connected domination and strongly chordal graphs. *Networks*, 15(1):109–124, 1985. doi:10.1002/net.3230150109.

A Proof of Lemma 7: Constructing Rest-Red-Blue-DomSet Instances

Let G be a chordal graph and let (T, \mathcal{M}) be a tree representation of G . We define the following functions.

- Let $f_T(G)$ denote the number of branching nodes $\gamma \in V(T)$ such that there exist both a red vertex and a blue vertex whose models contain γ .
- Let $f_r(G)$ denote the number of pairs of consecutive branching nodes α, β in T (that is, no node on the unique path in T from α to β is a branching node) such that there is red vertex whose model contains both α and β .
- Similarly, let $f_b(G)$ denote the number of pairs of consecutive branching nodes α, β in T such that there is blue vertex whose model contains both α and β .

We further define $\mu(G) := \text{lf}(G) + 2 \cdot (f_T(G) + f_r(G) + f_b(G))$. Note that, by definition, $\mu(G) \geq \text{lf}(G)$. We design a polynomial-time branching algorithm whose measure μ decreases in each branch. We first show that if $\mu(G) = \text{lf}(G)$ then $(G, (R, B), k)$ is in fact an instance of REST-RED-BLUE-DOMSET and then show how the branching algorithm proceeds.

Assume therefore that $\mu(G) = \text{lf}(G)$. Then $f_T(G) = f_r(G) = f_b(G) = 0$ by definition. However, when $f_T(G) = 0$, then, by definition, for every branching node $\gamma \in V(T)$, all the vertices containing γ in their model are either red or blue; and when $f_r(G) = f_b(G) = 0$

then, considering the fact that every model is a subtree in T , for every vertex in G , its model contains at most one branching node in T . Therefore if $\mu(G) = \text{lf}(G)$, then $(G, (R, B), k)$ is also an instance of REST-RED-BLUE-DOMSET.

Now assume that $\mu(G) > \text{lf}(G)$. Then $f_T(G) + f_r(G) + f_b(G) > 0$. We consider the following three exhaustive cases.

Case-I. $f_T(G) > 0$. Let γ be a branching node in T such that there is both a red-vertex and a blue-vertex whose models contain γ . Suppose that \mathcal{I} is a YES-instance of RED-BLUE-DOMSET and let D be a solution. Consider first the case where D includes a red vertex whose model contains γ . In this case, we return the instance $\mathcal{I}_1 = (G_1, (R_1, B_1), k)$ which is obtained as follows.

- Initialize $V(G_1) = V(G)$, $R_1 = R$, $B_1 = B$.
- Let T_1 be the tree obtained from T by adding a node δ and making it adjacent to γ only. Note that $V(T_1) \setminus \{\delta\} \subseteq V(T)$.
- For every red vertex $v \in V(G_1)$ such that $\gamma \in \mathcal{M}(v)$, add δ to its model, i.e., $\mathcal{M}_1(v) = \mathcal{M}(v) \cup \{\delta\}$.
- For every blue vertex $v \in V(G_1)$ such that $\gamma \in \mathcal{M}(v)$, delete v from $V(G_1)$.
- Add a new blue vertex x to $V(G_1)$ and to B_1 with $\mathcal{M}_1(x) = \{\delta\}$.
- For every (red or blue) vertex $v \in V(G)$ such that $\gamma \notin \mathcal{M}(v)$, define $\mathcal{M}_1(v_1) = \mathcal{M}(v)$.

It is easy to verify that (T_1, \mathcal{M}_1) is a tree representation of G_1 and that T_1 has exactly one more leaf than T , i.e., $\text{lf}(G_1) \leq \text{lf}(G) + 1$. However, since we have deleted all the blue vertices whose models contained γ , $f_T(G_1) = f_T(G) - 1$. As the other parts of the measure do not change, $\mu(G_1) < \mu(G)$.

In the second case where no vertex in D contains γ in its model, we return the instance $\mathcal{I}_2 = (G_2, (R_2, B), k)$ where G_2, R_2 are obtained from G, R , respectively, by deleting red vertices whose model contains γ . It is easy to verify that $\mu(G_2) < \mu(G)$.

If \mathcal{I} is a YES-instance, then at least one of \mathcal{I}_1 or \mathcal{I}_2 is a YES-instance as these two branches are exhaustive. If \mathcal{I}_1 is a YES-instance, then any optimum solution must include a red γ -vertex because of the newly added vertex x . As $R_2 \subseteq R$, if \mathcal{I}_2 is a YES-instance, then \mathcal{I} is a YES-instance. Hence, this branching step is correct.

Case-II. $f_T(G) = 0$ and $f_r(G) > 0$. Let α, β be two consecutive branching nodes in T such that there is a red vertex whose model contains both α and β . Suppose that \mathcal{I} is a YES-instance of RED-BLUE-DOMSET and let D be a solution. Consider the case where D includes a red vertex whose model contains both α and β . In this case, we return the instance $\mathcal{I}_1 = (G_1, (R_1, B_1), k)$ which is obtained as follows.

- Initialize $V(G_1) = R_1 = B_1 = \emptyset$.
- Let T_1 be the tree obtained from T by contracting the unique path $P_{\alpha\beta}$ from α to β in T and let $\gamma_{\alpha\beta}$ be the node resulting from this contraction. Add a node δ to T_1 and make it adjacent to $\gamma_{\alpha\beta}$ only. Note that $V(T_1) \setminus \{\gamma_{\alpha\beta}, \delta\} \subseteq V(T)$.
- For every red vertex $v \in V(G)$ such that $\mathcal{M}(v) \cap V(P_{\alpha\beta}) \neq \emptyset$, add a red vertex v_1 to $V(G_1)$ (and to R_1) with $\mathcal{M}_1(v_1) = (\mathcal{M}(v) \setminus V(P_{\alpha\beta})) \cup \{\gamma_{\alpha\beta}, \delta\}$.
- Add a new blue vertex x to $V(G_1)$ with $\mathcal{M}_1(x) = \{\delta\}$.
- For every (red or blue) vertex $v \in V(G)$ such that $\mathcal{M}(v) \cap V(P_{\alpha\beta}) = \emptyset$, add v_1 to G_1 (and to, respectively, either R_1 or B_1) with $\mathcal{M}_1(v_1) = \mathcal{M}(v)$.

Note that for every blue vertex $v \in V(G)$ such that $\mathcal{M}(G) \cap V(P_{\alpha\beta}) \neq \emptyset$, there is no corresponding blue vertex in G_1 . It is easy to verify that (T_1, \mathcal{M}_1) is a tree representation of G_1 and that T_1 has one more leaf than T which implies $\text{lf}(G_1) \leq \text{lf}(G) + 1$. Since we have contracted the path $P_{\alpha\beta}$ to obtain the node $\gamma_{\alpha\beta}$, $f_r(G_1) < f_r(G)$. As the other parts of the measure do not change, $\mu(G_1) < \mu(G)$.

In the second case where no vertex in D contains both α and β in its model, we return an instance $\mathcal{I}_2 = (G_2, (R_2, B), k)$ where G_2, R_2 are obtained from G, R , respectively, by deleting red vertices whose model contains both α and β . It is easy to verify that $\mu(G_2) < \mu(G)$. We argue as in the previous case for the correctness of this branching steps.

Case-III. $f_T(G) = 0$ and $f_b(G) > 0$. Let α, β be two consecutive branching nodes in T such that there is a blue vertex whose model contains both α and β . Note that since $f_T(G) = 0$, for every red vertex $v \in V(G)$ such that $\mathcal{M}(v) \cap V(P_{\alpha\beta}) \neq \emptyset$, in fact $\mathcal{M}(v) \subseteq V(P_{\alpha\beta}) \setminus \{\alpha, \beta\}$. Suppose that \mathcal{I} is a YES-instance of RED-BLUE-DOMSET and let D be a solution. Consider first the case where D includes a red vertex whose model is in $V(P_{\alpha\beta}) \setminus \{\alpha, \beta\}$. In this case, we return the instance $\mathcal{I}_1 = (G_1, (R, B_1), k)$ where G_1, B_1 are obtained from G and B as follows.

- Delete all the blue vertices whose model contains both α and β .
- Add a blue vertex x to $V(G_1)$ (and to B_1) with $\mathcal{M}(x) = V(P_{\alpha\beta}) \setminus \{\alpha, \beta\}$.

It is easy to verify that (T, \mathcal{M}) is a tree representation of G_1 and $f_b(G_1) < f_b(G)$. As the other parts of the measure do not change, $\mu(G_1) < \mu(G)$.

In the second case where there is no vertex in D whose model is in $V(P_{\alpha\beta}) \setminus \{\alpha, \beta\}$, we consider the following two subcases. If there is a blue vertex v such that $\mathcal{M}(v) \subseteq V(P_{\alpha\beta})$, then we return a trivial NO-instance. Otherwise, we return the instance $\mathcal{I}_2 = (G_2, (R_2, B_2), k)$ which is constructed as follows.

- Initialize $V(G_2) = R_2 = B_2 = \emptyset$.
- Let T_2 be the tree obtained from T by contracting the path $P_{\alpha\beta}$ from α to β in T and let $\gamma_{\alpha\beta}$ be the node resulting from this contraction. Note that $V(T_2) \setminus \{\gamma_{\alpha\beta}\} \subseteq V(T)$.
- For every (red or blue) vertex $v \in G$ such that $\mathcal{M}(v) \cap V(P_{\alpha\beta}) = \emptyset$, add a vertex v_2 to G_2 (and to, respectively, either R_2 or B_2) with $\mathcal{M}_2(v_2) = \mathcal{M}(v)$.
- For every blue vertex $v \in V(G)$ such that $\mathcal{M}(v) \cap V(P_{\alpha\beta}) \neq \emptyset$, add a blue vertex v_2 to $V(G_2)$ (and to B_2) with $\mathcal{M}_2(v_2) = (\mathcal{M}(v_2) \setminus V(P_{\alpha\beta})) \cup \{\gamma_{\alpha\beta}\}$.

Note that for any red vertex $v \in V(G)$ such that $\mathcal{M}(v) \subseteq V(P_{\alpha\beta}) \setminus \{\alpha, \beta\}$, there is no corresponding red vertex in G_2 . It is easy to verify that (T_2, \mathcal{M}_2) is a tree representation of G_2 . Furthermore, the number of leaves of T_2 is the same as T and $f_b(G_2) < f_b(G)$. As the other parts in the measure do not change, $\mu(G_2) < \mu(G)$.

The correctness of this branching step follows from the same arguments as in the previous cases and the fact that in the second case, since there is no red vertex whose model intersects $V(P_{\alpha\beta})$, it is safe to contract that path.

Finishing the Proof. The correctness of the overall algorithm follows from the correctness of branching steps in the above three cases. To bound its running time and the number of instances it outputs, note that $f_T(G) + f_r(G) + f_b(G) \leq 3 \cdot \text{lf}(G)$ as these functions either count the number of branching nodes or the unique paths containing exactly two (consecutive) branching nodes. ◀

B Correctness of the Algorithm for Multiway-Cut

► **Lemma 18.** *For any $i \in I$, if H_i has an (\mathbf{s}, \tilde{r}) -cut Y such that $\text{wt}_i(Y) \leq k - \text{cost}_i$, then $G_{|T_\alpha}$ has a $P_{|T_\alpha}$ -multiway-cut of size at most k .*

Proof. Assume that there exists $i \in I$ such that H_i has an (\mathbf{s}, \tilde{r}) -cut Y where $\text{wt}_i(Y) \leq k - \text{cost}_i$. For every $j \in [q] \setminus \{i\}$, let A_j be the set of tree arcs on the path $\vec{\rho}_j$ belonging to Y (recall that $\vec{\rho}_j$ is the path in H_i from η_j to \tilde{r} consisting only of tree arcs). Note that since Y is an (\mathbf{s}, \tilde{r}) -cut, $A_j \neq \emptyset$ for every $j \in [q] \setminus \{i\}$.

▷ **Claim 19.** For every terminal $j \in [q] \setminus \{i\}$, there exists an arc $(x, y) \in A_j$ such that for every $z \in N_{H_i}^+(x) \setminus (N_{H_i}^-(x) \cup \{y\})$, the sink arc with tail z belongs to Y .

Proof. Suppose for a contradiction that this does not hold for some index $j \in [q] \setminus \{i\}$, that is, for every arc $(x, y) \in A_j$, there exists $z \in N_{H_i}^+(x) \setminus (N_{H_i}^-(x) \cup \{y\})$ such that the sink arc with tail z does not belong to Y . Let $(x_1, y_1), \dots, (x_a, y_a)$ be the arcs of A_j ordered according to their order of appearance when traversing the path $\vec{\rho}_j$. We show that, in this case, there is a path from \mathbf{s} to \tilde{r} in $H - Y$. For every $b \in [a]$, denote by $Z_b \subseteq N_{H_i}^+(x_b) \setminus (N_{H_i}^-(x_b) \cup \{y_b\})$ the set of vertices z such that the sink arc with tail z does not belong to Y . Let $b_1, \dots, b_w \in [a]$ be the longest sequence defined as follows:

- $b_1 \in [a]$ is the largest index such that $Z_1 \cap Z_{b_1} \neq \emptyset$ and
- for every $l > 1$, $b_l \in [a]$ is the largest index such that $Z_{b_{l-1}+1} \cap Z_{b_l} \neq \emptyset$.

For every $l \in [w]$, consider a vertex $z_{b_l} \in Z_{b_l}$ and let $h_{b_l} \in N_{H_i}^+(z_{b_l})$ be the head of the sink arc with tail z_{b_l} . Then for every $l \in [w-1]$, h_{b_l} lies on the path $\vec{\rho}_j[y_{b_l}, x_{b_{l+1}}]$: indeed, since $z_{b_l} \notin Z_{b_{l+1}}$ by the choice of b_l , either $z_{b_l} \notin N_{H_i}^+(x_{b_{l+1}})$ or $z_{b_l} \in N_{H_i}^+(x_{b_{l+1}}) \cap N_{H_i}^-(x_{b_{l+1}})$; but $z_{b_l} \in N_{H_i}^+(x_{b_l}) \setminus N_{H_i}^-(x_{b_l})$ by construction, and so, h_{b_l} necessarily lies on $\vec{\rho}_j[y_{b_l}, x_{b_{l+1}}]$.

Now observe that, by maximality of the sequence, $b_w = a$: indeed, if $b_w < a$ then the sequence could be extended as $Z_{b_w+1} \neq \emptyset$ by assumption. Since $z_{b_w} \notin N_{H_i}^-(x_{b_w})$, this implies, in particular, that h_{b_w} lies on the path $\vec{\rho}_j[y_{b_w}, \tilde{r}]$. It follows that

$$\mathbf{s} \vec{\rho}_j[\eta_j, x_1] z_{b_1} \vec{\rho}_j[h_{b_1}, x_{b_1+1}] z_{b_2} \dots z_{b_l} \vec{\rho}_j[h_{b_l}, x_{b_l+1}] z_{b_{l+1}} \dots \vec{\rho}_j[h_{b_w-1}, x_{b_w-1+1}] z_{b_w} L[h_{b_w}, \tilde{r}]$$

is a path from \mathbf{s} to \tilde{r} in $H - Y$, a contradiction which proves our claim. ◁

For every $j \in [q] \setminus \{i\}$, let $e_j = (x_j, y_j) \in A_j$ be the arc closest to \tilde{r} such that for every $z \in N_{H_i}^+(x_j) \setminus (N_{H_i}^-(x_j) \cup \{y_j\})$, the sink arc with tail z belongs to Y (note that we may have $e_j = e_{j'}$ for two distinct $j, j' \in [q] \setminus \{i\}$). Denote by $\underline{E} = \{e_j \mid j \in [q] \setminus \{i\}\} \cup \{e^*\}$ where $e^* = (\eta_i, \text{parent}(\eta_i))$. For every $e = (x, y) \in \underline{E}$, let $\tilde{P}_e \subseteq \tilde{P}_i$ be the set of terminals in \tilde{P}_i which are also terminals in the instance restricted to T_x . Note that $\{P_e \mid e \in \underline{E} \setminus \{e^*\}\}$ is a partition of \tilde{P}_i : indeed, by construction, every $p \in \tilde{P}_i$ belongs to at least one such set and if there exist $e, e' \in \underline{E} \setminus \{e^*\}$ such that $\tilde{P}_e \cap \tilde{P}_{e'} \neq \emptyset$, then for any $j \in [q] \setminus \{i\}$ such that $p_j \in P_e \cap P_{e'}$, $e, e' \in A_j$; in particular, both e and e' lie on the path $\vec{\rho}_j$, a contradiction to the choice of the arc in A_j .

Now for every $e = (x, y) \in \underline{E}$, let S_e be a minimum $P_{|T_x}$ -multiway-cut in $G_{|T_x}$ and denote by $N_e = N_{H_i}^+(x) \setminus (N_{H_i}^-(x) \cup \{y\})$. We define

$$S = S_{e^*} \cup \bigcup_{e \in \underline{E} \setminus \{e^*\}} S_e \cup \{\gamma^{-1}(z) \mid z \in N_e\}.$$

▷ **Claim 20.** S is a $P_{|T_\alpha}$ -multiway-cut in $G_{|T_\alpha}$.

Proof. Since for every $e = (x, y) \in E$, S_e is a P_{T_x} -multiway-cut in $G_{|T_x}$, it is in fact enough to show that for every $e, e' \in E$, $p \in \tilde{P}_e$ and $p' \in \tilde{P}_{e'}$, there is no path from p to p' in $G_{|T_\alpha} - S$.

Consider therefore $j, j' \in [q] \setminus \{i\}$ such that $p_j \in \tilde{P}_e$ and $p_{j'} \in \tilde{P}_{e'}$ for two distinct $e, e' \in E$. Since, as shown above, $\{\tilde{P}_f \mid f \in E \setminus \{e^*\}\}$ is a partition of \tilde{P}_i , $p_{j'} \notin \tilde{P}_e$ and $p_j \notin \tilde{P}_{e'}$; in particular, e' does not lie on the path $\vec{\rho}_j$ and e does not lie on the path $\vec{\rho}_{j'}$. It follows that any path in $G_{|T_\alpha}$ from p_j to $p_{j'}$ contains at least one vertex x whose model contains the edge corresponding to e ; but then, $\gamma(x) \in N_e$ and so, $x \in S$ by construction. Thus, there is no path from p_j to $p_{j'}$ in $G_{|T_\alpha} - S$. \triangleleft

Finally, note that, by construction,

$$\begin{aligned} |S| &= |S_{e^*}| + \sum_{e \in E \setminus \{e^*\}} |S_e| + \left| \bigcup_{e \in E \setminus \{e^*\}} \{\gamma^{-1}(z) \mid z \in N_e\} \right| \\ &= |S_{e^*}| + \sum_{e \in E \setminus \{e^*\}} \text{wt}_i(e) + \sum_{z \in \bigcup_{e \in E \setminus \{e^*\}} N_e} \text{wt}_i((z, \text{top}_{\mathcal{M}}(\gamma^{-1}(z)))) \\ &\leq \text{cost}_i + \text{wt}_i(Y) \leq k \end{aligned}$$

which concludes the proof. \blacktriangleleft

► **Lemma 21.** *If $G_{|T_\alpha}$ has a P_{T_α} -multiway-cut X of size at most k , then there exists $i \in I$ such that H_i has an (s, \tilde{r}) -cut Y where $\text{wt}_i(Y) \leq k - \text{cost}_i$.*

Proof. Recall that for every $j \in [q]$, ρ_j is the unique (η_j, \tilde{r}) -path in \tilde{T} . To prove the lemma, we first show the following.

▷ **Claim 22.** If there exists $i \in [q]$ such that $G_{|T_\alpha}$ has a P_{T_α} -multiway-cut X of size at most k where

- (1) X does not destroy any edge of ρ_i and
 - (2) for every $j \in [q] \setminus \{i\}$, X destroys an edge of ρ_j ,
- then H_i has an (s, \tilde{r}) -cut Y such that $\text{wt}_i(Y) \leq k - \text{cost}_i$.

Proof. Assume that such an index $i \in [q]$ exists and let X be a P_{T_α} -multiway-cut X of size at most k satisfying item (1) and (2). Note that since X does not destroy any edge of ρ_i , $\tilde{P}_r = \emptyset$ for, otherwise, p_i and the root terminal would be in the same connected component of $G_{|T_\alpha} - X$ thereby contradicting the fact that X is a P_{T_α} -multiway-cut. For every $j \in [q] \setminus \{i\}$, let $e_j \in E(\tilde{T})$ be the closest edge to η_j on ρ_j such that $\text{ver}(e_j) \subseteq X$ (note that the edges e_1, \dots, e_q are not necessarily pairwise distinct). Denote by $E = \{e_j \mid j \in [q] \setminus \{i\}\}$. We construct an (s, \tilde{r}) -cut Y in H_i as follows: Y contains the tree arcs of H_i corresponding to the edges in E and for each $v \in X$ such that $\mathcal{M}(v)$ contains at least one edge of E (that is, $v \in \text{ver}(e)$ for some edge $e \in E$), we include in Y the sink arc $(\gamma(v), \text{top}_{\mathcal{M}}(v))$ of $E(H_i)$. Let us show that Y is indeed an (s, \tilde{r}) -cut in H_i .

For every $j \in [q] \setminus \{i\}$, let $V_-^j \subseteq V(\tilde{T})$ ($V_+^j \subseteq V(\tilde{T})$, respectively) be the set of nodes of the subpath of ρ_j from η_j to the tail of e_j (the head of e_j to \tilde{r} , respectively). We contend that for every $j \in [q] \setminus \{i\}$, there is no (V_-^j, V_+^j) -path in $H_i - Y$. Note that if true, this would prove that Y is indeed an (s, \tilde{r}) -cut in H_i . For the sake of contradiction, suppose that, for some $j \in [q] \setminus \{i\}$, there is a path L in $H_i - Y$ from a vertex $x \in V_-^j$ to a vertex $y \in V_+^j$. Since the tree arc in H_i corresponding e_j belongs to Y , there must exist a vertex $z \in V(L)$ such that $N_{H_i}^-(z) \cap V_-^j \cap V(L) \neq \emptyset$ and $N_{H_i}^+(z) \cap V_+^j \cap V(L) \neq \emptyset$; in particular, the sink arc e with

14:24 Domination and Cut Problems on Chordal Graphs with Bounded Leafage

tail z must belong to L . By construction of H_i , it must then be that $\mathcal{M}(\gamma^{-1}(z))$ contains the edge e_j , that is, $\gamma^{-1}(z) \in \mathbf{ver}(e_j)$; but then, $\gamma^{-1}(z) \in X$ and so, $e \in Y$ by construction, a contradiction which proves our claim.

Let us finally show that $\mathbf{wt}_i(Y) \leq k - \mathbf{cost}_i$. To this end, for every $e \in E$, let $X_e \subseteq X$ be the restriction of X to T_{t_e} where t_e is the endpoint of e the furthest from \tilde{r} (note that for any two distinct $e, e' \in E$, $X_e \cap X_{e'} = \emptyset$). Then, for every $e \in E$, X_e is a $P_{|T_{t_e}|}$ -multiway-cut in $G_{|T_{t_e}|}$ and so, $\mathbf{wt}_i(e) \leq |X_e|$. Similarly, the restriction X_i of X to T_{η_i} is a $P_{|T_{\eta_i}|}$ -multiway-cut in $G_{|T_{\eta_i}|}$ and so, $|X_i| \geq \mathbf{cost}_i$ (note that, by construction, $X_i \cap X_e = \emptyset$ for every $e \in E$). Letting $X' = \bigcup_{e \in E} \mathbf{ver}(e)$, it then follows from the definition of Y that

$$\mathbf{wt}_i(Y) = |X'| + \sum_{e \in E} \mathbf{wt}_i(e) \leq |X'| + \sum_{e \in E} |X_e| \leq |X| - |X_i| \leq k - \mathbf{cost}_i$$

as $X' \cap X_i = \emptyset$ and for every $e \in E$, $X' \cap X_e = \emptyset$. ◁

Using similar arguments, we can also prove the following.

▷ **Claim 23.** If $G_{|T_\alpha}$ has a $P_{|T_\alpha|}$ -multiway-cut X of size at most k such that for every $i \in [q]$, X destroys an edge of ρ_i , then H_0 has an (\mathbf{s}, \tilde{r}) -cut Y such that $\mathbf{wt}_i(Y) \leq k$.

To conclude the proof of Lemma 21, let us show that for any $P_{|T_\alpha|}$ -multiway-cut S in $G_{|T_\alpha|}$, S destroys an edge of every root-to-leaf path of \tilde{T} , except for at most one when $\tilde{P}_r = \emptyset$. Note that if the claim is true, the lemma would then follow from Claims 22 and 23.

Let S be a $P_{|T_\alpha|}$ -multiway-cut in $G_{|T_\alpha|}$. Observe first that if $\tilde{P}_r \neq \emptyset$ then for every $i \in [q]$, S must destroy an edge of ρ_i for, otherwise, p_i and the root terminal are in the same connected component of $G_{|T_\alpha|} - S$, thereby contradicting the fact that S is a $P_{|T_\alpha|}$ -multiway-cut. Assume therefore that $\tilde{P}_r = \emptyset$ and suppose, for the sake of contradiction, that there exist two distinct indices $i, j \in [q]$ such that S destroys no edge of ρ_i and no edge of ρ_j . Then for every edge e of $\rho_i \cup \rho_j$, $\mathbf{ver}(e) \setminus S \neq \emptyset$: for each such edge e , let $\alpha_e \in \mathbf{ver}(e) \setminus S$. It is now not difficult to see that there is a path in $G_{|T_\alpha|} - S$ from p_i to p_j using only vertices from $\{\alpha_e \mid e \text{ is an edge of } \rho_i \cup \rho_j\}$, a contradiction to the fact that S be a $P_{|T_\alpha|}$ -multiway-cut in $G_{|T_\alpha|}$. ◀

Slim Tree-Cut Width

Robert Ganian  

Algorithms and Complexity Group, TU Wien, Austria

Viktoriia Korchemna  

Algorithms and Complexity Group, TU Wien, Austria

Abstract

Tree-cut width is a parameter that has been introduced as an attempt to obtain an analogue of treewidth for edge cuts. Unfortunately, in spite of its desirable structural properties, it turned out that tree-cut width falls short as an edge-cut based alternative to treewidth in algorithmic aspects. This has led to the very recent introduction of a simple edge-based parameter called edge-cut width [WG 2022], which has precisely the algorithmic applications one would expect from an analogue of treewidth for edge cuts, but does not have the desired structural properties.

In this paper, we study a variant of tree-cut width obtained by changing the threshold for so-called thin nodes in tree-cut decompositions from 2 to 1. We show that this “slim tree-cut width” satisfies all the requirements of an edge-cut based analogue of treewidth, both structural and algorithmic, while being less restrictive than edge-cut width. Our results also include an alternative characterization of slim tree-cut width via an easy-to-use spanning-tree decomposition akin to the one used for edge-cut width, a characterization of slim tree-cut width in terms of forbidden immersions as well as an approximation algorithm for computing the parameter.

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

Keywords and phrases tree-cut width, structural parameters, graph immersions

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.15

Related Version *Predecessor*: <https://arxiv.org/abs/2206.15091>

Funding Robert Ganian and Viktoriia Korchemna acknowledge support by the Austrian Science Fund (FWF, project Y1329).

1 Introduction

Understanding which structural properties of inputs allow us to overcome the inherent intractability of problems of interest is a fundamental research area in computer science. In the context of parameterized complexity, one typically approaches this by asking which structural parameters of the input (or its graph representation) give rise to a fixed-parameter algorithm for a targeted problem. Treewidth [35] is the most prominent example of such a structural parameter, and can be viewed as a guarantee that a graph is iteratively decomposable along small vertex separators. Many problems are known to be fixed-parameter tractable when parameterized by treewidth – and for those that are not, there is a well-studied hierarchy of more restrictive¹ parameters based on vertex separators or vertex deletion that can sometimes be used instead (see, e.g., Figure 1 in [3]). Examples of such parameters include the vertex cover number [11, 14], the feedback vertex number [2, 27] and treedepth [19, 26, 31, 32].

However, such vertex based parameters seem ill suited for handling some problems. Consider, for instance, the classical EDGE DISJOINT PATHS problem (EDP): unlike VERTEX DISJOINT PATHS, EDP remains NP-hard not only on graphs of bounded treewidth, but

¹ We view parameter α as being more restrictive than parameter β if every graph class where α is bounded also has bounded β , but the opposite does not hold.



© Robert Ganian and Viktoriia Korchemna;

licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 15; pp. 15:1–15:18

Leibniz International Proceedings in Informatics



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

even on graphs with a vertex cover number of at most 3 [13]. While this effectively rules out the use of all parameters based on vertex separators, there is an intuitive expectation that EDP should be fixed-parameter tractable w.r.t. parameters that can guarantee an iterative decomposition of the graph along small edge cuts. Indeed, EDP is known to be fixed-parameter tractable w.r.t. two basic parameterizations which provide such a guarantee: the feedback edge number [20] and treewidth plus maximum degree [21].

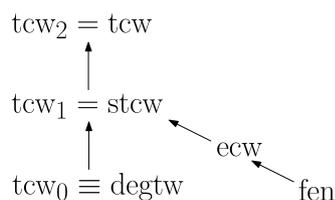
An ideal solution for handling such problems on more general inputs would be to use an alternative to treewidth that would be designed around edge cuts rather than vertex separators, one which would provide a unified justification for tractability w.r.t. the two basic “edge-cut restricting” parameterizations mentioned above. A candidate for such a parameter was proposed by Wollan, who defined *tree-cut width* along with *tree-cut decompositions* and described these as a variation of tree decompositions based on edge cuts instead of vertex separators [37]. But while it is true that “tree-cut decompositions share many of the natural properties of tree decompositions” [30], from the perspective of algorithmic design tree-cut width seems to behave differently than an edge-cut based alternative to treewidth. Indeed, not only does it fall short of yielding a fixed-parameter algorithm for EDP [20], it also fails to provide such algorithms for other problems one would expect to be fixed-parameter tractable w.r.t. an edge-cut based analogue to treewidth. In fact, out of twelve such problems where a tree-cut width parameterization has been pursued so far, only four are fixed-parameter tractable [16, 17] while eight turn out to be $W[1]$ -hard [5, 16, 18, 20, 24] (see the Related Work at the end of the Introduction for details).

Very recently, Brand, Ceylan, Ganian, Hatschka and Korchemna [4] introduced a parameter called *edge-cut width* which aimed at filling this gap in our understanding of edge-cut based graph parameters. On the algorithmic side, edge-cut width has precisely the properties one could hope to see in an edge-based analogue to treewidth: not only does it yield fixed-parameter algorithms for all twelve “candidate” problems [4], but it is also based on a very simple type of decomposition that is much easier to use than tree-cut decompositions. That being said, already the authors of that paper noted that the structural properties of edge-cut width are far from ideal – for instance, it is the only algorithmically used parameter we are aware of that is not closed under vertex deletion. Moreover, while edge-cut width is less restrictive than the feedback edge number, unlike tree-cut width it is incomparable to treewidth plus maximum degree (even in an asymptotic sense). Because of this, it cannot act as a common generalization that would capture both of these basic approaches of enforcing decomposability along small edge cuts.

Contribution. In this paper, we identify a graph parameter which combines the advantages of tree-cut width and edge-cut width while avoiding all of the shortcomings listed above. However, before we introduce it, it will be useful to establish at least some intuitive understanding of tree-cut width².

A graph G has tree-cut width at most k if it admits a tree-cut decomposition T of width k , whereas T is a rooted tree and its nodes act as bags that form a partitioning of $V(G)$. A non-root node t of T defines an edge cut between all vertices in the subtree rooted at t , and the rest of the graph. The definition of tree-cut width then restricts, for each node t , the number of its children defining an edge cut of size greater than 2. The constant “2” here arises from the structural properties Wollan aimed for when defining tree-cut width [37]; however, let us now pose the following question: How would the parameter change if we used a different constant c here instead?

² Formal definitions are provided in Section 2.



■ **Figure 1** Hierarchy of graph parameters based on edge cuts. Here ecw denotes edge-cut width and $\text{deg} \text{tw}$ denotes treewidth plus maximum degree. tcw_i denotes the parameter obtained from tree-cut width by setting the constant c described above to i . An arrow from p to q represents the fact that p is more restrictive than q , while asymptotic equivalence is depicted by \equiv .

On one hand, it is not difficult to observe that values of $c > 2$ would immediately lead to parameters without the properties we are aiming for, since these would be constant for, e.g., all 3-regular graphs. On the other hand, we show that for $c = 0$, one obtains an asymptotically equivalent characterization of one of the previously mentioned basic edge-cut restricting parameterizations: treewidth plus maximum degree. Our parameter of interest is then the outcome of setting $c = 1$; since this can be viewed as a variant of tree-cut width where all but a few children of each node need to have “even slimmer” edge-cuts, we refer to it as *slim tree-cut width* (stcw).

On the structural side, we show that stcw inherits the desirable properties of its “non-slim” namesake. In particular, unlike edge-cut width [4], stcw is closed under edge sums, vertex and edge deletion, as well as under the graph immersion operation. Similarly as Wollan did for tree-cut width [37], we also provide a set of forbidden immersions asymptotically characterizing stcw . Furthermore, we show that stcw is a common generalization of edge-cut width (and hence the feedback edge number), and treewidth plus maximum degree (see Figure 1).

Next, as one of our arguably most surprising results, we show that stcw is asymptotically equivalent to a slight generalization of edge-cut width: instead of measuring the width over the input graph G , we ask for the minimum edge-cut width of any supergraph of G . The transformation between these parameters is constructive and has interesting algorithmic implications. First of all, when designing algorithms it allows us to avoid the use of often cumbersome tree-cut decompositions, and instead opt for the simpler decompositions used for edge-cut width – which are nothing else than spanning trees (in this case of a supergraph). Second, all of the fixed-parameter algorithms recently designed for edge-cut width [4] rely on a dynamic programming traversal of the spanning tree, and can be straightforwardly adapted to work on spanning trees of supergraphs instead. This means that one can essentially reuse the same proofs to establish fixed-parameter tractability of all considered “candidate” problems w.r.t. stcw .

Naturally, a crucial prerequisite for algorithmically applying stcw is that we can actually compute it, or more precisely compute a suitable decomposition for graphs of small stcw . While the problem of computing an optimal decomposition remains open even for tree-cut width, a fixed-parameter approximation algorithm was obtained by Kim, Oum, Paul, Sau and Thilikos [28] and this suffices for the purposes of establishing fixed-parameter tractability. We obtain a similar outcome here and also provide a fixed-parameter approximation algorithm for stcw , albeit with a worse approximation factor than for tree-cut width.

■ **Table 1** The twelve candidate problems and their complexity w.r.t. edge-cut based parameters, where *degtw* denotes the maximum degree plus treewidth. Slim tree-cut width provides a unified explanation for why these problems are FPT w.r.t. both edge-cut width and *degtw*, and lifts these results to more general inputs.

Problem	tree-cut width	edge-cut width	degtw	stcw
CAPACITATED VERTEX COVER	FPT [16]	FPT	FPT	FPT
CAPACITATED DOMINATING SET	FPT [16]	FPT	FPT	FPT
IMBALANCE	FPT [16]	FPT	FPT	FPT
BOUNDED DEGREE DELETION	FPT [17]	FPT	FPT	FPT
EDGE DISJOINT PATHS	W[1]-hard [20]	FPT [4]	FPT [21]	FPT
LIST COLORING	W[1]-hard [16]	FPT [4]	FPT [16]	FPT
PRECOLORING EXTENSION	W[1]-hard [16]	FPT [4]	FPT [16]	FPT
BOOLEAN CONSTRAINT SATISFACTION	W[1]-hard [16]	FPT [4]	FPT [36]	FPT
BAYESIAN NETWORK STRUCTURE LEARNING	W[1]-hard [18]	FPT [4, 18]	FPT [33]	FPT
POLYTREE LEARNING	W[1]-hard [18]	FPT [4, 18]	FPT [18]	FPT
MIN. CHANGEOVER COST ARBORESCENCE	W[1]-hard [24]	FPT [4]	FPT [25]	FPT
MSRTIL ³	W[1]-hard [5]	FPT [4]	FPT [1, 5]	FPT

Related Work. Tree-cut width parameterizations were typically considered for problems which are not fixed-parameter tractable (FPT) w.r.t. treewidth, but are FPT w.r.t. feedback edge number and also FPT w.r.t. treewidth plus maximum degree. The twelve candidate problems where tree-cut width parameterizations have been considered are shown in Table 1.

The structural properties of tree-cut width have also been studied in a number of recent papers [22, 23]. Last but not least, we note that a preprint exploring a different parameter that is aimed at providing an edge-based alternative to treewidth was recently authored by Magne, Paul, Sharma and Thilikos [29]; the parameter is based on different ideas and is incomparable to both tree-cut width and slim tree-cut width.

2 Preliminaries

We use standard terminology for graph theory [9] and assume basic familiarity with the parameterized complexity paradigm including, in particular, the notions of *fixed-parameter tractability* and *W[1]-hardness* [8, 10]. Let \mathbb{N} denote the set of natural numbers including zero. We use $[i]$ to denote the set $\{0, 1, \dots, i\}$.

The *(open) neighborhood* of a vertex $x \in V(G)$ is the set $\{y \in V(G) \mid xy \in E(G)\}$ and is denoted by $N_G(x)$. For a vertex subset X , the neighborhood of X is defined as $\bigcup_{x \in X} N_G(x) \setminus X$ and denoted by $N_G(X)$; we drop the subscript if the graph is clear from the context. If H is a subgraph of G , we denote it by $H \subseteq G$. *Contracting* an edge $\{a, b\}$ is the operation of replacing vertices a, b by a new vertex whose neighborhood is $(N(a) \cup N(b)) \setminus \{a, b\}$. For a vertex set A (or edge set B), we use $G - A$ ($G - B$) to denote the graph obtained from G by deleting all vertices in A (edges in B), and we use $G[A]$ to denote the *subgraph induced on A* , i.e., $G - (V(G) \setminus A)$.

³ Maximum Stable Roommates with Ties and Incomplete Lists. For completeness, we note that the authors who showed W[1]-hardness w.r.t. tree-cut width also identified two additional restrictions which, when combined with tree-cut width, suffice for fixed-parameter tractability [5].

Let G be a graph and let x, y and z be three distinct vertices of G such that $(x, y), (y, z) \in E(G)$. To *lift* the pair of edges $(x, y), (y, z)$ means to delete the edges (x, y) and (y, z) from G and add (if it doesn't exist yet) a new edge (x, z) . We say that G contains H as a *weak immersion* (denoted $H \leq_I G$) if and only if H can be obtained from G by a sequence of edge deletion, vertex deletion, and lifting operations.

For a natural number k , we say that a graph G is a *k-edge sum* of vertex-disjoint graphs G_1 and G_2 if there exist vertices $v_i \in V(G_i)$ of degree k for $i = 1, 2$ and a bijection $\pi : N_{G_1}(v_1) \rightarrow N_{G_2}(v_2)$ such that G is obtained from $(G_1 - \{v_1\}) \cup (G_2 - \{v_2\})$ by adding an edge $(v, \pi(v))$ for every $v \in N_{G_1}(v_1)$. In this case we write $G = G_1 \oplus_k G_2$. Observe that the same pair of graphs may produce different *k-edge sums*.

Given two graph parameters $\alpha, \beta : G \mapsto \mathbb{N}$, we say that α *dominates* β if there exists a function p such that for each graph G , $\alpha(G) \leq p(\beta(G))$. If α dominates β but β does not dominate α , we often say that β is more restrictive than α ; as an example, treewidth dominates the vertex cover number. Two parameters that dominate each other are called asymptotically equivalent.

Tree-cut Width. The notion of tree-cut decompositions was introduced by Wollan [37], see also subsequent work by Marx and Wollan [30]. A family of subsets X_1, \dots, X_k of X is a *near-partition* of X if they are pairwise disjoint and $\bigcup_{i=1}^k X_i = X$, allowing the possibility of $X_i = \emptyset$.

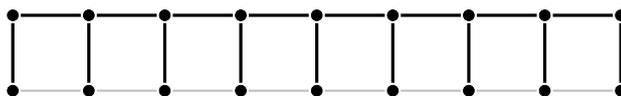
► **Definition 1.** A tree-cut decomposition of G is a pair (T, \mathcal{X}) which consists of a rooted tree T and a near-partition $\mathcal{X} = \{X_t \subseteq V(G) \mid t \in V(T)\}$ of $V(G)$. A set in the family \mathcal{X} is called a *bag* of the tree-cut decomposition.

For any node t of T other than the root r , let $e(t) = ut$ be the unique edge incident to t on the path to r . Let T_u and T_t be the two connected components in $T - e(t)$ which contain u and t , respectively. Note that $(\bigcup_{q \in T_u} X_q, \bigcup_{q \in T_t} X_q)$ is a near-partition of $V(G)$, and we use E_t to denote the set of edges with one endpoint in each part. We define the *adhesion* of t ($\text{adh}(t)$) as $|E_t|$; we explicitly set $\text{adh}(r) = 0$ and $E(r) = \emptyset$. The adhesion of (T, \mathcal{X}) is then $\text{adh}(T, \mathcal{X}) = \max_{t \in V(T)} \text{adh}(t)$.

The *torso* of a tree-cut decomposition (T, \mathcal{X}) at a node t , written as H_t , is the graph obtained from G as follows. If T consists of a single node t , then the torso of (T, \mathcal{X}) at t is G . Otherwise, let T_1, \dots, T_ℓ be the connected components of $T - t$. For each $i = 1, \dots, \ell$, the vertex set $Z_i \subseteq V(G)$ is defined as the set $\bigcup_{b \in V(T_i)} X_b$. The torso H_t at t is obtained from G by *consolidating* each vertex set Z_i into a single vertex z_i (this is also called *shrinking* in the literature). Here, the operation of consolidating a vertex set Z into z is to substitute Z by z in G , and for each edge e between Z and $v \in V(G) \setminus Z$, adding an edge zv in the new graph. We note that this may create parallel edges.

The operation of *suppressing* (also called *dissolving* in the literature) a vertex v of degree at most 2 consists of deleting v , and when the degree is two, adding an edge between the neighbors of v . Given a connected graph G and $X \subseteq V(G)$, let the *3-center* of (G, X) be the unique graph obtained from G by exhaustively suppressing vertices in $V(G) \setminus X$ of degree at most two. Finally, for a node t of T , we denote by \tilde{H}_t the 3-center of (H_t, X_t) , where H_t is the torso of (T, \mathcal{X}) at t . Let the *torso-size* $\text{tor}(t)$ denote $|\tilde{H}_t|$.

► **Definition 2.** The *width* of a tree-cut decomposition (T, \mathcal{X}) of G is $\max_{t \in V(T)} \{\text{adh}(t), \text{tor}(t)\}$. The *tree-cut width* of G , or $\text{tcw}(G)$ in short, is the minimum width of (T, \mathcal{X}) over all tree-cut decompositions (T, \mathcal{X}) of G .



■ **Figure 2** Example of a graph G with a spanning tree T (thick black) such that $\text{ecw}(G) = \text{ecw}(G, T) = 3$. The feedback edge number of G can be made arbitrarily large in this fashion.

Without loss of generality, we shall assume that $X_r = \emptyset$. We conclude this subsection with some notation related to tree-cut decompositions. Given a tree node t , let T_t be the subtree of T rooted at t . Let $Y_t = \bigcup_{b \in V(T_t)} X_b$, and let G_t denote the induced subgraph $G[Y_t]$. A node $t \neq r$ in a rooted tree-cut decomposition is *thin* if $\text{adh}(t) \leq 2$ and *bold* otherwise.

A tree-cut decomposition (T, \mathcal{X}) is *nice* if it satisfies the following condition for every thin node $t \in V(T)$: $N(Y_t) \cap (\bigcup_{b \text{ is a sibling of } t} Y_b) = \emptyset$. The intuition behind nice tree-cut decompositions is that we restrict the neighborhood of thin nodes in a way which facilitates dynamic programming. Every tree-cut decomposition of width k can be transformed into a nice tree-cut decomposition of the same width in cubic time [16]. Moreover, the resulting nice decomposition has the following property. For a node t , let $B_t = \{b \text{ is a child of } t \mid |N(Y_b)| \leq 2 \wedge N(Y_b) \subseteq X_t\}$ denote the set of thin children of t whose neighborhood is a subset of X_t , and let $A_t = \{a \text{ is a child of } t \mid a \notin B_t\}$ be the set of all other children of t . Then $|A_t| \leq 2k + 1$ for every node t [16].

We refer to previous work [16, 28, 30, 37] for a detailed comparison of tree-cut width to other parameters. Here, we mention only that tree-cut width is dominated by treewidth and dominates treewidth plus maximum degree, which we denote $\text{degtw}(G)$. It also dominates the feedback edge number (the size of a minimum feedback edge set), denoted $\text{fen}(G)$.

► **Lemma 3** ([16, 30, 37]). *For every graph G , $\text{tw}(G) \leq 2\text{tcw}(G)^2 + 3\text{tcw}(G)$ and $\text{tcw}(G) \leq \text{fen}(G) + 1$ and $\text{tcw}(G) \leq 4\text{degtw}(G)^2$.*

Edge-Cut Width. The notion of edge-cut width was introduced by Brand et al. [4]. For a graph G and a maximal spanning forest T of G , let the *local feedback edge set* at $v \in V$ be

$$E_{\text{loc}}^{G,T}(v) = \{uw \in E(G) \setminus E(T) \mid \text{the unique path between } u \text{ and } w \text{ in } T \text{ contains } v\}.$$

► **Definition 4.** *The edge-cut width of the pair (G, T) is $\text{ecw}(G, T) = 1 + \max_{v \in V} |E_{\text{loc}}^{G,T}(v)|$, and the edge-cut width of G (denoted $\text{ecw}(G)$) is the smallest edge-cut width among all possible maximal spanning forests T of G .*

► **Proposition 5** ([4]). *For every graph G , $\text{tcw}(G) \leq \text{ecw}(G) \leq \text{fen}(G) + 1$.*

In fact, it was shown in [4] that the gaps in both inequalities can be arbitrary large, see Figure 2 for a simple example of the second one.

Edge-cut width is not closed under vertex or edge deletions and is incomparable to degtw [4]. However, the fact that its decomposition is simply a spanning tree makes it easier to work with in dynamic programming applications than, e.g., tree-cut decompositions [4].

3 Refined Measures for Tree-Cut Decompositions

3.1 Definitions and Comparison

Let us now define our parameter of interest, obtained by altering the threshold for when a vertex is suppressed (dissolved) in the definition of tree-cut width. Formally, let (T, \mathcal{X}) be some tree-cut decomposition of G . Given a connected graph Q and $X \subseteq V(Q)$, let the

2-center of (Q, X) be the unique graph obtained from Q by exhaustively deleting vertices in $V(Q) \setminus X$ of degree at most one. For a node t of T , we denote by \bar{H}_t^2 the 2-center of (H_t, X_t) , where H_t is the torso of (T, \mathcal{X}) at t . Let us denote $|\bar{H}_t^2|$ by $\text{tor}_2(t)$.

► **Definition 6.** *The slim width of a tree-cut decomposition (T, \mathcal{X}) of a graph G is $\text{stcw}(T, \mathcal{X}) = \max_{t \in V(T)} \{\text{adh}(t), \text{tor}_2(t)\}$. The slim tree-cut width of G , or $\text{stcw}(G)$ in short, is the minimum slim width of (T, \mathcal{X}) over all tree-cut decompositions (T, \mathcal{X}) of G .*

Observe that the difference in definitions of $\text{tcw}(G)$ and $\text{stcw}(G)$ is whether we dissolve the vertices of degree at most two or at most one in the torso in each node. At this point, it would be reasonable to ask what happens if we dissolve only isolated vertices (i.e., vertices of degree 0) from the torso. Naturally extending the notions of 2- and 3-center for a connected graph Q and $X \subseteq V(Q)$, we define the 1-center of (Q, X) as the graph obtained from Q by deleting isolated vertices in $V(Q) \setminus X$. For a node t of T , we denote by \bar{H}_t^1 the 1-center of (H_t, X_t) , where H_t is the torso of (T, \mathcal{X}) at t . Let us denote $|\bar{H}_t^1|$ by $\text{tor}_1(t)$.

► **Definition 7.** *The 0-width of a tree-cut decomposition (T, \mathcal{X}) of G is $\max_{t \in V(T)} \{\text{adh}(t), \text{tor}_1(t)\}$. The 0-tree-cut width of G , or $\text{tcw}_0(G)$ in short, is the minimum 0-width of (T, \mathcal{X}) over all tree-cut decompositions (T, \mathcal{X}) of G .*

It follows from the definitions that for any tree-cut decomposition (T, \mathcal{X}) of G , for each node t of T , $\text{tor}(t) \leq \text{tor}_2(t) \leq \text{tor}_1(t)$. In particular, the width of (T, \mathcal{X}) is upper-bounded by its slim width, while the latter does not exceed the 0-width of (T, \mathcal{X}) .

► **Corollary 8.** *For any graph G , $\text{tcw}(G) \leq \text{stcw}(G) \leq \text{tcw}_0(G)$.*

The gaps in these inequalities can be arbitrarily large – and, more strongly, tcw_0 is a more restrictive parameter than stcw , which is in turn more restrictive than tcw . Indeed, for the comparison of tcw_0 and stcw consider the class of stars which have slim tree-cut width 1. Let S_r denote the star with r leaves (i.e., the complete bipartite graph $K_{1,r}$).

► **Lemma 9.** *For every positive integer $r \geq 1$, $\text{tcw}_0(S_{r^2}) \geq r$.*

Proof. Let (T, \mathcal{X}) be a tree-cut decomposition of S_{r^2} of 0-width k where the bags of leaves are non-empty. Let t be the node of T such that X_t contains the vertex of degree r^2 . Observe that t has at most $\text{tor}_1(t) - |X_t| \leq k - |X_t|$ children. For every child t' of t , $Y_{t'}$ contains at most $\text{adh}(t') \leq k$ vertices of S_{r^2} . In total, Y_t contains at most $|X_t| + k \cdot (k - |X_t|) \leq k^2$ vertices of S_{r^2} . Together with at most $\text{adh}(t) \leq k$ vertices outside of Y_t , S_{r^2} has at most $k \cdot (k + 1)$ vertices and hence $k \geq r$. ◀

To show the gap between stcw and tcw , let us denote by W_r the graph on $2r + 1$ vertices consisting of r triangles sharing one vertex; here we call such graphs windmills, and refer to Figure 3 later for an illustration. The class of windmills has tree-cut width 2 but, as the following lemma shows, unbounded slim tree-cut width.

► **Lemma 10.** *For every positive integer $r \geq 1$, $\text{stcw}(W_{r^2}) \geq r$.*

Proof. The case $r = 1$ is straightforward. For $r \geq 2$, assume, to the contrary, that there exists a tree-cut decomposition (T, \mathcal{X}) of W_{r^2} of slim width at most $r - 1$. Let t be the node of T such that X_t contains the vertex of degree $2r^2$. Without loss of generality, we assume that all the leaves of T have non-empty bags. Then the adhesion of any child t' of t is at least two, as $Y_{t'}$ contains some vertex v of W_{r^2} and the two edge-disjoint paths from v to the high-degree vertex in t each contribute to $\text{adh}(t')$. Hence, t has at most $\text{tor}_2(t) \leq r - 1$

children. Moreover, for every child t' of t , $Y_{t'}$ intersects at most $\frac{r-1}{2}$ distinct triangles of $W_{r,2}$, since each such triangle contributes 2 to $\text{adh}(t')$. Hence, for every child t' of t , $Y_{t'}$ contains at most $r-1$ vertices of $W_{r,2}$. In total, $Y_t \setminus X_t$ contains at most $(r-1)^2$ vertices of $W_{r,2}$. Since both $\text{adh}(t)$ and $|X_t|$ are upper-bounded by $r-1$ and the former bounds the number of vertices outside of Y_t by $r-1$, this would mean that $W_{r,2}$ has at most $(r-1)^2 + 2r - 2$ vertices, a contradiction with the definition of $W_{r,2}$. \blacktriangleleft

Given a graph G and its nice tree-cut decomposition (T, \mathcal{X}) of width at most k , let us denote by $B_t^{(2)}$ the set of children of t from B_t with adhesion precisely two; notice that $B_t^{(2)}$ does not necessarily contain all children of t with adhesion precisely two, since some may lie in A_t . Observe that for every fixed vertex t of T , if x is an element of 2-center of the torso at t and $x \notin X_t$, then x corresponds either to the parent of t in T or to some child of t from $A_t \cup B_t^{(2)}$. Hence $\text{tor}_2(t) \leq 1 + |X_t| + |A_t| + |B_t^{(2)}| \leq 3k + 2 + |B_t^{(2)}|$.

► **Corollary 11.** *Let G be a graph with tree-cut decomposition (T, \mathcal{X}) of width at most k . Then for each node t of T it holds that $|B_t^{(2)}| \geq \text{tor}_2(t) - 3k - 2$.*

3.2 Weak Immersions

Naturally extending the result of Wollan for tree-cut width [37], we show that both slim and 0-tree-cut width are closed under weak immersions.

► **Theorem 12.** *If G and H are graphs such that $H \leq_I G$ then $\text{stcw}(H) \leq \text{stcw}(G)$ and $\text{tcw}_0(H) \leq \text{tcw}_0(G)$.*

Proof. It is sufficient to prove the statement when H is obtained from G by precisely one edge deletion, isolated vertex deletion or lifting a pair of edges. Let (T, \mathcal{X}) be a tree-cut decomposition of G of minimum slim (or 0-) width. Then (T, \mathcal{X}) is also a tree-cut decomposition of $G \setminus e$ for any edge e of G with the same or smaller slim (0-) width. Similarly for the isolated vertex deletion: we just need to delete the vertex from the corresponding bag. It remains to consider the case $H = G \setminus \{(x, y), (y, z)\} \cup (x, z)$ for some $(x, y), (y, z) \in E(G)$.

Notice that the lifting operation doesn't increase adhesion of any node t of T : if the edge (x, z) has endpoints in different connected components of $T \setminus e(t)$ then so does at least one of the edges (x, y) or (y, z) . To see that $\text{tor}_2(t)$ and $\text{tor}_1(t)$ do not increase either, denote by Q_G and Q_H the torsos at t in (T, \mathcal{X}) for graphs G and H correspondingly. Every vertex of Q_G corresponds to a non-empty subset of the vertices of G . Depending on how the vertices x, y and z are split among these subsets, it holds that either $E(Q_H) \subseteq E(Q_G)$ (which yields the same or smaller 1-center and 2-center) or Q_H is obtained from Q_G by splitting a pair of edges. For the latter, observe that $v \in V(Q_G) \setminus X_t$ is not in the 2-center of (Q_G, X_t) if and only if v belongs to some induced subtree of Q_G connected to the rest of Q_G by at most one edge. It is not hard to see that lifting the pair of edges preserves the property. For the 1-center the situation is even simpler: isolated vertices of Q_G remain isolated. \blacktriangleleft

Recall that the weak immersion relation \leq_I is a transitive, reflexive and antisymmetric relation on the set of finite graphs, i.e., a partial order. The previous theorem showed that stcw is monotone with respect to \leq_I . Our next goal is to find graphs of simple structure but large slim (or 0-) tree-cut width, such that forbidding them as weak immersions bounds the corresponding width of a graph. Wollan in [37] characterized such graphs for tree-cut width. Namely, he established the following dichotomy:

► **Theorem 13.**

(a) *If G is a graph such that $H_{2r,2} \leq_I G$ for some $r \geq 3$, then $\text{tcw}(G) \geq r$.*

(b) *There exists a function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that if $\text{tcw}(G) \geq f(r)$, then $H_r \leq_I G$, $r \in \mathbb{N}$.*

Here H_r denotes the r -wall, the graph which can be obtained from the $r \times r$ grid by deleting every second vertical edge in each row, see [37] for the definition and Figure 3 for an illustration. We are going to complete the family of excluded immersions to provide similar characterizations for 0-tree-cut width and slim tree-cut width. Recall that the families of stars S_r and windmills W_r have unbounded 0- and slim tree-cut width, respectively (Lemmas 9 and 10). Combining this with Theorem 12, we immediately obtain:

► **Lemma 14.** *For every positive integer r , if $\text{stcw}(G) < r$ ($\text{tcw}_0(G) < r$), then G does not admit $W_{r,2}$ ($S_{r,2}$, respectively) as a weak immersion.*

As we will show in the remainder of this subsection, excluding W_r (S_r) as a weak immersion along with H_r is actually sufficient to bound slim tree-cut width (0-tree-cut width).

► **Theorem 15.** *If G is a graph such that $H_{2r,2} \leq_I G$ for some $r \geq 3$ or $S_{r,2} \leq_I G$ for some $r \geq 1$, then $\text{tcw}_0(G) \geq r$. Moreover, there exists a function $h : \mathbb{N} \rightarrow \mathbb{N}$ such that if $\text{tcw}_0(G) \geq h(r)$, then $H_r \leq_I G$ or $S_r \leq_I G$.*

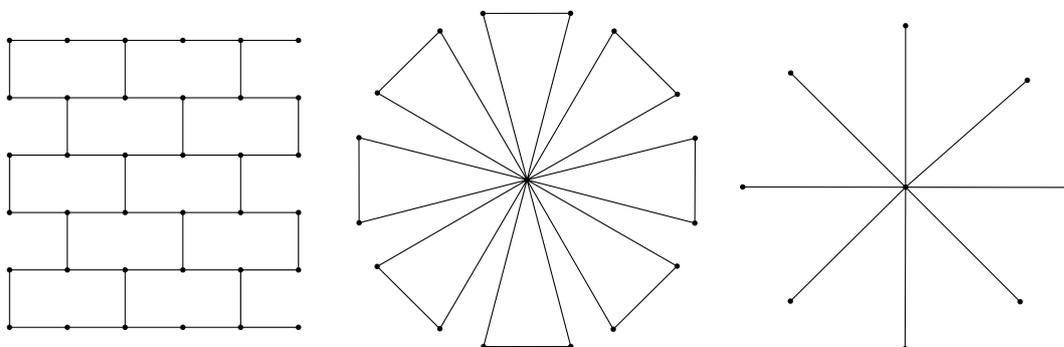
Proof. If $H_{2r,2} \leq_I G$ for some $r \geq 3$, we have that $\text{tcw}(G) \geq r$ by Theorem 13 and hence $\text{tcw}_0(G) \geq r$. In case $S_{r,2} \leq_I G$, the lower bound follows from Lemma 14.

Let f be the function given by Theorem 13. We define h by setting $h(r) = r \cdot f(r) + 3 \cdot f(r) + 2$. Assume that G is a graph such that $\text{tcw}_0(G) \geq h(r)$. If $\text{tcw}(G) \geq f(r)$, we immediately conclude that $H_r \leq_I G$ by Theorem 13. Otherwise, let (T, \mathcal{X}) be a nice tree-cut decomposition of G of width at most $f(r)$ with leaves having non-empty bags. There exists a node t of T such that $\text{tor}_1(t) \geq h(r)$, in particular, $B_t \geq r \cdot f(r)$. As the size of X_t is at most $f(r)$, some vertex of X_t has degree of at least r and hence $S_r \leq_I G$. ◀

Before providing similar characterization for slim tree-cut width, we introduce a simple technical modification of tree-cut decompositions, which will also be used later for establishing the connection between slim tree-cut width and edge-cut width. The aim is, roughly speaking, to avoid the situation where a thin child has adhesion 2, even though it consists of two completely independent components each of which could be a thin child of adhesion 1. Formally, let (T, \mathcal{X}) be a nice tree-cut decomposition of G . We say that a node t with parent t' in T is *decomposable* if the following conditions hold:

- $t \in B_{t'}$ and there exist two edges e_1 and e_2 between G_t and $G \setminus G_t$ in G ;
- the endpoints of e_1 and e_2 in G_t belong to different connected components of G_t .

► **Lemma 16.** *Any nice tree-cut decomposition of G can be transformed into a nice tree-cut decomposition of the same tree-cut width with no decomposable nodes.*



■ **Figure 3** Illustrations of forbidden weak immersions for the graphs with bounded standard, slim or 0-tree-cut width. Left: 6-wall H_6 , Middle: windmill W_8 , Right: star S_8 .

Proof. Let (T', \mathcal{X}') be a nice tree-cut decomposition of G with at least one decomposable node. Let t be a decomposable node of T' with minimum distance to the root, and let e_1 and e_2 be the edges between G_t and $G \setminus G_t$ in G . We create a copy T'_* of the rooted subtree T'_t where the copy of $s \in T'_t$ is $s^* \in T'_{t^*}$. We then connect t^* to the parent of t . Let G_1 be the connected component of G_t containing an endpoint of e_1 . For every $s \in V(T'_t)$ we set $X_s = X'_s \cap V(G_1)$ and $X_{s^*} = X'_s \setminus X_s$. For the rest of nodes s of T' we set $X_s = X'_s$. Finally, we exhaustively remove empty bags which are leaves and denote the obtained tree by T . Observe that the resulting decomposition (T, \mathcal{X}) is nice and its width is not greater than the width of (T', \mathcal{X}') . Moreover, our transformation doesn't create any decomposable nodes outside of subtrees rooted in t and t^* ; both t and t^* have an adhesion of one and hence are not decomposable. Therefore, after a finite number of such steps we obtain some nice tree-cut decomposition of G of the same width but with no decomposable nodes. ◀

Further, as a technical term, we will refer to nice decompositions with no decomposable nodes as *very nice* decompositions.

► **Corollary 17.** *Every tree-cut decomposition can be transformed into a very nice tree-cut decomposition in quartic time, without increasing the width.*

Proof. Let (T'', \mathcal{X}'') be a tree-cut decomposition of G of width k . We transform (T'', \mathcal{X}'') into a nice tree-cut decomposition (T', \mathcal{X}') of width at most k (this can be done in cubic time, see [16] for details). Further, we apply Lemma 16 on (T', \mathcal{X}') . This requires at most quartic time, since every node of T' is decomposed at most once and every such decomposition can be performed in cubic time. Then the resulting decomposition (T, \mathcal{X}) is very nice and has width of at most k . ◀

With this transformation in hand, we are now ready to fully characterize forbidden weak immersions for graphs of bounded slim tree-cut width.

► **Theorem 18.** *If G is a graph such that $H_{2r,2} \leq_I G$ for some $r \geq 3$ or $W_{r,2} \leq_I G$ for some $r \geq 1$, then $\text{stcw}(G) \geq r$. Moreover, there exists a function $g : \mathbb{N} \rightarrow \mathbb{N}$ such that if $\text{stcw}(G) \geq g(r)$, then $H_r \leq_I G$ or $W_r \leq_I G$.*

Proof. If $H_{2r,2} \leq_I G$ for some $r \geq 3$, we have that $\text{tcw}_2(G) \geq r$ by Theorem 13 and hence $\text{stcw}(G) \geq r$. In case $W_{r,2} \leq_I G$, the lower bound follows from Lemma 14.

Let f be the function given by Theorem 13. We define g by setting $g(r) = 2r \cdot f^2(r) + 3 \cdot f(r) + 2$. Assume that G is a graph such that $\text{stcw}(G) \geq g(r)$. If $\text{tcw}(G) \geq f(r)$, we immediately conclude that $H_r \leq_I G$ by Theorem 13. Otherwise, by Corollary 17 there exists a very nice tree-cut decomposition (T, \mathcal{X}) of G of width at most $f(r)$. Let us pick a node t of T such that $\text{tor}_2(t) \geq g(r)$. By Corollary 11 we have that $|B_t^{(2)}| \geq g(r) - 3 \cdot f(r) - 2 = 2r \cdot f^2(r)$. Since (T, \mathcal{X}) is very nice, all the children of t in $B_t^{(2)}$ are non-decomposable. Recall that for every $t' \in B_t^{(2)}$, the neighbourhood of $Y_{t'}$ in G is a one- or two-element subset of X_t , and hence $Y_{t'}$ provides a path between some (possibly equal) vertices of X_t . As the size of X_t is at most $f(r)$, G contains either $2r$ cycles intersecting in one vertex of X_t or $2r$ paths between two vertices of X_t . Since every such pair of paths can be transformed into a cycle by lifting the pair of their first edges, in both cases we have $W_r \leq_I G$. ◀

3.3 k -Edge Sums

Another natural property Wollan [37] established for tree-cut width is that the parameter is closed under the operation of taking k -edge sum for small k . Specifically, he proved:

► **Lemma 19** ([37]). *Let G , G_1 , and G_2 be graphs such that $G = G_1 \oplus_k G_2$. If G_j has a tree-cut decomposition (T_j, \mathcal{X}_j) for $j = 1, 2$, then G has a tree-cut decomposition (T, \mathcal{X}) such that $\text{adh}(T, \mathcal{X}) = \max\{k, \text{adh}(T_1, \mathcal{X}_1), \text{adh}(T_2, \mathcal{X}_2)\}$. Moreover, for every $t \in V(T)$, the torso H_t of t in (T, \mathcal{X}) is isomorphic to the torso of some vertex of (T_1, \mathcal{X}_1) or (T_2, \mathcal{X}_2) .*

Based on this result for optimal decompositions (T_1, \mathcal{X}_1) and (T_2, \mathcal{X}_2) , we immediately obtain the upper bound on 0- and slim tree-cut width for k -edge sums:

► **Corollary 20.** *Let G , G_1 and G_2 be graphs such that $G = G_1 \oplus_k G_2$. Then it holds that $\text{stcw}(G) \leq \max\{k, \text{stcw}(G_1), \text{stcw}(G_2)\}$ and $\text{tcw}_0(G) \leq \max\{k, \text{tcw}_0(G_1), \text{tcw}_0(G_2)\}$.*

In particular, if both G_1 and G_2 have 0-, slim or standard width of at most ω and $k \leq \omega$, we may conclude that the corresponding width of G is at most ω .

4 Alternative Characterizations

In this section, we study alternative characterizations of slim tree-cut width and 0-tree-cut width. In particular, we observe that the latter is asymptotically equivalent to maximum degree plus treewidth. This provides an interesting connection between tree decompositions and tree-cut decompositions, but essentially rules out its study as a means of establishing novel tractability results. For slim tree-cut width, however, we obtain a characterization that ties it to the previously studied edge-cut width and has algorithmic implications.

4.1 Characterization of 0-Tree-Cut Width

Wollan [37] showed that a bound on the treewidth and maximum degree implies a bound on the tree-cut width of a graph:

► **Proposition 21.** *Let G be a graph with maximal degree d and treewidth w . Then there exists a tree-cut decomposition of adhesion at most $(2w + 2)d$ such that every torso has at most $(d + 1)(w + 1)$ vertices.*

In particular, as $\text{tor}_1(t) \leq |H_t| \leq (d + 1)(w + 1) \leq (2w + 2)d$ for every node t of T , we have $\text{tcw}_0(G) \leq (2w + 2)d$. In the following proposition, we show that the converse is true as well: bounded tcw_0 implies bounded treewidth and maximum degree of a graph.

► **Proposition 22.** *Let G be a graph with $\text{tcw}_0(G) = k$. Then every vertex of G has degree of at most $k^2 + 2k$ and $\text{tw}(G) \leq 2k^2 + 3k$.*

Proof. By Lemma 3 and Corollary 8 we have that $\text{tw}(G) \leq 2 \text{tcw}_0(G)^2 + 3 \text{tcw}_0(G) \leq 2k^2 + 3k$. Since $\text{tcw}_0(G) = k$, Lemma 9 implies that G does not contain $S_{(k+1)^2}$ as a weak immersion, in particular, degree of any vertex of G is at most $k^2 + 2k$. ◀

► **Corollary 23.** *0-tree-cut width is asymptotically equivalent to maximum degree plus treewidth.*

4.2 Characterization of Slim Tree-Cut Width

Recall that edge-cut width is a parameter that is defined over spanning trees in the input graph G , which serve as the corresponding decompositions. Let us now consider a slight generalization of this where we consider not only spanning trees over G , but of any supergraph

15:12 Slim Tree-Cut Width

of G . Such a generalization would – unlike edge-cut width itself – trivially be closed under both vertex and edge deletion. For our considerations, let us denote this parameter *super edge-cut width* ($\text{sec}(G)$):

$$\text{sec}(G) = \min\{\text{ecw}(H, T) \mid H \supseteq G \text{ and } T \text{ is a spanning forest of } H\}.$$

If $H \supseteq G$ is a supergraph of G and T is a spanning forest of H such that $\text{ecw}(H, T) \leq k$, we say that T *witnesses* $\text{sec}(G) \leq k$. Observe that there always exists a connected witness, i.e., a tree. Indeed, if H consists of $m > 1$ connected components, we can arbitrarily extend it to a connected graph H^* by adding $m - 1$ edges. The addition of these edges to T then results in the tree T^* witnessing $\text{sec}(G) \leq k$. Moreover, notice that any witness of $\text{ecw}(G) \leq k$ is also a witness of $\text{sec}(G) \leq k$.

► **Corollary 24.** *For every graph G , $\text{sec}(G) \leq \text{ecw}(G)$.*

However, graphs of constant super edge-cut width can have arbitrarily large edge-cut width, as will become clear at the end of the section. A slight modification of the proof of Proposition 5 yields:

► **Proposition 25.** *For every graph G , $\text{tcw}(G) \leq \text{sec}(G)$.*

Proof. Let Q be the supergraph of G and let T be the spanning tree of Q such that $\text{ecw}(Q, T) = \text{sec}(G)$. We construct a tree-cut decomposition (T, \mathcal{X}) of G where each bag contains at most one vertex, notably by setting $X_t = \{t\}$ for each $t \in V(G)$ and $X_t = \emptyset$ for each $t \in V(Q) \setminus V(G)$. Fix any node t in T other than the root, let u be the parent of t in T . All the edges of $G \setminus ut$ with one endpoint in the rooted subtree T_t and another outside of T_t belong to $E_{loc}^{Q, T}(t)$, so $\text{adh}_T(t) \leq |E_{loc}^{Q, T}(t)| + 1 \leq \text{sec}(G)$.

Let H_t be the torso of (T, \mathcal{X}) in t , then $V(H_t) = X_t \cup \{z_1 \dots z_l\}$ where z_i correspond to connected components of $T \setminus t$, $i \in [l]$. In \tilde{H}_t , only z_i with degree at least 3 are preserved. But all such z_i are the endpoints of at least two edges in $|E_{loc}^{Q, T}(t)|$, so $\text{tor}(t) = |V(\tilde{H}_t)| \leq 1 + |E_{loc}^{Q, T}(t)| \leq \text{sec}(G)$. Thus $\text{tcw}(G) \leq \text{sec}(G)$. ◀

To represent a deeper connection between tree-cut decompositions and super edge-cut width, it will be convenient to work with very nice decompositions introduced in subsection 3.2.

► **Proposition 26.** *Let (T, \mathcal{X}) be a very nice tree-cut decomposition of G of width at most k . Then for each node t of T , $|B_t^{(2)}| \leq k \cdot \text{sec}(G)$. In particular, $\text{stcw}(G) \leq \text{sec}(G)^2 + 4 \cdot \text{sec}(G)$.*

Proof. Assume that T^* is a spanning tree of $H \supseteq G$ such that $\text{sec}(G) = \text{ecw}(H, T^*)$. For any node t of T and $b \in B_t^{(2)}$, b has one of three types (see Figure 4):

1. $N(Y_b) = \{x\}$ for some $x \in X_t$, x is connected to distinct x_b^1 and x_b^2 from Y_b ;
2. $N(Y_b) = \{x_1, x_2\}$ for $x_1 \neq x_2$, x_1 and x_2 are connected to the same $x_b \in Y_b$;
3. $N(Y_b) = \{x_1, x_2\}$ for $x_1 \neq x_2$, x_1 and x_2 are connected to distinct x_b^1 and x_b^2 from Y_b correspondingly;

Let us start with the first type. If $x_b^i x$ doesn't belong to T^* for $i = 1$ or $i = 2$, then $x_b^i x \in E_{loc}^{H, T^*}(x)$. Otherwise, x_b^1 and x_b^2 are connected via x in T^* . Then $T^*[Y_b]$ has precisely two connected components. As b is not decomposable, there exists a path p between x_b^1 and x_b^2 in G_b containing precisely one edge outside of T^* . This edge contributes to $E_{loc}^{H, T^*}(x)$.

As T^* is a tree, there can be at most $|X_t| - 1 \leq k - 1$ thin children b of the second type such that x_b is adjacent to two elements of X_t in T^* . For the rest of b of the second type, there exists $x \in X_t$ such that $xx_b \in G \setminus T^* \subseteq H \setminus T^*$ and therefore $xx_b \in E_{loc}^{H, T^*}(x)$.

Let b be a thin node of the third type. If x_1^b and x_2^b are connected via a path in $T^*[Y_b]$, we can apply the same argument as for the second type. Otherwise, $T^*[Y_b]$ has precisely two connected components and, analogously to the first type, there exists an edge in $G_b \cup \{x_1x_1^b, x_2x_2^b\}$ that belongs to $E_{loc}^{H, T^*}(x_1)$.

To conclude, any node of $B_t^{(2)}$ either increases $E_{loc}^{H, T^*}(x)$ for some $x \in X_t$ or creates a path in T^* between two vertices of X_t . Since T^* is a tree, $|X_t| \leq k$ and $|E_{loc}^{H, T^*}(x)| \leq \text{sec}(G) - 1$ for every $x \in X_t$, the size of $B_t^{(2)}$ is at most $(k - 1) + \sum_{x \in X_t} |E_{loc}^{H, T^*}(x)| \leq k \cdot \text{sec}(G) - 1$. Then $\text{tor}_2(t) \leq |A_t| + |X_t| + 1 + |B_t^{(2)}| \leq 3k + 1 + k \cdot \text{sec}(G) \leq k \cdot (\text{sec}(G) + 4)$. Since the bound holds for every node t of T , we may conclude that the slim width of (T, \mathcal{X}) is at most $k \cdot (\text{sec}(G) + 4)$. By Proposition 25 and Corollary 17, there exists a very nice tree-cut decomposition of G of width $k \leq \text{sec}(G)$, therefore $\text{stcw}(G) \leq \text{sec}(G)^2 + 4 \cdot \text{sec}(G)$. ◀

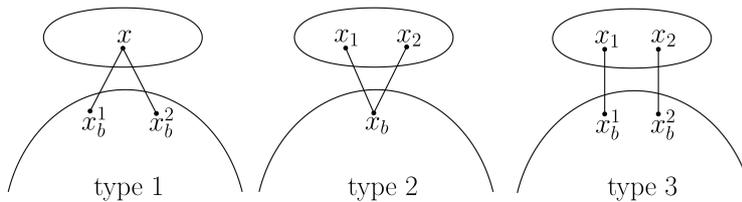
Hence, slim tree-cut width of any graph is upper-bounded by a quadratic function of its super edge-cut width. Next, we show that the converse statement holds as well:

► **Proposition 27.** *For every graph G , $\text{sec}(G) \leq 3 \cdot (\text{stcw}(G) + 1)^2$. Moreover, given a tree-cut decomposition of G of slim width k , it is possible to compute a supergraph $Q \supseteq G$ and its spanning tree T witnessing $\text{sec}(G) \leq 3(k + 1)^2$ in cubic time.*

Proof. Let (T_0, \mathcal{X}_0) be a tree-cut decomposition of G of slim width k . We start by transforming it into a nice tree-cut decomposition (T, \mathcal{X}) in cubic time as in [16]. The transformation procedure acts on the 2-centers of torsos only by contracting some edges. Recall that $v \in V(H_t) \setminus X_t$ is not in the 2-center of (H_t, X_t) if and only if v belongs to some induced subtree of H_t connected to the rest of H_t by at most one edge. Since contracting an edge either preserves the property or merges v with some other vertex, it doesn't increase $\text{tor}_2(t)$ for any node t of T . In particular, the slim width of (T, \mathcal{X}) is at most k .

Let $\Omega \subseteq \mathcal{X}$ be the set of empty bags of (T, \mathcal{X}) , we construct $Q \supseteq G$ along with its tree-cut decomposition (T, \mathcal{X}') as follows. Firstly, we add to G vertices v_t for every $t \in \Omega$. We define $X'_t = \{v_t\}$ if $X_t = \emptyset$ and $X'_t = X_t$ otherwise. For every node $t \in T$, construct an arbitrary tree T_t^* over X'_t and add its edges to Q . Further, we process every edge $e = pt \in E(T)$ such that p is the parent of t in T and either $N(Y_t) \not\subseteq X_t$ or $\text{adh}(t) > 1$ as follows. If G doesn't contain an edge between X'_t and X'_p , we add to $E(Q)$ arbitrary edge with endpoints in X'_t and X'_p . This increases the adhesion of e by at most one.

Now we proceed to the choice of the spanning tree T^* in Q . For every $t \in T$ other than the root, let p be the parent of t in T . If $\text{adh}(t) = 1$ and $N(Y_t) \subseteq X_t$, we denote by e_t the unique edge between Y'_t and X'_p in Q . Otherwise, let e_t be arbitrary edge of Q with endpoints in X'_t and X'_p . We then construct T^* by gluing together all T_t^* via edges e_t : $T^* = (\cup_{t \in V(T)} T_t^*) \cup (\cup_{t \in V(T) \setminus r} \{e_t\})$. Obviously the construction can be performed in cubic time; we will show that $\text{sec}(Q, T^*) \leq 3(k + 1)^2$.



■ **Figure 4** Possible configurations of edges between thin child $b \in B_t^{(2)}$ and its parent t .

To this end, fix any node t of T and $x \in X'_t$ and denote $E_{loc}(x) = E_{loc}^{Q, T^*}(x)$. If T^* contains more than one edge between Y'_t and rest of T^* , then all but one of them are the unique edges connecting Q'_q to the rest of Q for some descendants q of t in T . Hence, they don't belong to any path in T^* between the endpoints of some feedback edge $e \in E(Q) \setminus E(T^*)$. Therefore, every edge of $E_{loc}(x)$ has at least one endpoint in Y'_t . The number of edges in $E_{loc}(x)$ with both endpoints in X'_t is at most $|X'_t| \cdot (|X'_t| - 1) \leq k \cdot (k - 1)$. Every edge with one endpoint in X'_t and another outside of Y'_t contributes to the adhesion of t in (T, \mathcal{X}') , so their number is bounded by $k + 1$.

Finally, if $e = yz \in E_{loc}(x)$ contains an endpoint y in $Y'_t \setminus X'_t$, then $y \in Y'_q$ for some child q of t . Then Q contains a cycle intersecting Y'_q and $x \in X_t$. In particular, by construction of Q we may conclude $q \in A_t \cup B_t^{(2)}$ w.r.t. the decomposition (T, \mathcal{X}) . By the same arguments as for the node t , we conclude that at most one edge between Y'_q and the rest of T^* belongs to any path in T^* between the endpoints of some feedback edge $e \in E(Q) \setminus E(T^*)$, so $z \notin Y'_q$ and e contributes to the adhesion of q in (T, \mathcal{X}') . In particular, $E_{loc}(x)$ contains at most $\text{adh}(q) + 1$ edges with an endpoint in Y'_q . In total, at most $\max_{q \in A_t} (\text{adh}(q) + 1) \cdot |A_t| + \max_{q \in B_t^{(2)}} (\text{adh}(q) + 1) \cdot |B_t^{(2)}| \leq (k + 1)(2k + 1) + 3k = 2k^2 + 6k + 1$ edges in $E_{loc}(x)$ have an endpoint in $Y'_t \setminus X'_t$, so $|E_{loc}(x)| \leq k \cdot (k - 1) + (k + 1) + 2k^2 + 6k + 1 = 3k^2 + 6k + 2$ and hence $\text{sec}(Q, T^*) \leq 3k^2 + 6k + 3 = 3(k + 1)^2$. ◀

► **Corollary 28.** *sec and stcw are asymptotically equivalent.*

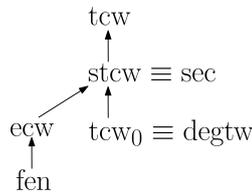
The results of this section are summarized in Figure 5. In particular, the graph family provided in [4, Lemma 2] shows that graphs of constant super edge-cut width may have arbitrarily large edge-cut width.

5 Approximating Slim Tree-Cut Width

In this section we show how to efficiently construct a tree-cut decomposition of a graph G with slim width bounded by a cubic function of its optimal value $\text{stcw}(G)$. As a starting point for our approximation, we use the following result of Kim, Oum, Paul, Sau and Thilikos:

► **Theorem 29** ([28]). *There exists an algorithm that, given a graph G and $\omega \in \mathbb{N}$, either outputs a tree-cut decomposition of G with width at most 2ω or correctly reports that no tree-cut decomposition of G with width at most ω exists in $2^{\mathcal{O}(\omega^2 \cdot \log \omega)} \cdot n^2$ steps.*

As an observant reader might have already noticed, if G has bounded slim tree-cut width, it imposes some restrictions on the structure of possible decompositions of G of small (standard) tree-cut width. This fact enables us to construct an efficient approximation for $\text{stcw}(G)$.



■ **Figure 5** Position of slim and 0-tree-cut width in the hierarchy of edge-cut based parameters. An arrow from p to q represents the fact that p is more restrictive than q , while asymptotic equivalence is depicted by \equiv .

► **Theorem 30.** *There exists an algorithm that, given a graph G and $\omega \in \mathbb{N}$, either outputs a tree-cut decomposition of G with slim width at most $6(\omega + 1)^3$ or correctly reports that no tree-cut decomposition of G with slim width at most ω exists in $2^{\mathcal{O}(\omega^2 \cdot \log \omega)} \cdot n^4$ steps.*

Proof. Given a graph G and $\omega \in \mathbb{N}$, let us run the algorithm from Theorem 29. If it reports that $\text{tcw}(G) > \omega$, we may conclude that $\text{stcw}(G) > \omega$ by Corollary 8. In case the algorithm returns a tree-cut decomposition (T', \mathcal{X}') of width at most 2ω , we invoke Corollary 17 to transform this decomposition into a very nice decomposition (T, \mathcal{X}) of the same width in at most quartic time. By Proposition 26, we have that $|B_t^{(2)}| \leq 2\omega \cdot \text{sec}(G)$ for each node t of T . If for some node t the size of $B_t^{(2)}$ exceeds $6\omega \cdot (\omega + 1)^2$, then $\text{sec}(G) > 3(\omega + 1)^2$ and by Proposition 27 we may correctly report that $\text{stcw}(G) > \omega$. Otherwise, $\text{tor}_2(t) \leq 1 + |X_t| + |A_t| + |B_t^{(2)}| \leq 1 + 2\omega + (4\omega + 1) + 6\omega \cdot (\omega + 1)^2 \leq 6(\omega + 1)^3$ for any node t of T . Hence, the slim width of (T, \mathcal{X}) is at most $6(\omega + 1)^3$. ◀

6 Discussion of Algorithmic Applications

Having established its structural properties, we now turn to the algorithmic aspects of slim tree-cut width. Here, Corollary 28 shows that instead of using a tree-cut decomposition of the input graph G to design fixed-parameter algorithms – as was done in past dynamic programming algorithms that utilized tree-cut width – we can perform dynamic programming along a spanning tree T of a supergraph Q of G . Both Q and T can be computed from G in a pre-processing stage by using Proposition 27, and using a spanning tree instead of a tree-cut decomposition typically leads to significantly more concise (and conceptually cleaner) algorithms.

The cost for this simplification is the quadratic gap between the widths of these decompositions. We note that this situation is somewhat analogous to how one still typically uses clique-width [7] as a general and easy-to-use parameterization for various problems (especially when aiming for instances with higher edge-densities), even though rank-width [34] and Boolean-width [6] are asymptotically equivalent parameterizations which have been shown to yield more efficient algorithms [15] – there, the gap is even exponential.

Recall that a number of problems which remain $W[1]$ -hard w.r.t. tree-cut width have recently been shown to be fixed-parameter tractable when parameterized by edge-cut width [4, 18], via explicit dynamic programming algorithms which proceed along the spanning tree of the input graph. While the functional gap between edge-cut width and super edge-cut width (and, analogously, slim tree-cut width) may be arbitrarily large, it is not difficult to see that each of the algorithms provided in those papers can be straightforwardly lifted to fixed-parameter algorithms w.r.t. super edge-cut width. Indeed, the only amendment one needs to make is to deal with the presence of “ghost” edges and vertices which occur in the spanning tree but not in the graph, and the computation of the records in these algorithms can easily deal with such vertices and edges.

To provide a concrete illustration of how this can be done, let us revisit the dynamic programming algorithm for the EDGE DISJOINT PATHS problem parameterized by edge-cut width [4, Theorem 2]. No change is needed to the records. When the algorithm attempts to compute the set of “valid records” for a vertex v from the sets of valid records for some of its children v_1, \dots, v_ψ in the spanning tree, the algorithm performs a branching step in which it considers all possible ways the paths can be routed between the subtrees rooted at these children (See the “If v is an internal node” paragraph in the proof). At this branching step, we simply discard all routings which use edges that are not present in G . The situation is no more complicated for the other considered problems – in essentially all cases, the change simply boils down to ignoring the vertices and edges which do not exist in G .

Hence, we obtain:

► **Corollary 31** (Theorems 2-6 in [4], Theorems 6 and 14 in [18]). *LIST COLORING, PRECOLORING EXTENSION, BOOLEAN CONSTRAINT SATISFACTION, EDGE DISJOINT PATHS, BAYESIAN NETWORK STRUCTURE LEARNING, POLYTREE LEARNING, MINIMUM CHANGEOVER COST ARBORESCENCE, and MAXIMUM STABLE ROOMMATES WITH TIES AND INCOMPLETE LISTS are fixed-parameter tractable w.r.t. slim tree-cut width.*

Last but not least, given the ease with transferring dynamic programming algorithms from edge-cut width to slim tree-cut width, an inquisitive reader might be wondering whether it is not possible to formally prove that *every* problem which is FPT w.r.t. former is also FPT w.r.t. the latter. That is, however, not true in general: one can construct entirely artificial problems which do not behave in this way.

To illustrate this on a high level, let us consider an arbitrary graph problem P which remains NP-hard even on trees (as an example, the FIREFIGHTER problem [12]) and can be solved on general n -vertex graphs in time $\tau(n)$. Moreover, let $\iota(n)$ denote the time required to compute the slim tree-cut width of a graph G via an exhaustive brute force search, and let ψ be a function which dominates both τ and ι . We now define an artificial new problem P' as follows:

- every n -vertex graph G such that $\psi(\text{ecw}(G)) \leq n$ is a YES-instance, and otherwise
- G is a YES-instance if and only if G is a YES-instance of FIREFIGHTER.

Then P' is FPT parameterized by edge-cut width. Indeed, given an instance (G, k) of P' , one can attempt to run a brute-force search to determine the edge-cut width (which is promised to be at most k) with a time-out of $\psi(\psi(k))$. If the algorithm times out, this implies that $\psi(\text{ecw}(G)) \leq n$ and we correctly output “Yes”. If not, we proceed by calling a brute-force algorithm to solve FIREFIGHTER on G , and this must once again complete in time at most $\psi(\psi(k))$. On the other hand, P' remains NP-hard even on graph classes with constant $\text{stcw}(G)$ – consider, for instance, the class of all graphs with two connected components, one of which (C_1) is a tree and the other (C_2) a graph from the class with constant slim tree-cut width but unbounded edge-cut width (one such class is depicted in Figure 2 of [4]). On some inputs from this class, P' will ask for a solution to the FIREFIGHTER problem (which is NP-hard on trees) but the parameter $\text{stcw}(G)$ will remain constant.

7 Conclusion

The contribution of this work is mainly conceptual: it provides a possible resolution to the search for an alternative to treewidth for edge cuts which is both structurally sound and exhibits the expected (and desired) algorithmic properties. Slim tree-cut width can be viewed as the “missing link” which explains why the problems depicted in Table 1 admit fixed-parameter algorithms that exploit dynamic programming along small edge cuts w.r.t. both edge-cut width (as a generalization of the feedback edge number) and treewidth plus maximum degree. We firmly believe that there are many more problems of interest where edge-cut based parameters may help push the frontiers of tractability. On this front, the alternative characterization via the edge-cut width of a supergraph provides decompositions which are better suited for dynamic programming than tree-cut decompositions.

The problem of computing optimal decompositions for slim tree-cut width remains, similarly as in the case of tree-cut width [28], as a prominent open question. Moreover, we believe that the ideas used to obtain a 2-approximation algorithm for tree-cut width could also be used to obtain an improved constant-factor approximation for slim tree-cut width.

References

- 1 Deeksha Adil, Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. Parameterized algorithms for stable matching with ties and incomplete lists. *Theor. Comput. Sci.*, 723:1–10, 2018. doi:10.1016/j.tcs.2018.03.015.
- 2 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. doi:10.1007/s00453-020-00777-5.
- 3 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Preprocessing for treewidth: A combinatorial analysis through kernelization. *SIAM J. Discret. Math.*, 27(4):2108–2142, 2013.
- 4 Cornelius Brand, Esra Ceylan, Christian Hatschka, Robert Ganian, and Viktoriia Korchemna. Edge-cut width: An algorithmically driven analogue of treewidth based on edge cuts. In *Graph-Theoretic Concepts in Computer Science – 48th International Workshop, WG 2022*, Lecture Notes in Computer Science. Springer, 2022. to appear. arXiv:2202.13661.
- 5 Robert Brederick, Klaus Heeger, Dusan Knop, and Rolf Niedermeier. Parameterized complexity of stable roommates with ties and incomplete lists through the lens of graph parameters. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPICs*, pages 44:1–44:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
- 6 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Boolean-width of graphs. *Theor. Comput. Sci.*, 412(39):5187–5204, 2011. doi:10.1016/j.tcs.2011.05.022.
- 7 B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.
- 8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 9 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 10 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer Verlag, 2013. doi:10.1007/978-1-4471-5559-1.
- 11 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In *ISAAC*, Lecture Notes in Computer Science, pages 294–305. Springer, 2008.
- 12 Stephen Finbow, Andrew D. King, Gary MacGillivray, and Romeo Rizzi. The firefighter problem for graphs of maximum degree three. *Discret. Math.*, 307(16):2094–2105, 2007. doi:10.1016/j.disc.2005.12.053.
- 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.
- 14 Robert Ganian. Improving vertex cover as a graph parameter. *Discret. Math. Theor. Comput. Sci.*, 17(2):77–100, 2015. URL: <http://dmtcs.episciences.org/2136>.
- 15 Robert Ganian and Petr Hliněný. On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. *Discr. Appl. Math.*, 158(7):851–867, 2010.
- 16 Robert Ganian, Eun Jung Kim, and Stefan Szeider. Algorithmic applications of tree-cut width. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science 2015 – 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, volume 9235 of *Lecture Notes in Computer Science*, pages 348–360. Springer, 2015. to appear in the Siam Journal on Discrete Mathematics. arXiv:2206.00752.
- 17 Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On structural parameterizations of the bounded-degree vertex deletion problem. *Algorithmica*, 83(1):297–336, 2021.
- 18 Robert Ganian and Viktoriia Korchemna. The complexity of bayesian network learning: Revisiting the superstructure. In *Proceedings of NeurIPS 2021, the Thirty-fifth Conference on Neural Information Processing Systems*, 2021. to appear.

- 19 Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. *Artif. Intell.*, 257:61–71, 2018.
- 20 Robert Ganian and Sebastian Ordyniak. The power of cut-based parameters for computing edge-disjoint paths. *Algorithmica*, 83(2):726–752, 2021.
- 21 Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. On structural parameterizations of the edge disjoint paths problem. *Algorithmica*, 83(6):1605–1637, 2021.
- 22 Archontia C. Giannopoulou, O-joung Kwon, Jean-Florent Raymond, and Dimitrios M. Thilikos. Lean tree-cut decompositions: Obstructions and algorithms. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPICs*, pages 32:1–32:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
- 23 Archontia C. Giannopoulou, O-joung Kwon, Jean-Florent Raymond, and Dimitrios M. Thilikos. A menger-like property of tree-cut width. *J. Comb. Theory, Ser. B*, 148:1–22, 2021. doi:10.1016/j.jctb.2020.12.005.
- 24 Didem Gözüpek, Sibel Özkan, Christophe Paul, Ignasi Sau, and Mordechai Shalom. Parameterized complexity of the MINCCA problem on graphs of bounded decomposability. *Theor. Comput. Sci.*, 690:91–103, 2017.
- 25 Didem Gözüpek, Hadas Shachnai, Mordechai Shalom, and Shmuel Zaks. Constructing minimum changeover cost arborescences in bounded treewidth graphs. *Theor. Comput. Sci.*, 621:22–36, 2016. doi:10.1016/j.tcs.2016.01.022.
- 26 Gregory Z. Gutin, Mark Jones, and Magnus Wahlström. The mixed chinese postman problem parameterized by pathwidth and treedepth. *SIAM J. Discret. Math.*, 30(4):2177–2205, 2016.
- 27 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.
- 28 Eun Jung Kim, Sang-il Oum, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. An FPT 2-approximation for tree-cut decomposition. *Algorithmica*, 80(1):116–135, 2018.
- 29 Loïc Magne, Christophe Paul, Abhijat Sharma, and Dimitrios M. Thilikos. Edge-treewidth: Algorithmic and combinatorial properties. *CoRR*, abs/2112.07524, 2021. arXiv:2112.07524.
- 30 Dániel Marx and Paul Wollan. Immersions in highly edge connected graphs. *SIAM J. Discrete Math.*, 28(1):503–520, 2014.
- 31 Jesper Nederlof, Michal Pilipczuk, Céline M. F. Swennenhuis, and Karol Wegrzycki. Hamiltonian cycle parameterized by treedepth in single exponential time and polynomial space. In Isolde Adler and Haiko Müller, editors, *Graph-Theoretic Concepts in Computer Science – 46th International Workshop, WG 2020, Leeds, UK, June 24-26, 2020, Revised Selected Papers*, volume 12301 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2020.
- 32 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.
- 33 Sebastian Ordyniak and Stefan Szeider. Parameterized complexity results for exact bayesian network structure learning. *J. Artif. Intell. Res.*, 46:263–302, 2013. doi:10.1613/jair.3744.
- 34 Sang-il Oum. Approximating rank-width and clique-width quickly. In *Graph-Theoretic Concepts in Computer Science, 31st International Workshop, WG 2005, Metz, France, June 23-25, 2005, Revised Selected Papers*, volume 3787 of *Lecture Notes in Computer Science*, pages 49–58. Springer Verlag, 2005.
- 35 Neil Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.
- 36 Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *J. of Computer and System Sciences*, 76(2):103–114, 2010.
- 37 Paul Wollan. The structure of graphs not admitting a fixed immersion. *J. Comb. Theory, Ser. B*, 110:47–66, 2015.

A Fixed-Parameter Algorithm for the Schrijver Problem

Ishay Haviv

School of Computer Science, The Academic College of Tel Aviv-Yaffo, Israel

Abstract

The Schrijver graph $S(n, k)$ is defined for integers n and k with $n \geq 2k$ as the graph whose vertices are all the k -subsets of $\{1, 2, \dots, n\}$ that do not include two consecutive elements modulo n , where two such sets are adjacent if they are disjoint. A result of Schrijver asserts that the chromatic number of $S(n, k)$ is $n - 2k + 2$ (Nieuw Arch. Wiskd., 1978). In the computational SCHRIJVER problem, we are given an access to a coloring of the vertices of $S(n, k)$ with $n - 2k + 1$ colors, and the goal is to find a monochromatic edge. The SCHRIJVER problem is known to be complete in the complexity class PPA. We prove that it can be solved by a randomized algorithm with running time $n^{O(1)} \cdot k^{O(k)}$, hence it is fixed-parameter tractable with respect to the parameter k .

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph coloring; Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Probabilistic algorithms

Keywords and phrases Schrijver graph, Kneser graph, Fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.16

Related Version *Full Version:* <http://arxiv.org/abs/2204.09009>

Funding Research supported in part by the Israel Science Foundation (grant No. 1218/20).

Acknowledgements We thank Gabriel Istrate for clarifications on [18] and the anonymous reviewers for their useful suggestions.

1 Introduction

The Kneser graph $K(n, k)$ is defined for integers n and k with $n \geq 2k$ as the graph whose vertices are all the k -subsets of $[n] = \{1, 2, \dots, n\}$ where two such sets are adjacent if they are disjoint. In 1955, Kneser [19] observed that the chromatic number of the graph $K(n, k)$ satisfies $\chi(K(n, k)) \leq n - 2k + 2$, that is, there exists a proper coloring of its vertices with $n - 2k + 2$ colors, and conjectured that this upper bound on the chromatic number is tight. The conjecture was proved in 1978 by Lovász [20] as an application of the Borsuk-Ulam theorem from algebraic topology [2]. Following this result, topological methods have become a powerful tool in combinatorics, discrete geometry, and theoretical computer science (see, e.g., [22]).

The Schrijver graph $S(n, k)$ is defined as the subgraph of $K(n, k)$ induced by the collection of all k -subsets of $[n]$ that do not include two consecutive elements modulo n (i.e., the k -subsets $A \subseteq [n]$ such that if $i \in A$ then $i + 1 \notin A$, and if $n \in A$ then $1 \notin A$). Schrijver proved in [26], strengthening Lovász's result, that the chromatic number of $S(n, k)$ is equal to that of $K(n, k)$. His proof technique relies on a proof of Kneser's conjecture due to Bárány [1], which was obtained soon after the one of Lovász and combined the topological Borsuk-Ulam theorem with a lemma of Gale [12]. It was further proved in [26] that $S(n, k)$ is vertex-critical, that is, the chromatic number of any proper induced subgraph of $S(n, k)$ is strictly smaller than that of $S(n, k)$.



© Ishay Haviv;

licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 16; pp. 16:1–16:16

Leibniz International Proceedings in Informatics



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

In the computational **KNESER** and **SCHRIJVER** problems, we are given an access to a coloring with $n - 2k + 1$ colors of the vertices of $K(n, k)$ and $S(n, k)$ respectively, and the goal is to find a monochromatic edge, i.e., two vertices with the same color that correspond to disjoint sets. Since the number of colors used by the input coloring is strictly smaller than the chromatic number of the graph [20, 26], it follows that every instance of these problems has a solution. However, the topological argument behind the lower bound on the chromatic number is not constructive, in the sense that it does not suggest an efficient algorithm for finding a monochromatic edge. By an efficient algorithm we mean that its running time is polynomial in n , whereas the number of vertices might be exponentially larger. Hence, it is natural to assume that the input coloring is given as an access to an oracle that given a vertex of the graph returns its color. The input can also be given by some succinct representation, e.g., a Boolean circuit that computes the color of any given vertex.

In recent years, it has been shown that the complexity class **PPA** perfectly captures the complexity of several total search problems for which the existence of the solution relies on the Borsuk-Ulam theorem. This complexity class belongs to a family of classes that were introduced in 1994 by Papadimitriou [25] in the attempt to characterize the mathematical arguments that lie behind the existence of solutions to search problems of **TFNP**. The complexity class **TFNP**, introduced in [24], is the class of total search problems in **NP**, namely, the search problems in which a solution is guaranteed to exist and can be verified in polynomial running time. Papadimitriou has introduced in [25] several subclasses of **TFNP**, each of which consists of the total search problems that can be efficiently reduced to a problem that represents some mathematical argument. One of those subclasses was **PPA** (Polynomial Parity Argument) that corresponds to the fact that every (undirected) graph with maximum degree 2 that has a vertex of degree 1 must have another degree 1 vertex. Hence, **PPA** is the class of all problems in **TFNP** that can be efficiently reduced to the **LEAF** problem, in which given a succinct representation of a graph with maximum degree 2 and given a vertex of degree 1 in the graph, the goal is to find another such vertex.

A prominent example of a **PPA**-complete problem whose totality is related to the Borsuk-Ulam theorem is the one associated with the Consensus Halving theorem [17, 27]. The **PPA**-completeness of the problem was proved for an inverse-polynomial precision parameter by Filos-Ratsikas and Goldberg [7, 8], and this was improved to a constant precision parameter in a recent work of Deligkas, Fearnley, Hollender, and Melissourgos [4]. The hardness of the Consensus Halving problem was used to derive the **PPA**-completeness of several other problems. This includes the Splitting Necklace problem with two thieves, the Discrete Sandwich problem [7, 8], the Fair Independent Set in Cycle problem, and the aforementioned **SCHRIJVER** problem (with the input coloring given as a Boolean circuit) [14]. As for the **KNESER** problem, the question of whether it is **PPA**-complete was proposed by Deng, Feng, and Kulkarni [5]. It is interesting to mention that this question is motivated by connections of the **KNESER** problem to a resource allocation problem called Agreeable Set, that was introduced by Manurangsi and Suksompong [21] and further studied in [13, 15]. It is also motivated by the extension of the **KNESER** problem to Kneser hypergraphs, for which the complexity question was raised by Filos-Ratsikas, Hollender, Sotiraki, and Zampetakis [9].

In the area of parameterized complexity, a problem whose instances involve a parameter k is said to be fixed-parameter tractable with respect to k if it admits an algorithm whose running time is bounded by a polynomial in the input size multiplied by an arbitrary function of k (see, e.g., [3]). Adopting this notion to our setting, where the instance is not given explicitly but as an oracle access, we say that an algorithm for the **KNESER** and **SCHRIJVER** problems is fixed-parameter with respect to k if its running time on an input coloring of,

respectively, $K(n, k)$ and $S(n, k)$ is bounded by $n^{O(1)} \cdot f(k)$ for some function f . In the recent work [15], it was shown that the KNESER problem is fixed-parameter tractable with respect to the parameter k . More specifically, it was shown there that there exists a randomized algorithm that solves the KNESER problem on an input coloring of a Kneser graph $K(n, k)$ in running time $n^{O(1)} \cdot k^{O(k)}$.

1.1 Our Contribution

In the current work, we prove that the SCHRIJVER problem on graphs $S(n, k)$ is fixed-parameter tractable with respect to the parameter k .

► **Theorem 1.** *There exists a randomized algorithm that given integers n and k with $n \geq 2k$ and an oracle access to a coloring of the vertices of the Schrijver graph $S(n, k)$ with $n - 2k + 1$ colors, runs in time $n^{O(1)} \cdot k^{O(k)}$ and returns a monochromatic edge with high probability.*

A few remarks about Theorem 1 are in order here.

- The algorithmic task of finding a monochromatic edge in the Schrijver graph $S(n, k)$ given a coloring of its vertices with $n - 2k + 1$ colors is at least as hard as that of finding a monochromatic edge in the Kneser graph $K(n, k)$ given such a coloring. Indeed, $S(n, k)$ is an induced subgraph of $K(n, k)$ with the same chromatic number. Therefore, the KNESER problem can be solved by applying an algorithm for the SCHRIJVER problem to the restriction of a coloring of a Kneser graph to its Schrijver subgraph. This implies that Theorem 1 strengthens the fixed-parameter tractability result of [15], and yet achieves the same asymptotic dependence on k in the running time.
- In contrast to the current situation of the KNESER problem [5], the SCHRIJVER problem is known to be PPA-complete [14]. Hence, the study of its fixed-parameter tractability is motivated in a stronger sense.
- As mentioned earlier, the Schrijver graph $S(n, k)$ was shown in [26] to be vertex-critical. It follows that for every vertex A of the graph $S(n, k)$, there exists a coloring of its vertices with $n - 2k + 1$ colors, for which only edges that are incident with A are monochromatic. An algorithm for the SCHRIJVER problem, while running on such an input coloring, must be able to find an edge that is incident with this specified vertex A . Nevertheless, the algorithm given in Theorem 1 manages to do so in running time much smaller than the number of vertices, provided that n is sufficiently larger than k .
- Borrowing the terminology of the area of parameterized complexity, our algorithm for the SCHRIJVER problem can be viewed as a randomized polynomial Turing kernelization algorithm for the problem (see, e.g., [10, Chapter 22]). Namely, the problem of finding a monochromatic edge in a Schrijver graph $S(n, k)$ can essentially be reduced by a randomized efficient algorithm to finding a monochromatic edge in a Schrijver graph $S(n', k)$ for $n' = O(k^4)$. This aspect of the algorithm is common to the algorithm for the KNESER problem given in [15] (see [15, Section 3.4] for the details).

Our algorithm for the SCHRIJVER problem extends the approach developed in [15] for the KNESER problem. The adaptation to the SCHRIJVER problem relies on structural properties of induced subgraphs of Schrijver graphs (see Section 3). Their proofs involve some ideas that were applied in the context of Frege propositional proof systems by Istrate, Bonchis, and Craciun [18]. In the remainder of this section, we give an overview of the proof of Theorem 1.

1.2 Proof Overview

Our algorithm for the SCHRIJVER problem is based on the strategy developed in [15] for the KNESER problem. We start by describing the algorithm of [15] for the KNESER problem and then present the modification in the algorithm and in its analysis needed for the SCHRIJVER problem.

Suppose that we are given an oracle access to a coloring of the vertices of the Kneser graph $K(n, k)$ with $n - 2k + 1$ colors. In order to find a monochromatic edge in the graph, we use an efficient algorithm, called “element elimination”, that reduces our problem to that of finding a monochromatic edge in a subgraph of $K(n, k)$ isomorphic to $K(n - 1, k)$ whose vertices are colored by $n - 2k$ colors. Since the chromatic number of the latter is $n - 2k + 1$ [20], such a coloring is guaranteed to have a monochromatic edge in the subgraph. By repeatedly applying this algorithm, we obtain a coloring with $n' - 2k + 1$ colors of a subgraph of $K(n, k)$ isomorphic to $K(n', k)$. When the size n' of the ground set is sufficiently small and depends only on k , a brute force algorithm that queries the oracle for the colors of all vertices allows us to find a monochromatic edge in running time that essentially depends only on k .

We turn to describe now the “element elimination” algorithm. This algorithm picks uniformly and independently polynomially many vertices of the graph $K(n, k)$ and queries the oracle for their colors. If the random samples include two vertices that form a monochromatic edge in the graph, then this edge is returned and we are done. Otherwise, the algorithm identifies a color $i \in [n - 2k + 1]$ that appears on a largest number of vertices among the random samples and an element $j \in [n]$ that is particularly popular on the sampled vertices colored i (say, that belongs to a constant fraction of them). The “element elimination” algorithm suggests to remove the element j from the ground set, and to keep looking for a monochromatic edge in the subgraph induced by the k -subsets of $[n] \setminus \{j\}$.

The correctness of the “element elimination” algorithm for Kneser graphs relies on structural properties of intersecting families of k -subsets of $[n]$. A stability result of Hilton and Milner [16] for the celebrated Erdős-Ko-Rado theorem [6] says that any sufficiently large intersecting family of k -subsets of $[n]$ satisfies that all of its members share a common element. This is used in [15], combined with an idea of Frankl and Kupavskii [11], to show that as long as $n \geq \Omega(k^4)$, if a large color class of the input coloring does not have an element that is quite popular on its members, then the sampled vertices include with high probability a monochromatic edge. Otherwise, for every color i of a large color class there exists some element j that is popular on its vertices, and the algorithm finds such a pair (i, j) with high probability. If this element j belongs to *all* the vertices colored i , then the restriction of the input coloring to the k -subsets of $[n] \setminus \{j\}$ is a coloring with $n - 2k$ colors of a graph isomorphic to $K(n - 1, k)$, as required.

However, although the element j belongs to a significant fraction of the vertices colored i , the coloring might use the color i for vertices that do not include the element j . This might lead to an elimination of the element j while the restriction of the coloring to the k -subsets of $[n] \setminus \{j\}$ still uses the color i , hence the corresponding subgraph is not guaranteed to have a monochromatic edge. This situation is handled in [15] by showing that every vertex colored i that does not include j is disjoint from a non-negligible fraction of the vertices colored i . Therefore, in case that the brute force algorithm that is applied to the subgraph obtained after all iterations of the “element elimination” algorithm finds a vertex A colored by a color i that is associated with an eliminated element j , we pick uniformly at random vertices from the subgraph of the corresponding iteration and with high probability find a neighbor of A colored i and thus a monochromatic edge. This completes the high-level description of the algorithm for the KNESER problem from [15].

Our algorithm for finding a monochromatic edge in a Schrijver graph $S(n, k)$ given a coloring of its vertices with $n - 2k + 1$ colors also uses an “element elimination” algorithm as a main ingredient. Observe, however, that whenever an element $j \in [n]$ is eliminated, the subgraph induced by the k -subsets of $[n] \setminus \{j\}$ is not isomorphic to a Schrijver graph. We therefore consider the subgraph of $S(n, k)$ that corresponds to the cyclic ordering of the elements of $[n] \setminus \{j\}$ which is induced by the cyclic ordering of the elements of $[n]$ (where $j - 1$ precedes $j + 1$). This allows the algorithm to proceed by looking for a monochromatic edge in a graph isomorphic to $S(n - 1, k)$. As before, the eliminated element is chosen as an element $j \in [n]$ that is quite popular on the vertices colored by a color i that corresponds to a large color class of the input coloring. The pair of the color i and the element j is identified using polynomially many vertices chosen uniformly at random from the vertex set of $S(n, k)$.

The main contribution of the current work lies in the analysis of the “element elimination” algorithm for Schrijver graphs. Consider first the case where the input coloring has a large color class that does not have a popular element in its members. For this case we prove that the selected random vertices include a monochromatic edge with high probability. In contrast to the analysis used for Kneser graphs, here we cannot apply the Hilton-Milner theorem [16] that deals with intersecting families of general k -subsets of $[n]$. We overcome this issue using a Hilton-Milner-type result for stable sets, i.e., for vertices of the Schrijver graph, borrowing ideas that were applied by Istrate, Bonchis, and Craciun [18] in the context of Frege propositional proof systems (see Lemma 7). Note that this can be interpreted as an approximate stability result for the analogue of the Erdős-Ko-Rado theorem for stable sets that was proved in 2003 by Talbot [28]. The Hilton-Milner-type result is combined with the approach of [15] and with an idea of [11] to prove that if the vertices of a large color class do not have a popular element, then a pair of vertices chosen uniformly at random from $S(n, k)$ forms a monochromatic edge with a non-negligible probability (see Lemma 8). Hence, picking a polynomial number of them suffices to catch such an edge.

Consider next the case where every large color class of the input coloring has a popular element. Here, the “element elimination” algorithm identifies with high probability a color i of a large color class and an element j that is popular on its vertices. If all the vertices colored i include j then we can safely look for a monochromatic edge in the subgraph of $S(n, k)$ induced by the cyclic ordering of $[n]$ without the element j , as this means that the size of the ground set and the number of colors are both reduced by 1. However, for the scenario where the color class of i involves vertices A that do not include j , we prove, as in [15], that such an A is disjoint from a random set from the color class of i with a non-negligible probability. Note that the analysis in this case again employs the ideas applied by Istrate et al. [18] (see Lemma 10). Then, when such a set A is found by the algorithm, we can go back to the subgraph of the run of the “element elimination” algorithm that identified the color of A and find a neighbor of A from this color class using random samples.

1.3 Outline

The rest of the paper is organized as follows. In Section 2, we gather several definitions and results that will be used throughout the paper. In Section 3, we present and prove several structural results on induced subgraphs of Schrijver graphs needed for the analysis of our algorithm. Finally, in Section 4, we present and analyze our randomized fixed-parameter algorithm for the SCHRIJVER problem and prove Theorem 1.

2 Preliminaries

2.1 Kneser and Schrijver Graphs

Consider the following definition.

► **Definition 2.** For a family \mathcal{F} of non-empty sets, let $K(\mathcal{F})$ denote the graph on the vertex set \mathcal{F} in which two vertices are adjacent if they represent disjoint sets.

For a set X and an integer k , let $\binom{X}{k}$ denote the family of all k -subsets of X . Note that the Kneser graph $K(n, k)$ can be defined for integers n and k with $n \geq 2k$ as the graph $K(\binom{[n]}{k})$.

A set $A \subseteq [n]$ is said to be *stable* if it does not include two consecutive elements modulo n , that is, it forms an independent set in the n -vertex cycle with the numbering from 1 to n along the cycle. For integers n and k with $n \geq 2k$, let $\binom{[n]}{k}_{\text{stab}}$ denote the collection of all stable k -subsets of $[n]$. The Schrijver graph $S(n, k)$ is defined as the graph $K(\binom{[n]}{k}_{\text{stab}})$. Equivalently, it is the subgraph of $K(n, k)$ induced by the vertex set $\binom{[n]}{k}_{\text{stab}}$.

For a set $X \subseteq [n]$ consider the natural cyclic ordering of the elements of X induced by that of $[n]$, and let $\binom{X}{k}_{\text{stab}}$ denote the collection of all k -subsets of X that do not include two consecutive elements according to this ordering. More formally, letting $j_1 < j_2 < \dots < j_{|X|}$ denote the elements of X , $\binom{X}{k}_{\text{stab}}$ stands for the collection of all independent sets of size k in the cycle on the vertex set X with the numbering $j_1, \dots, j_{|X|}$ along the cycle. Note that $\binom{X}{k}_{\text{stab}} \subseteq \binom{X'}{k}_{\text{stab}}$ whenever $X \subseteq X' \subseteq [n]$. Note further that the graph $K(\binom{X}{k}_{\text{stab}})$ is isomorphic to the Schrijver graph $S(|X|, k)$.

The chromatic number of the graph $S(n, k)$ was determined by Schrijver [26], strengthening a result of Lovász [20].

► **Theorem 3** ([26]). For all integers n and k with $n \geq 2k$, $\chi(S(n, k)) = n - 2k + 2$.

A family $\mathcal{F} \subseteq \binom{[n]}{k}$ of k -subsets of $[n]$ is called *intersecting* if for every two sets $F_1, F_2 \in \mathcal{F}$ it holds that $F_1 \cap F_2 \neq \emptyset$. Note that such a family forms an independent set in the graph $K(n, k)$. If the members of \mathcal{F} share a common element, then we say that the intersecting family \mathcal{F} is *trivial*.

The computational search problem associated with the Schrijver graph is defined as follows.

► **Definition 4.** In the SCHRIJVER problem, the input is a coloring $c : \binom{[n]}{k}_{\text{stab}} \rightarrow [n - 2k + 1]$ of the vertices of the Schrijver graph $S(n, k)$ with $n - 2k + 1$ colors for integers n and k with $n \geq 2k$, and the goal is to find a monochromatic edge.

The existence of a solution to every instance of the SCHRIJVER problem follows from Theorem 3. In our algorithm for the SCHRIJVER problem, we consider the black-box input model, where the input coloring is given as an oracle access that for a vertex A returns its color $c(A)$. This reflects the fact that the algorithm does not rely on the representation of the input coloring.

2.2 Chernoff-Hoeffding Bound

We need the following concentration result (see, e.g., [23, Theorem 2.1]).

► **Theorem 5** (Chernoff-Hoeffding Bound). Let $0 < p < 1$, let X_1, \dots, X_m be m independent binary random variables satisfying $\Pr[X_i = 1] = p$ and $\Pr[X_i = 0] = 1 - p$ for all i , and put $\bar{X} = \frac{1}{m} \cdot \sum_{i=1}^m X_i$. Then, for any $\mu \geq 0$,

$$\Pr[|\bar{X} - p| \geq \mu] \leq 2 \cdot e^{-2m\mu^2}.$$

3 Induced Subgraphs of Schrijver Graphs

In this section, we provide a couple of lemmas on induced subgraphs of Schrijver graphs that will play a central role in the analysis of our algorithm for the SCHRIJVER problem. We start with some preliminary claims related to counting stable sets.

3.1 Counting Stable Sets

The following claim employs an argument of Istrate, Bonchis, and Craciun [18]. Its proof is omitted and can be found in the full version of the paper.

▷ **Claim 6.** For integers $k \geq 2$ and $n \geq 2k$ and for every two distinct integers $a, b \in [n]$, the number of stable k -subsets F of $[n]$ satisfying $\{a, b\} \subseteq F$ is at most $\binom{n-k-2}{k-2}$.

The following result stems from Claim 6 and can be viewed as an approximate variant of the Hilton-Milner theorem of [16] for stable sets (see [18]).

► **Lemma 7.** For integers $k \geq 2$ and $n \geq 2k$, every non-trivial intersecting family \mathcal{F} of stable k -subsets of $[n]$ satisfies $|\mathcal{F}| \leq k^2 \cdot \binom{n-k-2}{k-2}$.

Proof. Let \mathcal{F} be a non-trivial intersecting family of stable k -subsets of $[n]$. Consider an arbitrary set $A = \{a_1, \dots, a_k\}$ in \mathcal{F} . Since \mathcal{F} is non-trivial, for every $t \in [k]$, there exists a set $B_t \in \mathcal{F}$ satisfying $a_t \notin B_t$. Since \mathcal{F} is intersecting, every set in \mathcal{F} intersects A , and therefore includes the element a_t for some $t \in [k]$. Such a set further intersects B_t , hence it also includes some element $b \in B_t$ (which is different from a_t). By Claim 6, the number of stable k -subsets of $[n]$ that include both a_t and b does not exceed $\binom{n-k-2}{k-2}$. Since there are at most k^2 ways to choose the elements a_t and b , this implies that $|\mathcal{F}| \leq k^2 \cdot \binom{n-k-2}{k-2}$, as required. ◀

3.2 Induced Subgraphs of Schrijver Graphs

We are ready to prove the lemmas that lie at the heart of the analysis of our algorithm for the SCHRIJVER problem. The first lemma, given below, shows that in a large induced subgraph of the Schrijver graph $S(n, k)$ whose vertices do not have a popular element, a random pair of vertices forms an edge with a non-negligible probability.

► **Lemma 8.** For integers $k \geq 2$ and $n \geq 2k$, let \mathcal{F} be a family of stable k -subsets of $[n]$ whose size satisfies $|\mathcal{F}| \geq k^2 \cdot \binom{n-k-2}{k-2}$ and let $\gamma \in (0, 1]$. Suppose that every element of $[n]$ belongs to at most γ fraction of the sets of \mathcal{F} . Then, the probability that two random sets chosen uniformly and independently from \mathcal{F} are adjacent in $K(\mathcal{F})$ is at least

$$\frac{1}{2} \cdot \left(1 - \gamma - \frac{k^2}{|\mathcal{F}|} \cdot \binom{n-k-2}{k-2}\right) \cdot \left(1 - \frac{k^2}{|\mathcal{F}|} \cdot \binom{n-k-2}{k-2}\right).$$

Proof. Let $\mathcal{F} \subseteq \binom{[n]}{k}_{\text{stab}}$ be a family of sets as in the statement of the lemma. We first claim that every subfamily $\mathcal{F}' \subseteq \mathcal{F}$ whose size satisfies

$$|\mathcal{F}'| \geq \gamma \cdot |\mathcal{F}| + k^2 \cdot \binom{n-k-2}{k-2} \tag{1}$$

spans an edge in $K(\mathcal{F})$. To see this, consider such an \mathcal{F}' , and notice that the assumption that every element of $[n]$ belongs to at most γ fraction of the sets of \mathcal{F} , combined with the fact that $|\mathcal{F}'| > \gamma \cdot |\mathcal{F}|$, implies that \mathcal{F}' is not a trivial family, that is, its sets do not share a

16:8 A Fixed-Parameter Algorithm for the Schrijver Problem

common element. In addition, using $|\mathcal{F}'| > k^2 \cdot \binom{n-k-2}{k-2}$, it follows from Lemma 7 that \mathcal{F}' is not a non-trivial intersecting family. We thus conclude that \mathcal{F}' is not an intersecting family, hence it spans an edge in $K(\mathcal{F})$.

We next show a lower bound on the size of a maximum matching in $K(\mathcal{F})$. Consider the process that maintains a subfamily \mathcal{F}' of \mathcal{F} , initiated as \mathcal{F} , and that removes from \mathcal{F}' the two endpoints of some edge spanned by \mathcal{F}' as long as its size satisfies the condition given in (1). The pairs of vertices that are removed during the process form a matching \mathcal{M} in $K(\mathcal{F})$, whose size satisfies

$$\begin{aligned} |\mathcal{M}| &\geq \frac{1}{2} \cdot \left(|\mathcal{F}| - \left(\gamma \cdot |\mathcal{F}| + k^2 \cdot \binom{n-k-2}{k-2} \right) \right) \\ &= \frac{1}{2} \cdot \left((1-\gamma) \cdot |\mathcal{F}| - k^2 \cdot \binom{n-k-2}{k-2} \right). \end{aligned} \quad (2)$$

We now consider the sum of the degrees of adjacent vertices in the graph $K(\mathcal{F})$. Let $A, B \in \mathcal{F}$ be any adjacent vertices in $K(\mathcal{F})$. Since A and B are adjacent, they satisfy $A \cap B = \emptyset$, hence every vertex of \mathcal{F} that is not adjacent to A nor to B must include two distinct elements $a \in A$ and $b \in B$. For every two such elements, it follows from Claim 6 that the number of stable k -subsets of $[n]$ that include them both is at most $\binom{n-k-2}{k-2}$. Therefore, the number of vertices of \mathcal{F} that are not adjacent to A nor to B does not exceed $k^2 \cdot \binom{n-k-2}{k-2}$. This implies that the degrees of A and B in $K(\mathcal{F})$ satisfy

$$d(A) + d(B) \geq |\mathcal{F}| - k^2 \cdot \binom{n-k-2}{k-2}.$$

Let \mathcal{E} denote the edge set of $K(\mathcal{F})$. We combine the above bound with the lower bound given in (2) on the size of the matching \mathcal{M} , to obtain that

$$\begin{aligned} 2 \cdot |\mathcal{E}| = \sum_{F \in \mathcal{F}} d(F) &\geq \sum_{\{A, B\} \in \mathcal{M}} (d(A) + d(B)) \geq |\mathcal{M}| \cdot \left(|\mathcal{F}| - k^2 \cdot \binom{n-k-2}{k-2} \right) \\ &\geq \frac{1}{2} \cdot \left((1-\gamma) \cdot |\mathcal{F}| - k^2 \cdot \binom{n-k-2}{k-2} \right) \cdot \left(|\mathcal{F}| - k^2 \cdot \binom{n-k-2}{k-2} \right). \end{aligned}$$

Finally, consider a pair of random vertices chosen uniformly and independently from \mathcal{F} . The probability that they form an edge in $K(\mathcal{F})$ is twice the number of edges in $K(\mathcal{F})$ divided by $|\mathcal{F}|^2$. Hence, the above bound on $2 \cdot |\mathcal{E}|$ completes the proof. \blacktriangleleft

As a corollary of Lemma 8, we obtain the following. The proof is omitted.

► Corollary 9. *For integers $k \geq 2$ and $n \geq 10k^4$, let \mathcal{F} be a family of stable k -subsets of $[n]$ of size $|\mathcal{F}| \geq \frac{1}{2n} \cdot \left| \binom{[n]}{k}_{\text{stab}} \right|$ and let $\gamma \in (0, 1]$. Suppose that every element of $[n]$ belongs to at most γ fraction of the sets of \mathcal{F} . Then, the probability that two random sets chosen uniformly and independently from \mathcal{F} are adjacent in $K(\mathcal{F})$ is at least $\frac{3}{8} \cdot \left(\frac{3}{4} - \gamma \right)$.*

The following lemma shows that if a large collection of vertices of $S(n, k)$ has a quite popular element, then every k -subset of $[n]$ that does not include this element is disjoint from many of the vertices in the collection.

► **Lemma 10.** *For integers $k \geq 2$ and $n \geq 2k$, let $X \subseteq [n]$ be a set, let $\mathcal{F} \subseteq \binom{X}{k}_{\text{stab}}$ be a family, and let $\gamma \in (0, 1]$. Let $j \in X$ be an element that belongs to at least γ fraction of the sets of \mathcal{F} , and suppose that $A \in \binom{[n]}{k}$ is a set satisfying $j \notin A$. Then, the probability that a random set chosen uniformly from \mathcal{F} is disjoint from A is at least*

$$\gamma - \frac{k}{|\mathcal{F}|} \cdot \binom{|X| - k - 2}{k - 2}.$$

Proof. Let $\mathcal{F} \subseteq \binom{X}{k}_{\text{stab}}$ be a family as in the lemma, and put $\mathcal{F}' = \{F \in \mathcal{F} \mid j \in F\}$. By assumption, it holds that $|\mathcal{F}'| \geq \gamma \cdot |\mathcal{F}|$. Suppose that $A \in \binom{[n]}{k}$ is a set satisfying $j \notin A$. We claim that for every $i \in A$, the number of sets $B \in \binom{X}{k}_{\text{stab}}$ satisfying $\{i, j\} \subseteq B$ does not exceed $\binom{|X| - k - 2}{k - 2}$. Indeed, if $i \notin X$ then there are no such sets, so the bound trivially holds, and if $i \in X$, the bound follows from Claim 6, using the one-to-one correspondence between $\binom{X}{k}_{\text{stab}}$ and the vertex set of $S(|X|, k)$. This implies that the number of sets $B \in \binom{X}{k}_{\text{stab}}$ with $j \in B$ that intersect A does not exceed $k \cdot \binom{|X| - k - 2}{k - 2}$. It thus follows that the number of sets of \mathcal{F} that are disjoint from A is at least

$$|\mathcal{F}'| - k \cdot \binom{|X| - k - 2}{k - 2} \geq \gamma \cdot |\mathcal{F}| - k \cdot \binom{|X| - k - 2}{k - 2}.$$

Hence, a random set chosen uniformly from \mathcal{F} is disjoint from A with the desired probability. ◀

As a corollary of Lemma 10, we obtain the following. The proof is omitted.

► **Corollary 11.** *For integers $k \geq 2$ and n , let $X \subseteq [n]$ be a set of size $|X| \geq 10k^3$, let $\mathcal{F} \subseteq \binom{X}{k}_{\text{stab}}$ be a family of size $|\mathcal{F}| \geq \frac{1}{2|X|} \cdot \left| \binom{X}{k}_{\text{stab}} \right|$, and let $\gamma \in (0, 1]$. Let $j \in X$ be an element that belongs to at least γ fraction of the sets of \mathcal{F} , and suppose that $A \in \binom{[n]}{k}$ is a set satisfying $j \notin A$. Then, the probability that a random set chosen uniformly from \mathcal{F} is disjoint from A is at least $\gamma - \frac{1}{4}$.*

4 A Fixed-Parameter Algorithm for the Schrijver Problem

In this section we provide our randomized fixed-parameter algorithm for the SCHRIJVER problem. We first describe the “element elimination” algorithm and then use it to present the final algorithm and to prove Theorem 1.

4.1 The Element Elimination Algorithm

The “element elimination” algorithm, given by the following theorem, will be used to repeatedly reduce the size of the ground set of a Schrijver graph while looking for a monochromatic edge.

► **Theorem 12.** *There exists a randomized algorithm that given integers n and k , a set $X \subseteq [n]$ of size $|X| \geq 10k^4$, a parameter $\varepsilon > 0$, a set of colors $C \subseteq [n - 2k + 1]$ of size $|C| = |X| - 2k + 1$, and an oracle access to a coloring $c : \binom{X}{k}_{\text{stab}} \rightarrow [n - 2k + 1]$ of the vertices of $K(\binom{X}{k}_{\text{stab}})$, runs in time $\text{poly}(n, \ln(1/\varepsilon))$ and returns, with probability at least $1 - \varepsilon$,*

- a monochromatic edge of $K(\binom{X}{k}_{\text{stab}})$, or*
- a vertex $A \in \binom{X}{k}_{\text{stab}}$ satisfying $c(A) \notin C$, or*
- a color $i \in C$ and an element $j \in X$ such that for every $A \in \binom{[n]}{k}_{\text{stab}}$ with $j \notin A$, a random vertex B chosen uniformly from $\binom{X}{k}_{\text{stab}}$ satisfies $c(B) = i$ and $A \cap B = \emptyset$ with probability at least $\frac{1}{9n}$.*

16:10 A Fixed-Parameter Algorithm for the Schrijver Problem

Proof. For integers n and k , let $X \subseteq [n]$, $C \subseteq [n - 2k + 1]$, and $c : \binom{X}{k}_{\text{stab}} \rightarrow [n - 2k + 1]$ be an input satisfying $|X| \geq 10k^4$ and $|C| = |X| - 2k + 1$ as in the statement of the theorem. It can be assumed that $k \geq 2$. Indeed, Theorem 3 guarantees that the graph $K(\binom{X}{k}_{\text{stab}})$, which is isomorphic to $S(|X|, k)$, has either a monochromatic edge or a vertex whose color does not belong to C . Hence, for $k = 1$, an output of type (a) or (b) can be found by querying the oracle for the colors of all the vertices in time polynomial in n . For $k \geq 2$, consider the algorithm that given an input as above acts as follows (see Algorithm 1).

The algorithm first selects uniformly and independently m random sets $A_1, \dots, A_m \in \binom{X}{k}_{\text{stab}}$ for $m = b \cdot n^2 \cdot \ln(n/\varepsilon)$, where b is some fixed constant to be determined later (see lines 1–2), and queries the oracle for their colors. If the sampled sets include two vertices that form a monochromatic edge in $K(\binom{X}{k}_{\text{stab}})$, then the algorithm returns such an edge (output of type (a); see line 5). If they include a vertex whose color does not belong to C , then the algorithm returns it (output of type (b); see line 10). Otherwise, the algorithm defines $i^* \in C$ as a color that appears on a largest number of sampled sets A_t (see lines 13–16). It further defines $j^* \in X$ as an element that belongs to a largest number of sampled sets A_t with $c(A_t) = i^*$ (see lines 17–20). Then, the algorithm returns the pair (i^*, j^*) (output of type (c); see line 21).

The running time of the algorithm is clearly polynomial in n and in $\ln(1/\varepsilon)$. We turn to prove that for every input, the algorithm returns a valid output, of type (a), (b), or (c), with probability at least $1 - \varepsilon$. We start with the following lemma that shows that if the input coloring has a large color class with no popular element, then the algorithm returns a valid output of type (a) with the desired probability.

► **Lemma 13.** *Suppose that the input coloring c has a color class $\mathcal{F} \subseteq \binom{X}{k}_{\text{stab}}$ of size $|\mathcal{F}| \geq \frac{1}{2|X|} \cdot \left| \binom{X}{k}_{\text{stab}} \right|$ such that every element of X belongs to at most half of the sets of \mathcal{F} . Then, Algorithm 1 returns a monochromatic edge with probability at least $1 - \varepsilon$.*

Proof. Let \mathcal{F} be as in the lemma. Using the assumptions $k \geq 2$ and $|X| \geq 10k^4$ and using the one-to-one correspondence between $\binom{X}{k}_{\text{stab}}$ and the vertex set of $S(|X|, k)$, we can apply Corollary 9 with $\gamma = \frac{1}{2}$ to obtain that two random sets chosen uniformly and independently from \mathcal{F} are adjacent in $K(\mathcal{F})$ with probability at least $\frac{3}{8} \cdot (\frac{3}{4} - \gamma) = \frac{3}{32}$. Further, since the family \mathcal{F} satisfies $|\mathcal{F}| \geq \frac{1}{2|X|} \cdot \left| \binom{X}{k}_{\text{stab}} \right|$, a random vertex chosen uniformly from $\binom{X}{k}_{\text{stab}}$ belongs to \mathcal{F} with probability at least $\frac{1}{2|X|}$. Hence, for two random vertices chosen uniformly and independently from $\binom{X}{k}_{\text{stab}}$, the probability that they both belong to \mathcal{F} is at least $(\frac{1}{2|X|})^2$, and conditioned on this event, their probability to form an edge in $K(\mathcal{F})$ is at least $\frac{3}{32}$. This implies that the probability that two random vertices chosen uniformly and independently from $\binom{X}{k}_{\text{stab}}$ form a monochromatic edge in $K(\binom{X}{k}_{\text{stab}})$ is at least $(\frac{1}{2|X|})^2 \cdot \frac{3}{32} = \frac{3}{128|X|^2}$.

Now, by considering $\lfloor m/2 \rfloor$ pairs of the random sets chosen by Algorithm 1 (line 2), it follows that the probability that no pair forms a monochromatic edge does not exceed

$$\left(1 - \frac{3}{128|X|^2}\right)^{\lfloor m/2 \rfloor} \leq e^{-3 \cdot \lfloor m/2 \rfloor / (128|X|^2)} \leq \varepsilon,$$

where the last inequality follows by $|X| \leq n$ and by the choice of m , assuming that the constant b is sufficiently large. It thus follows that with probability at least $1 - \varepsilon$, the algorithm returns a monochromatic edge, as required. ◀

■ **Algorithm 1** Element Elimination Algorithm (Theorem 12).

Input: Integers n and $k \geq 2$, a set $X \subseteq [n]$ of size $|X| \geq 10k^4$, a set of colors $C \subseteq [n - 2k + 1]$ of size $|C| = |X| - 2k + 1$, and an oracle access to a coloring $c : \binom{X}{k}_{\text{stab}} \rightarrow [n - 2k + 1]$ of $K(\binom{X}{k}_{\text{stab}})$.

Output: (a) A monochromatic edge of $K(\binom{X}{k}_{\text{stab}})$, or (b) a vertex $A \in \binom{X}{k}_{\text{stab}}$ satisfying $c(A) \notin C$, or (c) a color $i \in C$ and an element $j \in X$ such that for every $A \in \binom{[n]}{k}_{\text{stab}}$ with $j \notin A$, a random vertex $B \in \binom{X}{k}_{\text{stab}}$ chosen uniformly satisfies $c(B) = i$ and $A \cap B = \emptyset$ with probability at least $\frac{1}{9n}$.

```

1:  $m \leftarrow b \cdot n^2 \cdot \ln(n/\varepsilon)$  for a sufficiently large constant  $b$ 
2: pick uniformly and independently at random sets  $A_1, \dots, A_m \in \binom{X}{k}_{\text{stab}}$ 
3: for all  $t, t' \in [m]$  do
4:   if  $c(A_t) = c(A_{t'})$  and  $A_t \cap A_{t'} = \emptyset$  then
5:     return  $\{A_t, A_{t'}\}$  ▷ output of type (a)
6:   end if
7: end for
8: for all  $t \in [m]$  do
9:   if  $c(A_t) \notin C$  then
10:    return  $A_t$  ▷ output of type (b)
11:  end if
12: end for
13: for all  $i \in C$  do
14:    $\tilde{\alpha}_i \leftarrow \frac{1}{m} \cdot |\{t \in [m] \mid c(A_t) = i\}|$ 
15: end for
16:  $i^* \leftarrow$  an  $i \in C$  with largest value of  $\tilde{\alpha}_i$ 
17: for all  $j \in X$  do
18:    $\tilde{\gamma}_{i^*,j} \leftarrow \frac{1}{m} \cdot |\{t \in [m] \mid c(A_t) = i^* \text{ and } j \in A_t\}|$ 
19: end for
20:  $j^* \leftarrow$  a  $j \in X$  with largest value of  $\tilde{\gamma}_{i^*,j}$ 
21: return  $(i^*, j^*)$  ▷ output of type (c)

```

We next handle the case in which every large color class of the input coloring has a popular element. To do so, we first show that the samples of the algorithm provide a good estimation for the fraction of vertices in each color class as well as for the fraction of vertices that share any given element in each color class. For every color $i \in C$, let α_i denote the fraction of vertices of $K(\binom{X}{k}_{\text{stab}})$ colored i , that is,

$$\alpha_i = \frac{|\{A \in \binom{X}{k}_{\text{stab}} \mid c(A) = i\}|}{|\binom{X}{k}_{\text{stab}}|},$$

and let $\tilde{\alpha}_i$ denote the fraction of the vertices sampled by the algorithm that are colored i (see line 14). Similarly, for every $i \in C$ and $j \in X$, let $\gamma_{i,j}$ denote the fraction of vertices of $K(\binom{X}{k}_{\text{stab}})$ colored i that include j , that is,

$$\gamma_{i,j} = \frac{|\{A \in \binom{X}{k}_{\text{stab}} \mid c(A) = i \text{ and } j \in A\}|}{|\binom{X}{k}_{\text{stab}}|},$$

and let $\tilde{\gamma}_{i,j}$ denote the fraction of the vertices sampled by the algorithm that are colored i and include j . Let E denote the event that

$$|\alpha_i - \tilde{\alpha}_i| \leq \frac{1}{2|X|} \quad \text{and} \quad |\gamma_{i,j} - \tilde{\gamma}_{i,j}| \leq \frac{1}{2|X|} \quad \text{for all } i \in C, j \in X. \quad (3)$$

16:12 A Fixed-Parameter Algorithm for the Schrijver Problem

By a standard concentration argument, we obtain the following lemma.

► **Lemma 14.** *The probability of the event E is at least $1 - \varepsilon$.*

Proof. By the Chernoff-Hoeffding bound (Theorem 5) applied with $\mu = \frac{1}{2|X|}$, the probability that an inequality from (3) does not hold is at most

$$2 \cdot e^{-2m/(4|X|^2)} \leq \frac{\varepsilon}{n^2},$$

where the inequality follows by $|X| \leq n$ and by the choice of m , assuming that the constant b is sufficiently large. By the union bound over all the colors $i \in C$ and all the pairs $(i, j) \in C \times X$, that is, over $|C| + |C| \cdot |X| = |C| \cdot (1 + |X|) \leq n^2$ events, we get that all the inequalities in (3) hold with probability at least $1 - n^2 \cdot \frac{\varepsilon}{n^2} = 1 - \varepsilon$, as required. ◀

We now show that if every large color class of the input coloring has a popular element and the event E occurs, then the algorithm returns a valid output.

► **Lemma 15.** *Suppose that the coloring c satisfies that for every color class $\mathcal{F} \subseteq \binom{X}{k}_{\text{stab}}$ whose size satisfies $|\mathcal{F}| \geq \frac{1}{2|X|} \cdot \left| \binom{X}{k}_{\text{stab}} \right|$ there exists an element of X that belongs to more than half of the sets of \mathcal{F} . Then, if the event E occurs, Algorithm 1 returns a valid output.*

Proof. Assume that the event E occurs. If Algorithm 1 returns an output of type (a) or (b), i.e., a monochromatic edge or a vertex whose color does not belong to C , then the output is verified before it is returned and is thus valid. So suppose that the algorithm returns a pair $(i^*, j^*) \in C \times X$. Recall that the color i^* is defined by Algorithm 1 as an $i \in C$ with largest value of $\tilde{\alpha}_i$ (see line 16). Since the colors of all the sampled sets belong to C , it follows that $\sum_{i \in C} \tilde{\alpha}_i = 1$, and thus

$$\tilde{\alpha}_{i^*} \geq \frac{1}{|C|} \geq \frac{1}{|X|}, \quad (4)$$

where the last inequality follows by $|C| = |X| - 2k + 1 \leq |X|$.

Let \mathcal{F} be the family of vertices of $K\left(\binom{X}{k}_{\text{stab}}\right)$ colored i^* , i.e.,

$$\mathcal{F} = \left\{ A \in \binom{X}{k}_{\text{stab}} \mid c(A) = i^* \right\}.$$

Since the event E occurs (see (3)), it follows from (4) that

$$|\mathcal{F}| = \alpha_{i^*} \cdot \left| \binom{X}{k}_{\text{stab}} \right| \geq \left(\tilde{\alpha}_{i^*} - \frac{1}{2|X|} \right) \cdot \left| \binom{X}{k}_{\text{stab}} \right| \geq \frac{1}{2|X|} \cdot \left| \binom{X}{k}_{\text{stab}} \right|.$$

Hence, by the assumption of the lemma, there exists an element $j \in X$ that belongs to more than half of the sets of \mathcal{F} , that is, $\gamma_{i^*, j} > \frac{1}{2}$. Since the event E occurs, it follows that this j satisfies $\tilde{\gamma}_{i^*, j} > \frac{1}{2} - \frac{1}{2|X|}$. Recalling that the element j^* is defined by Algorithm 1 as a $j \in X$ with largest value of $\tilde{\gamma}_{i^*, j}$ (see line 20), it must satisfy $\tilde{\gamma}_{i^*, j^*} > \frac{1}{2} - \frac{1}{2|X|}$, and using again the fact that the event E occurs, we derive that $\gamma_{i^*, j^*} \geq \tilde{\gamma}_{i^*, j^*} - \frac{1}{2|X|} > \frac{1}{2} - \frac{1}{|X|}$.

By $k \geq 2$ and $|X| \geq 10k^4$, we can apply Corollary 11 with \mathcal{F} , j^* , and $\gamma = \frac{1}{2} - \frac{1}{|X|}$ to obtain that for every set $A \in \binom{[n]}{k}_{\text{stab}}$ with $j^* \notin A$, the probability that a random set chosen uniformly from \mathcal{F} is disjoint from A is at least $\gamma - \frac{1}{4}$. Since the probability that a random set

chosen uniformly from $\binom{X}{k}_{\text{stab}}$ belongs to \mathcal{F} is at least $\frac{1}{2|X|}$, it follows that the probability that a random set B chosen uniformly from $\binom{X}{k}_{\text{stab}}$ satisfies $c(B) = i^*$ and $A \cap B = \emptyset$ is at least

$$\frac{1}{2|X|} \cdot \left(\gamma - \frac{1}{4}\right) = \frac{1}{2|X|} \cdot \left(\frac{1}{4} - \frac{1}{|X|}\right) \geq \frac{1}{9|X|} \geq \frac{1}{9n},$$

where the first inequality holds because $k \geq 2$ and $|X| \geq 10k^4$. This implies that (i^*, j^*) is a valid output of type (c). ◀

Equipped with Lemmas 13, 14, and 15, we are ready to derive the correctness of Algorithm 1 and to complete the proof of Theorem 12. If the input coloring c has a color class $\mathcal{F} \subseteq \binom{X}{k}_{\text{stab}}$ of size $|\mathcal{F}| \geq \frac{1}{2|X|} \cdot \left|\binom{X}{k}_{\text{stab}}\right|$ such that every element of X belongs to at most half of the sets of \mathcal{F} , then, by Lemma 13, the algorithm returns with probability at least $1 - \varepsilon$ a monochromatic edge, i.e., a valid output of type (a). Otherwise, the input coloring c satisfies that for every color class $\mathcal{F} \subseteq \binom{X}{k}_{\text{stab}}$ of size $|\mathcal{F}| \geq \frac{1}{2|X|} \cdot \left|\binom{X}{k}_{\text{stab}}\right|$ there exists an element of X that belongs to more than half of the sets of \mathcal{F} . By Lemma 14, the event E occurs with probability at least $1 - \varepsilon$, implying by Lemma 15 that with such probability, the algorithm returns a valid output. It thus follows that for every input coloring the algorithm returns a valid output with probability at least $1 - \varepsilon$, and we are done. ◀

4.2 The Fixed-Parameter Algorithm for the Schrijver Problem

We turn to present our fixed-parameter algorithm for the SCHRIJVER problem and to complete the proof of Theorem 1.

Proof of Theorem 1. Suppose that we are given, for integers n and k with $n \geq 2k$, an oracle access to a coloring $c : \binom{[n]}{k}_{\text{stab}} \rightarrow [n - 2k + 1]$ of the vertices of the Schrijver graph $S(n, k)$. It suffices to present an algorithm with success probability at least $1/2$, because the latter can be easily amplified by repetitions. Our algorithm has two phases, as described below (see Algorithm 2).

In the first phase, the algorithm repeatedly applies the “element elimination” algorithm given in Theorem 12 (Algorithm 1). Initially, we define

$$s = \max(n - 10k^4, 0), \quad X_0 = [n], \quad \text{and} \quad C_0 = [n - 2k + 1].$$

In the l th iteration, $0 \leq l < s$, we call Algorithm 1 with $n, k, X_l, C_l, \varepsilon = \frac{1}{4n}$ and with the restriction of the given coloring c to the vertices of $\binom{X_l}{k}_{\text{stab}}$ to obtain with probability at least $1 - \varepsilon$,

- (a). a monochromatic edge $\{A, B\}$ of $K\left(\binom{X_l}{k}_{\text{stab}}\right)$, or
- (b). a vertex $A \in \binom{X_l}{k}_{\text{stab}}$ satisfying $c(A) \notin C_l$, or
- (c). a color $i_l \in C_l$ and an element $j_l \in X_l$ such that for every $A \in \binom{[n]}{k}_{\text{stab}}$ with $j_l \notin A$, a random vertex B chosen uniformly from $\binom{X_l}{k}_{\text{stab}}$ satisfies $c(B) = i_l$ and $A \cap B = \emptyset$ with probability at least $\frac{1}{9n}$.

As will be explained shortly, if the output of Algorithm 1 is of type (a) or (b) then we either return a monochromatic edge or declare “failure”, and if the output is a pair (i_l, j_l) of type (c) then we define $X_{l+1} = X_l \setminus \{j_l\}$ and $C_{l+1} = C_l \setminus \{i_l\}$ and, as long as $l < s$, proceed to the next call of Algorithm 1. Note that the sizes of the sets X_l and C_l are reduced by 1 in every iteration, hence we maintain the equality $|C_l| = |X_l| - 2k + 1$ for all l . We now describe how the algorithm acts in the l th iteration for each type of output returned by Algorithm 1.

16:14 A Fixed-Parameter Algorithm for the Schrijver Problem

■ **Algorithm 2** The Algorithm for the SCHRIJVER Problem (Theorem 1).

Input: Integers n, k with $n \geq 2k$ and an oracle access to a coloring $c : \binom{[n]}{k}_{\text{stab}} \rightarrow [n - 2k + 1]$.

Output: A monochromatic edge of $S(n, k)$.

```

1:  $s \leftarrow \max(n - 10k^4, 0)$ ,  $X_0 \leftarrow [n]$ ,  $C_0 \leftarrow [n - 2k + 1]$  ▷  $|C_0| = |X_0| - 2k + 1$ 
2: for all  $l = 0, 1, \dots, s - 1$  do ▷ first phase
3:   call Algorithm 1 with  $n, k, X_l, C_l, \varepsilon = \frac{1}{4n}$  and with the restriction of  $c$  to  $\binom{X_l}{k}_{\text{stab}}$ 
4:   if Algorithm 1 returns an edge  $\{A, B\}$  with  $c(A) = c(B)$  then ▷ output of type (a)
5:     return  $\{A, B\}$ 
6:   end if
7:   if Algorithm 1 returns a vertex  $A \in \binom{X_l}{k}_{\text{stab}}$  with  $c(A) = i_r \notin C_l$  then ▷ output of
   type (b)
8:     for all  $t \in [18n]$  do
9:       pick uniformly at random a set  $B_t \in \binom{X_r}{k}_{\text{stab}}$ 
10:      if  $c(B_t) = i_r$  and  $A \cap B_t = \emptyset$  then
11:        return  $\{A, B_t\}$ 
12:      end if
13:    end for
14:    return “failure”
15:  end if
16:  if Algorithm 1 returns a pair  $(i_l, j_l) \in C_l \times X_l$  then ▷ output of type (c)
17:     $X_{l+1} \leftarrow X_l \setminus \{j_l\}$ ,  $C_{l+1} \leftarrow C_l \setminus \{i_l\}$  ▷  $|C_{l+1}| = |X_{l+1}| - 2k + 1$ 
18:  end if
19: end for
20: query the oracle for the colors of all the vertices of  $K(\binom{X_s}{k}_{\text{stab}})$  ▷ second phase
21: if there exists a vertex  $A \in \binom{X_s}{k}_{\text{stab}}$  of color  $c(A) = i_r \notin C_s$  then
22:   for all  $t \in [18n]$  do
23:     pick uniformly at random a set  $B_t \in \binom{X_r}{k}_{\text{stab}}$ 
24:     if  $c(B_t) = i_r$  and  $A \cap B_t = \emptyset$  then
25:       return  $\{A, B_t\}$ 
26:     end if
27:   end for
28:   return “failure”
29: else
30:   find  $A, B \in \binom{X_s}{k}_{\text{stab}}$  satisfying  $c(A) = c(B)$  and  $A \cap B = \emptyset$  ▷ exist by Theorem 3 [20]
31:   return  $\{A, B\}$ 
32: end if

```

If the output is of type (a), then the returned monochromatic edge of $K(\binom{X_l}{k}_{\text{stab}})$ is also a monochromatic edge of $S(n, k)$, so we return it (see lines 4–6).

If the output is of type (b), then we are given a vertex $A \in \binom{X_l}{k}_{\text{stab}}$ satisfying $c(A) = i_r \notin C_l$ for some $r < l$. Since $i_r \notin C_l$, it follows that $j_r \notin X_l$, and thus $j_r \notin A$. In this case, we pick uniformly and independently $18n$ random sets from $\binom{X_r}{k}_{\text{stab}}$ and query the oracle for their colors. If we find a vertex B that forms together with A a monochromatic edge in $S(n, k)$, we return the monochromatic edge $\{A, B\}$, and otherwise we declare “failure” (see lines 7–15).

If the output of Algorithm 1 is a pair (i_l, j_l) of type (c), then we define, as mentioned above, the sets $X_{l+1} = X_l \setminus \{j_l\}$ and $C_{l+1} = C_l \setminus \{i_l\}$ (see lines 16–18). Observe that for $0 \leq l < s$, it holds that $|X_l| = n - l > n - s = 10k^4$, allowing us, by Theorem 12, to call Algorithm 1 in the l th iteration.

In case that all the s calls to Algorithm 1 return an output of type (c), we arrive to the second phase of the algorithm. Here, we are given the sets X_s and C_s that satisfy $|X_s| = n - s \leq 10k^4$ and $|C_s| = |X_s| - 2k + 1$, and we query the oracle for the colors of each and every vertex of the graph $K\left(\binom{X_s}{k}_{\text{stab}}\right)$. If we find a vertex $A \in \binom{X_s}{k}_{\text{stab}}$ satisfying $c(A) = i_r \notin C_s$ for some $r < s$, then, as before, we pick uniformly and independently $18n$ random sets from $\binom{X_r}{k}_{\text{stab}}$ and query the oracle for their colors. If we find a vertex B that forms together with A a monochromatic edge in $S(n, k)$, we return the monochromatic edge $\{A, B\}$, and otherwise we declare “failure” (see lines 21–28). Otherwise, all the vertices of $K\left(\binom{X_s}{k}_{\text{stab}}\right)$ are colored by colors from C_s . By Theorem 3, the chromatic number of the graph $K\left(\binom{X_s}{k}_{\text{stab}}\right)$, which is isomorphic to $S(|X_s|, k)$, is $|X_s| - 2k + 2 > |C_s|$. Hence, there must exist a monochromatic edge in $K\left(\binom{X_s}{k}_{\text{stab}}\right)$, and by checking all the pairs of its vertices we find such an edge and return it (see lines 30–31).

We turn to analyze the probability that Algorithm 2 returns a monochromatic edge. Note that whenever the algorithm returns an edge, it checks that it is monochromatic and thus ensures that it forms a valid solution. Hence, it suffices to show that the algorithm declares “failure” with probability at most $1/2$. To see this, recall that the algorithm calls Algorithm 1 at most $s < n$ times, and that by Theorem 12 the probability that its output is not valid is at most $\varepsilon = \frac{1}{4n}$. By the union bound, the probability that any of the calls to Algorithm 1 returns an invalid output does not exceed $1/4$. The only situation in which Algorithm 2 declares “failure” is when it finds, for some $r < s$, a vertex $A \in \binom{[n]}{k}_{\text{stab}}$ with $c(A) = i_r$ and $j_r \notin A$, and none of the $18n$ sampled sets $B \in \binom{X_r}{k}_{\text{stab}}$ satisfies $c(B) = i_r$ and $A \cap B = \emptyset$ (see lines 7–15, 21–28). However, assuming that all the calls to Algorithm 1 return valid outputs, the r th run guarantees, by Theorem 12, that a random vertex B uniformly chosen from $\binom{X_r}{k}_{\text{stab}}$ satisfies $c(B) = i_r$ and $A \cap B = \emptyset$ for the given A with probability at least $\frac{1}{9n}$. Hence, the probability that the algorithm declares “failure” does not exceed $(1 - \frac{1}{9n})^{18n} \leq e^{-2} < \frac{1}{4}$. Using again the union bound, it follows that the probability that Algorithm 2 either gets an invalid output from Algorithm 1 or fails to find a vertex that forms a monochromatic edge with a set A as above is at most $1/2$. Therefore, the probability that Algorithm 2 successfully finds a monochromatic edge is at least $1/2$, as desired.

We finally analyze the running time of Algorithm 2. In its first phase, the algorithm calls Algorithm 1 at most $s < n$ times, where the running time needed for each call is, by Theorem 12 and by our choice of ε , polynomial in n . It is clear that the other operations made throughout this phase can also be implemented in time polynomial in n . In its second phase, the algorithm enumerates all the vertices of $K\left(\binom{X_s}{k}_{\text{stab}}\right)$. This phase can be implemented in running time polynomial in n and in the number of vertices of this graph. The latter is $\left|\binom{X_s}{k}_{\text{stab}}\right| \leq |X_s|^k \leq (10k^4)^k = k^{O(k)}$. It thus follows that the total running time of Algorithm 2 is $n^{O(1)} \cdot k^{O(k)}$, completing the proof. ◀

References

- 1 Imre Bárány. A short proof of Kneser’s conjecture. *J. Comb. Theory, Ser. A*, 25(3):325–326, 1978.
- 2 Karol Borsuk. Drei Sätze über die n -dimensionale euklidische Sphäre. *Fundamenta Mathematicae*, 20(1):177–190, 1933.
- 3 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 4 Argyrios Deligkas, John Fearnley, Alexandros Hollender, and Themistoklis Melissourgos. Constant inapproximability for PPA. In *Proc. of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC’22)*, pages 1010–1023, 2022.
- 5 Xiaotie Deng, Zhe Feng, and Rucha Kulkarni. Octahedral Tucker is PPA-complete. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:118, 2017.

16:16 A Fixed-Parameter Algorithm for the Schrijver Problem

- 6 Paul Erdős, Chao Ko, and Richard Rado. Intersection theorems for systems of finite sets. *Quart. J. Math.*, 12(1):313–320, 1961.
- 7 Aris Filos-Ratsikas and Paul W. Goldberg. Consensus halving is PPA-complete. In *Proc. of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC'18)*, pages 51–64, 2018.
- 8 Aris Filos-Ratsikas and Paul W. Goldberg. The complexity of splitting necklaces and bisecting ham sandwiches. In *Proc. of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC'19)*, pages 638–649, 2019.
- 9 Aris Filos-Ratsikas, Alexandros Hollender, Katerina Sotiraki, and Manolis Zampetakis. A topological characterization of modulo- p arguments and implications for necklace splitting. In *Proc. of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA'21)*, pages 2615–2634, 2021.
- 10 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.
- 11 Peter Frankl and Andrey Kupavskii. Maximal degrees in subgraphs of Kneser graphs. *arXiv*, abs/2004.08718, 2020. [arXiv:2004.08718](https://arxiv.org/abs/2004.08718).
- 12 David Gale. Neighboring vertices on a convex polyhedron. In H. W. Kuhn and A.W. Tucker, editors, *Linear Inequalities and Related Systems*, volume 38 of *Annals of Math. Studies*, pages 255–263. Princeton University Press, 1956.
- 13 Paul W. Goldberg, Alexandros Hollender, Ayumi Igarashi, Pasin Manurangsi, and Warut Suksompong. Consensus halving for sets of items. In *Proc. 16th Web and Internet Economics International Conference (WINE'20)*, pages 384–397, 2020.
- 14 Ishay Haviv. The complexity of finding fair independent sets in cycles. In *12th Innovations in Theoretical Computer Science Conference (ITCS'21)*, pages 4:1–4:14, 2021.
- 15 Ishay Haviv. A fixed-parameter algorithm for the Kneser problem. In *49th International Colloquium on Automata, Languages, and Programming (ICALP'22)*, pages 72:1–72:18, 2022.
- 16 Anthony J. W. Hilton and Eric Charles Milner. Some intersection theorems for systems of finite sets. *Quart. J. Math.*, 18(1):369–384, 1967.
- 17 Charles R. Hobby and John R. Rice. A moment problem in L_1 approximation. *Proc. Amer. Math. Soc.*, 16(4):665–670, 1965.
- 18 Gabriel Istrate, Cosmin Bonchis, and Adrian Craciun. Kernelization, proof complexity and social choice. In *48th International Colloquium on Automata, Languages, and Programming (ICALP'21)*, pages 135:1–135:21, 2021.
- 19 Martin Kneser. Aufgabe 360. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 58(2):27, 1955.
- 20 László Lovász. Kneser's conjecture, chromatic number, and homotopy. *J. Comb. Theory, Ser. A*, 25(3):319–324, 1978.
- 21 Pasin Manurangsi and Warut Suksompong. Computing a small agreeable set of indivisible items. *Artif. Intell.*, 268:96–114, 2019. Preliminary versions in IJCAI'16 and IJCAI'17.
- 22 Jiří Matoušek. *Using the Borsuk-Ulam Theorem: Lectures on Topological Methods in Combinatorics and Geometry*. Springer Publishing Company, Incorporated, 2007.
- 23 Colin McDiarmid. Concentration. In *Probabilistic methods for algorithmic discrete mathematics*, volume 16 of *Algorithms Combin.*, pages 195–248. Springer, Berlin, 1998.
- 24 Nimrod Megiddo and Christos H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theor. Comput. Sci.*, 81(2):317–324, 1991.
- 25 Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.
- 26 Alexander Schrijver. Vertex-critical subgraphs of Kneser graphs. *Nieuw Arch. Wiskd.*, 26(3):454–461, 1978.
- 27 Forest W. Simmons and Francis Edward Su. Consensus-halving via theorems of Borsuk-Ulam and Tucker. *Math. Soc. Sci.*, 45(1):15–25, 2003.
- 28 John Talbot. Intersecting families of separated sets. *J. London Math. Soc.*, 68(1):37–51, 2003.

Towards Exact Structural Thresholds for Parameterized Complexity

Falko Hegerfeld  

Humboldt-Universität zu Berlin, Germany

Stefan Kratsch  

Humboldt-Universität zu Berlin, Germany

Abstract

Parameterized complexity seeks to optimally use input structure to obtain faster algorithms for NP-hard problems. This has been most successful for graphs of low treewidth, i.e., graphs decomposable by small separators: Many problems admit fast algorithms relative to treewidth and many of them are optimal under the Strong Exponential-Time Hypothesis (SETH). Fewer such results are known for more general structure such as low clique-width (decomposition by large and dense but structured separators) and more restrictive structure such as low deletion distance to some sparse graph class.

Despite these successes, such results remain “islands” within the realm of possible structure. Rather than adding more islands, we seek to determine the transitions between them, that is, we aim for structural thresholds where the complexity increases as input structure becomes more general. Going from deletion distance to treewidth, is a single deletion set to a graph with simple components enough to yield the same lower bound as for treewidth or does it take many disjoint separators? Going from treewidth to clique-width, how much more density entails the same complexity as clique-width? Conversely, what is the most restrictive structure that yields the same lower bound?

For treewidth, we obtain both refined and new lower bounds that apply already to graphs with a single separator X such that $G - X$ has treewidth at most $r = \mathcal{O}(1)$, while G has treewidth $|X| + \mathcal{O}(1)$. We rule out algorithms running in time $\mathcal{O}^*((r + 1 - \varepsilon)^k)$ for DELETION TO r -COLORABLE parameterized by $k = |X|$; this implies the same lower bound relative to treedepth and (hence) also to treewidth. It specializes to $\mathcal{O}^*((3 - \varepsilon)^k)$ for ODD CYCLE TRANSVERSAL where $\text{tw}(G - X) \leq r = 2$ is best possible. For clique-width, an extended version of the above reduction rules out time $\mathcal{O}^*((4 - \varepsilon)^k)$, where X is allowed to be a possibly large separator consisting of k (true) twinclasses, while the treewidth of $G - X$ remains r ; this is proved also for the more general DELETION TO r -COLORABLE and it implies the same lower bound relative to clique-width. Further results complement what is known for VERTEX COVER, DOMINATING SET and MAXIMUM CUT. All lower bounds are matched by existing and newly designed algorithms.

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

Keywords and phrases Parameterized complexity, lower bound, vertex cover, odd cycle transversal, SETH, modulator, treedepth, cliquewidth

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.17

Related Version *Full Version:* <https://arxiv.org/abs/2107.06111> [27]

Funding *Falko Hegerfeld:* Partially supported by DFG Emmy Noether-grant (KR 4286/1).

1 Introduction

The goal of parameterized complexity is to leverage input structure to obtain faster algorithms than in the worst case and to identify algorithmically useful structure. The most prominent structural graph parameter *treewidth* measures the size of *separators* decomposing the graph. Many problems admit fast algorithms relative to treewidth and we can often certify their optimality assuming the *Strong Exponential-Time Hypothesis* (SETH) [7, 11, 12, 13, 43].



© Falko Hegerfeld and Stefan Kratsch;

licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 17; pp. 17:1–17:20

Leibniz International Proceedings in Informatics



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

Such (conditional) *optimality results* allow us to conduct a precise study of the impact of structure on the running time, whereas otherwise the currently best running time might be an artifact due to the momentary lack of algorithmic tools and not inherent to the structure.

The structure captured by treewidth can be varied in several ways: In the *sparse setting*, we may restrict the interplay of separators and/or allow additional connected components from some graph class \mathcal{H} ; this yields notions such as treedepth as well as deletion resp. elimination distance to \mathcal{H} . In the *dense setting*, we may allow large and dense but structured separators; this yields e.g. clique-width and rank-width. Conceptually, the difference between parameters may be quite large: if the complexity of a problem changes between two parameters, then it is difficult to pinpoint which structural feature has led to the change in complexity.

We seek to delineate more *exact structural thresholds* between these parameters. This can be done by designing algorithms relative to more permissible parameters or by establishing the same lower bounds relative to more restrictive parameters. We focus on the latter approach in a fine-grained setting, i.e., all considered problems can be solved in time $\mathcal{O}^*(c^k)^1$ for some constant c and parameter k and we determine the precise value of the base c .

For parameters other than treewidth far fewer optimality results are known. In particular, to the best of our knowledge, the only known fine-grained optimality results for NP-hard problems relative to a deletion distance are for r -COLORING [35, 43], its generalization LIST HOMOMORPHISM [52], and isolated results on VERTEX COVER [33] and CONNECTED VERTEX COVER [9]. The crux is that other lower bound proofs deal with more complex problems (e.g., deletion of vertices, packing of subgraphs, etc.) by copying the same (type of) partial solution over many *noncrossing* separators; this addresses several obstacles but makes the approach unsuitable for deletion distance parameters (or even for treedepth). We show that a much broader range of problems may admit such improved lower bounds by giving the new tight lower bounds for *vertex deletion problems* such as VERTEX COVER and ODD CYCLE TRANSVERSAL relative to deletion distance parameters, in both sparse and dense settings.

Sparse Setting. Our main problem of study is DELETION TO r -COLORABLE, i.e., delete as few vertices as possible so that an r -colorable graph remains, which specializes to VERTEX COVER for $r = 1$ and to ODD CYCLE TRANSVERSAL for $r = 2$. The first parameterization which we study is the size $|X|$ of a *modulator* $X \subseteq V(G)$, or deletion distance, to treewidth r , i.e., $\text{tw}(G - X) \leq r$. Our main result in the sparse setting is the following.

► **Theorem 1.1.** *If there are $r \geq 2$, $\varepsilon > 0$ such that DELETION TO r -COLORABLE can be solved in time $\mathcal{O}^*((r + 1 - \varepsilon)^{|X|})$, where X is a modulator to treewidth r , then SETH is false.²*

The general construction for DELETION TO r -COLORABLE, $r \geq 2$, does not work for the case $r = 1$, i.e., VERTEX COVER, and we fill this gap by providing a simple ad-hoc construction for VERTEX COVER parameterized by a modulator to pathwidth 2.

► **Theorem 1.2.** *If there is an $\varepsilon > 0$ such that VERTEX COVER can be solved in time $\mathcal{O}^*((2 - \varepsilon)^{|X|})$, where X is a modulator to pathwidth 2, then SETH is false.*

These results improve the known lower bounds for VERTEX COVER and ODD CYCLE TRANSVERSAL parameterized by pathwidth and provide new tight lower bounds for $r \geq 3$ as a matching upper bound follows from generalizing the known algorithm for ODD CYCLE TRANSVERSAL parameterized by treewidth. Note that in Theorem 1.1 the treewidth bound r

¹ The \mathcal{O}^* -notation suppresses factors that are polynomial in the input size.

² We assume that an appropriate decomposition is given, thus strengthening the lower bounds.

is the same as the bound r on the number of colors. This treewidth bound, at least for $r = 2$, and the pathwidth bound in Theorem 1.2 cannot be improved due to upper bounds obtained by Lokshtanov et al. [44] for VERTEX COVER and ODD CYCLE TRANSVERSAL parameterized by an odd cycle transversal or a feedback vertex set. Lokshtanov et al. [43] asked if the complexity of problems, other than r -COLORING (where a modulator to a single path is already sufficient [35]), relative to treewidth could already be explained with parameterization by feedback vertex set. As argued, this cannot be true for VERTEX COVER and ODD CYCLE TRANSVERSAL, so our results are essentially the next best explanation.

Furthermore, the previous two theorems also imply the same lower bound for parameterization by *treedepth*³, thus yielding the first tight lower bounds relative to treedepth for vertex selection problems and partially resolving a question of Jaffke and Jansen [35] regarding the complexity relative to treedepth for problems studied by Lokshtanov et al. [43].

► **Corollary 1.3.** *If there is an $r \geq 1$ and an $\varepsilon > 0$ such that DELETION TO r -COLORABLE can be solved in time $\mathcal{O}^*((r + 1 - \varepsilon)^{\text{td}(G)})$, then SETH is false.*

Dense Setting. Our results on deletion distances can actually be lifted to the *dense setting*. We do so by considering *twinclasses*, which are arguably the simplest form of dense structure. A twinclass is an equivalence class of the *twin*-relation, which says that two vertices u and v are twins if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$, i.e., u and v have the same neighborhood outside of $\{u, v\}$. Given two distinct twinclasses, either all edges between them exist or none of them do. Contracting each twinclass yields the *quotient graph* G^q and we obtain *twinclass-variants* of the usual graph parameters treedepth, cutwidth, pathwidth, and treewidth by measuring these parameters on the quotient graph G^q , e.g., the *twinclass-pathwidth* of G is $\text{tc-pw}(G) = \text{pw}(G^q)$. The parameters twinclass-pathwidth and twinclass-treewidth have been studied before under the name *modular pathwidth* and *modular treewidth* [42, 47, 51]. Furthermore, we remark that the previously studied parameter *neighborhood diversity* satisfies $\text{nd}(G) = |V(G^q)|$ [41]. Relationships between parameters transfer to their twinclass-variants and twinclass-pathwidth is more restrictive than *linear-clique-width*. Similarly, we obtain *twinclass-modulators*, but we measure the complexity of the remaining components on the level of the original graph, i.e., a twinclass-modulator (TCM) \mathcal{X} to treewidth r is a family \mathcal{X} of twinclasses such that $\text{tw}(G - \bigcup_{X \in \mathcal{X}} X) \leq r$. We can now state our second main result, which, similarly to the sparse setting, also carries over to twinclass-treedepth.

► **Theorem 1.4.** *If there are $r \geq 2$, $\varepsilon > 0$ such that DELETION TO r -COLORABLE can be solved in time $\mathcal{O}^*((2^r - \varepsilon)^{|\mathcal{X}|})$, where \mathcal{X} is a TCM to treewidth r , then SETH is false.*

Additionally, it follows that if there are $r \geq 2$, $\varepsilon > 0$ such that DELETION TO r -COLORABLE can be solved in time $\mathcal{O}^((2^r - \varepsilon)^{\text{tc-td}(G)})$, then SETH is false.*

Due to the inequalities $\text{cw}(G) \leq \text{tc-pw}(G) + 3$ and $\text{pw}(G) \leq \text{td}(G)$, cf. Lampis [42] and Nešetřil and Ossona de Mendez [48], we see that $\text{cw}(G) \leq \text{tc-td}(G) + 3$. Hence any $\mathcal{O}^*(c^{\text{cw}(G)})$ -time algorithm also implies a $\mathcal{O}^*(c^{\text{tc-td}(G)})$ -time algorithm. Thus, the following result, relying on standard techniques for dynamic programming on graph decompositions such as the $(\min, +)$ -cover product, yields a tight upper bound complementing the previous lower bounds.

³ If $\text{tw}(G - X) \leq t$, then $\text{td}(G) \leq |X| + (t + 1) \log_2 |V|$, cf. Nešetřil and Ossona de Mendez [48], and $\mathcal{O}^*(c^{\text{td}(G)}) = \mathcal{O}^*(c^{|X|} |V|^{(t+1) \log_2 c}) = \mathcal{O}^*(c^{|X|})$.

► **Theorem 1.5.** *Given a k -clique-expression μ for G , DELETION TO r -COLORABLE on G can be solved in time $\mathcal{O}^*((2^r)^k)$.⁴*

There is no further lower bound result for VERTEX COVER, since $r + 1 = 2^r$ for $r = 1$ and hence Theorem 1.2 already yields a tight lower bound for the clique-width-parameterization.

Going into more detail, the twinclasses of the modulator in the construction for Theorem 1.4 are *true twinclasses*, i.e., each twinclass induces a clique, and moreover they are of size r (with a small exception). Intuitively, allowing for deletions, there are 2^r possible sets of at most r colors that can be assigned to a clique of size r , e.g., the empty set \emptyset corresponds to deleting the clique completely. Hence, our results essentially show that it is necessary and optimal to go through all of these color sets for each twinclass in the modulator.

In contrast, consider the situation for r -COLORING where Lampis [42] has obtained tight running times of $\mathcal{O}^*\left(\binom{r}{\lfloor r/2 \rfloor}^{\text{tc-tw}(G)}\right)$ when parameterized by twinclass-treewidth and of time $\mathcal{O}^*((2^r - 2)^{\text{cw}(G)})$ when parameterized by clique-width. Whereas the complexities for r -COLORING vary between the twinclass-setting and clique-width, this is not the case for DELETION TO r -COLORABLE. The base $\binom{r}{\lfloor r/2 \rfloor}$ is due to the fact that without deletions only color sets of the same size as the considered (true) twinclass can be attained and the most sets are possible when the size is $\lfloor r/2 \rfloor$. For clique-width, a *label class* may induce more complicated graphs than cliques or independent sets and the interaction between two label classes may also be more intricate. Lampis [42] shows that the extremal cases of color sets \emptyset and $[r] = \{1, \dots, r\}$ can be handled separately, thus yielding the base $2^r - 2$ for clique-width.

Additional results. As separate results, we obtain the following four results:

- **Theorem 1.6.** *Assuming the SETH, the following lower bounds hold:*
- *DOMINATING SET cannot be solved in time $\mathcal{O}^*((4 - \varepsilon)^{\text{tc-ctw}(G)})$ for any $\varepsilon > 0$.*
 - *TOTAL DOMINATING SET cannot be solved in time $\mathcal{O}^*((4 - \varepsilon)^{\text{ctw}(G)})$ for any $\varepsilon > 0$.*
 - *MAXIMUM CUT cannot be solved in time $\mathcal{O}^*((2 - \varepsilon)^{|X|})$ for any $\varepsilon > 0$, where X is a modulator to treewidth at most 2.*
 - *K_r -FREE DELETION cannot be solved in time $\mathcal{O}^*((2 - \varepsilon)^{|X|})$ for any $\varepsilon > 0$ and $r \geq 3$, where X is a modulator to treewidth at most $r - 1$.*

The first result improves the parameterization of the tight lower bound for DOMINATING SET obtained by Katsikarelis et al. [38] from linear-clique-width to twinclass-cutwidth. We prove this by reducing TOTAL DOMINATING SET parameterized by cutwidth to DOMINATING SET parameterized by twinclass-cutwidth and providing a lower bound construction for TOTAL DOMINATING SET parameterized by cutwidth.

Lastly, the lower bound for VERTEX COVER, Theorem 1.2, also implies tight lower bounds for MAXIMUM CUT and K_r -FREE DELETION, which again imply the same lower bounds parameterized by treedepth. The former also partially answers a question of Jaffke and Jansen [35], by being another problem considered by Lokshtanov et al. [43] whose running time cannot be improved when parameterizing by treedepth instead of treewidth.

⁴ Jacob et al. [34] have simultaneously proven this upper and lower bound for the special case of ODD CYCLE TRANSVERSAL, $r = 2$, parameterized by clique-width. Their construction also proves the lower bound for linear-clique-width, but not for the more restrictive twinclass-treedepth or twinclass-modulator like our construction.

Technical contribution. We start by recalling the standard approach of Lokshtanov et al. [43] to proving tight lower bounds for problems parameterized by pathwidth at a high level. Given a SATISFIABILITY instance σ , the variables are partitioned into t groups of constant size. For each variable group, a *group gadget* is constructed that can encode all assignments of this variable group into partial solutions of the considered target problem. The group gadget usually consists of a bundle of long path-like gadgets inducing a sequence of *disjoint* separators. Further gadgets attached to these separators decode the partial solutions and check whether the corresponding assignment satisfies some clause. Ideally, the path gadgets are designed so that a partial solution transitions through a well-defined sequence of states when viewed at consecutive separators. For most problems, the gadgets do not behave this nicely though. For example, in ODD CYCLE TRANSVERSAL it is locally always preferable to delete a vertex instead of not deleting it. Such behavior leads to undesired state changes called *cheats*, but for appropriate path gadgets there can only be a constant number of cheats on each path. By making the path gadgets long enough, one can then find a region containing no cheats where we can safely decode the partial solutions.

For problems such as r -COLORING, all states are equally constraining and such cheats do not occur, hence enabling us to prove the same lower bounds under more restrictive parameters such as feedback vertex set. But for *vertex deletion problems*, like ODD CYCLE TRANSVERSAL, these cheats do occur and pose a big issue when trying to compress the path gadgets into a single separator X , since deletions in X are highly favorable. On a single separator X such behavior means that one partial solution is *dominating* another and if we cannot control this behavior, then we lose the dominated partial solution for the purpose of encoding group assignments. Concretely, for ODD CYCLE TRANSVERSAL we obtain dominating partial solutions by deleting further vertices in the single separator X . The number of deletions is bounded from above by the *budget constraint*, but if we limit the number of deletions in X , then we do not have $3^{|X|}$ partial solutions anymore and the construction may not be able to attain the desired base in the running time.

To resolve this issue we expand upon a technique of Cygan et al. [9] and construct an instance with a slightly large parameter value, i.e., a slightly larger single separator X . Thus, we can limit the number of deletions and are still able to encode sufficiently many group assignments. More precisely, we consider only partial solutions with the same number of deletions in X , hence only pairwise non-dominating partial solutions remain. We construct a *structure gadget* to enforce a lower bound on the number of deletions in X . A positive side effect is that the remaining gadgets can also leverage the structure of the partial solutions.

In the dense setting and especially for a higher number r of colors, this issue is amplified. Here, we consider the states of twinclasses, instead of single vertices, in a partial solution. For a twinclass, there is a hierarchy of dominating states: any state that does not delete all vertices in the twinclass is dominated by a state that deletes further vertices in the twinclass. For DELETION TO r -COLORABLE, the maximum number of states is achieved on a true twinclass of size r and we can partition the states into *levels* based on the number of deletions they induce. Within each level, the states are pairwise non-dominating. Consequently, we restrict the family of partial solutions so that for every level the number of twinclasses with that level is fixed. This requires a considerably more involved construction of the structure gadget which now has to distinguish states based on their level.

Related work. There is a long line of work relative to treewidth [2, 7, 11, 12, 15, 16, 17, 21, 39, 43, 46, 45, 49, 50] and all of these lower bounds, except for the result by Egri et al. [17], already apply to pathwidth. In the sparse setting, there is further work on the parameterization by

cutwidth [8, 26, 37, 45, 52, 53] and by feedback vertex set [43, 52]. We remark that the works of van Geffen et al. [53] and Piecyk and Rzażewski [52] show that previous lower bounds relative to pathwidth already hold for more restrictive parameterizations. In the dense setting, there are some results [32, 34, 38, 42] on parameterization by clique-width and these lower bounds already apply to linear-clique-width, but not to the more restrictive parameters that we consider. The work by Iwata and Yoshida [32] also provides equivalences between different lower bounds and works under a weaker assumption than SETH, unfortunately their techniques blow up the modulator too much and are not applicable in our case. Finally, the complexity of r -COLORING and the more general homomorphism problems has been extensively studied [17, 21, 24, 35, 42, 49, 50, 52], only two of these articles [24, 42] consider the dense setting. Jaffke and Jansen [35] closely study the complexity of r -COLORING parameterized by the deletion distance to various graph classes \mathcal{F} ; in particular, the base for treewidth can already be explained by deletion distance to a single path.

On the algorithmic side, the study of heterogeneous parameterizations has been gaining traction [3, 4, 18, 20, 19, 29, 36], yielding the notions of \mathcal{H} -treewidth and \mathcal{H} -elimination distance, which is a generalization of treedepth. Currently, only few of these works [18, 36] contain algorithmic results that are sufficiently optimized to apply to our fine-grained setting. Jansen et al. [36] show that VERTEX COVER can be solved in time $\mathcal{O}^*(2^k)$ and ODD CYCLE TRANSVERSAL in time $\mathcal{O}^*(3^k)$ when parameterized by bipartite-treewidth. Eiben et al. [18] show that MAXIMUM CUT can be solved in time $\mathcal{O}^*(2^k)$ when parameterized by \mathcal{R}_w -treewidth, where \mathcal{R}_w denotes the graphs of rank-width at most w .

Another line of work is on depth-parameters in the dense setting [5, 14, 22, 23, 25, 28, 40] such as *shrub-depth* and *sc-depth*. The algorithmic results relative to these parameters are largely concerned with meta-results so far [5, 25] and their relation to clique-width is not strong enough to preserve the complexity in our fine-grained setting.

Organization. We discuss the preliminaries and basic notation in Section 2. The relationships between the considered parameters are discussed in Section 3. In Section 4, we give an outline of our two main results: the lower bound for DELETION TO r -COLORABLE in the sparse setting and the dense setting. The algorithm for DELETION TO r -COLORABLE parameterized by clique-width is given in Section 5. We conclude in Section 6. Appendix A contains the formal definitions of the considered problems. The remaining results, including the missing proofs, can be found in the full version of the paper [27].

2 Preliminaries

If n is a positive integer, we define $[n] = \{1, \dots, n\}$. If S is a set, we define $\mathcal{P}(S) = \{T \subseteq S\}$ and if $0 \leq k \leq |S|$, we define $\binom{S}{k} = \{T \subseteq S : |T| = k\}$ and $\binom{S}{\leq k} = \{T \subseteq S : |T| \leq k\}$ and $\binom{S}{\geq k}$ analogously. If $0 \leq k \leq n$, we define $\binom{n}{\leq k} = |\binom{[n]}{\leq k}|$ and similarly $\binom{n}{\geq k}$. If \mathcal{S} is a set family, we define $\bigcup(\mathcal{S}) = \bigcup_{S \in \mathcal{S}} S$. If $f: A \rightarrow C$ is a function and $B \subseteq A$, then $f|_B$ denotes the *restriction* of f to B . If $f, g: A \rightarrow B$ are two functions, we write $f \equiv g$ if $f(a) = g(a)$ for all $a \in A$. If p is a boolean predicate, we let $[p]$ denote the *Iverson bracket* of p , which is 1 if p is true and 0 if p is false.

We use common graph-theoretic notation and assume that the reader knows the essentials of parameterized complexity. Let $G = (V, E)$ be an undirected graph. For a vertex set $X \subseteq V$, we denote by $G[X]$ the subgraph of G that is induced by X . The *open neighborhood* of a vertex v is given by $N(v) = \{u \in V : \{u, v\} \in E\}$, whereas the *closed neighborhood* is given by

$N[v] = N(v) \cup \{v\}$. For sets $X \subseteq V$ we define $N[X] = \bigcup_{v \in X} N[v]$ and $N(X) = N[X] \setminus X$. For two disjoint vertex subsets $A, B \subseteq V$, adding a *join* between A and B means adding all edges between A and B . For a vertex set $X \subseteq V$, we define $\delta(X) = \{\{x, y\} \in E : x \in X, y \notin X\}$.

An r -*coloring* of a graph $G = (V, E)$ is a function $\varphi: V \rightarrow [r]$ such that $\varphi(u) \neq \varphi(v)$ for all $\{u, v\} \in E$. We say that G is r -*colorable* if there is an r -coloring of G . The *chromatic number* of G , denoted by $\chi(G)$, is the minimum r such that G is r -colorable.

Quotients and twins. Let Π be a partition of $V(G)$. The *quotient graph* G/Π is given by $V(G/\Pi) = \Pi$ and $E(G/\Pi) = \{\{B_1, B_2\} : \exists u \in B_1, v \in B_2 : \{u, v\} \in E(G)\}$. We say that two vertices u, v are *twins* if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. The equivalence classes of this relation are called *twinclasses*. More specifically, if $N(u) = N(v)$, then u and v are *false twins* and if $N[u] = N[v]$, then u and v are *true twins*. Every twinclass of size at least 2 consists of only false twins or only true twins. A false twinclass induces an independent set and a true twinclass induces a clique. Let $\Pi_{tc}(G)$ be the partition of $V(G)$ into twinclasses.

2.1 Graph Parameters

Sparse Parameters. The definition of treewidth, pathwidth, treedepth, and cutwidth are standard and can be found in the full version. We will construct graphs that have small treewidth except for one central part. This structure is captured by the concept of a *modulator*. We say that $X \subseteq V(G)$ is a *modulator to treewidth/pathwidth r* for G if $\text{tw}(G - X) \leq r$ or $\text{pw}(G - X) \leq r$, respectively.

Lifting to Twinclasses

We define the *twinclass-treewidth*, *twinclass-pathwidth*, *twinclass-treedepth*, and *twinclass-cutwidth* of G by $\text{tc-tw}(G) = \text{tw}(G/\Pi_{tc}(G))$, $\text{tc-pw}(G) = \text{pw}(G/\Pi_{tc}(G))$, $\text{tc-td}(G) = \text{td}(G/\Pi_{tc}(G))$, and $\text{tc-ctw}(G) = \text{ctw}(G/\Pi_{tc}(G))$, respectively. The parameters twinclass-treewidth and twinclass-pathwidth have been considered before under the name modular treewidth and modular pathwidth [42, 47, 51]. We prefer to use the prefix twinclass instead of modular to distinguish from the case where one works with the quotient graph arising from the *modular partition* of G .

► **Definition 2.1.** Let $G = (V, E)$ be a graph. A *twinclass-modulator (TCM)* $\mathcal{X} \subseteq \Pi_{tc}(G)$ of G to treewidth r is a set of twinclasses of G such that $\text{tw}(G - \bigcup(\mathcal{X})) \leq r$. The *size* of a twinclass-modulator \mathcal{X} is $|\mathcal{X}|$, i.e., the number of twinclasses \mathcal{X} contains.

Clique-Width

A *labeled graph* is a graph $G = (V, E)$ together with a label function $\ell: V \rightarrow \mathbb{N} = \{1, 2, 3, \dots\}$. We say that a labeled graph is k -*labeled* if $\ell(v) \leq k$ for all $v \in V$. For a label i , we denote by G^i the subgraph induced by the vertices with label i , i.e. $G^i = G[\ell^{-1}(i)]$. We consider the following three operations on labeled graphs: the *union*-operation $\text{union}(G_1, G_2)$ constructs the disjoint union of two labeled graphs G_1 and G_2 ; the *relabel*-operation $\text{lab}_{i \rightarrow j}(G)$ changes the label of all vertices in G with label i to label j ; the *join*-operation $\text{join}_{i,j}(G)$, $i \neq j$, adds all possible edges between vertices in G with label i and vertices in G with label j . As a base case, we have the *introduce*-operation $\text{in}_i(v)$ which constructs a single-vertex graph whose unique vertex v has label i . A valid expression that only consists of introduce-, union-, relabel-, and join-operations is called a *clique-expression*. The labeled graph constructed by a clique-expression μ is denoted $G(\mu)$. To a clique-expression μ we associate a syntax tree T_μ

in the natural way and to each node $t \in V(\mathcal{T}_\mu)$ the corresponding operation. For any node $t \in V(\mathcal{T}_\mu)$, the subtree rooted at t induces a subexpression μ_t and we define $G_t = G(\mu_t)$ as the labeled graph constructed by μ_t .

We say that a clique-expression σ is a k -clique-expression or just k -expression if G_t is k -labeled for all $t \in V(\mathcal{T}_\mu)$. The *clique-width* of a graph G , denoted by $\text{cw}(G)$, is the minimum k such that there exists a k -expression μ such that G is isomorphic to $G(\mu)$ after forgetting the labels. A clique-expression μ is *linear* if in every union-operation the second graph consists only of a single vertex. Accordingly, we also define the *linear-clique-width* of a graph G , denoted $\text{lin-cw}(G)$, by only considering linear clique-expressions.

2.2 Strong Exponential-Time Hypothesis

For our lower bounds, we assume the *Strong Exponential-Time Hypothesis* (SETH) [31] which concerns the complexity of q -SATISFIABILITY, i.e., SATISFIABILITY where all clauses contain at most q literals. Let $c_q = \inf\{\delta : q\text{-SATISFIABILITY can be solved in time } \mathcal{O}(2^{\delta n})\}$ for all $q \geq 3$. The weaker *Exponential-Time Hypothesis* (ETH) of Impagliazzo and Paturi [30] posits that $c_3 > 0$, whereas the Strong Exponential-Time Hypothesis states that $\lim_{q \rightarrow \infty} c_q = 1$. When proving lower bounds based on SETH, we make use of the following equivalent formulations.

► **Theorem 2.2** ([9]). *The following statements are equivalent to SETH:*

1. *For all $\delta < 1$, there is a clause size q such that q -SATISFIABILITY cannot be solved in time $\mathcal{O}(2^{\delta n})$, where n is the number of variables.*
2. *For all $\delta < 1$, there is a set size q such that q -HITTING SET, i.e., all sets contain at most q elements, cannot be solved in time $\mathcal{O}(2^{\delta n})$, where n is the universe size.*

3 Relations between Parameters

In this section we discuss the relationships between the parameters considered in this article.

► **Lemma 3.1** ([1], Chapter 6 of [48]). *For any graph G , we have that $\text{tw}(G) \leq \text{pw}(G) \leq \text{td}(G) - 1$, $\text{td}(G) \leq (\text{tw}(G) + 1) \log_2 |V(G)|$, $\text{tw}(G) \leq \text{pw}(G) \leq \text{ctw}(G)$, and $\text{td}(G) \leq \text{td}(G - v) + 1$ for any vertex $v \in V(G)$. These inequalities come with algorithms that can transform the appropriate decomposition in polynomial time.*

► **Corollary 3.2.** *For any graph $G = (V, E)$ and $c, r \in \mathbb{N}$, if there is a modulator $X \subseteq V$ to treewidth r , i.e., $\text{tw}(G - X) \leq r$, then we have that $\text{td}(G) \leq |X| + (r + 1) \log_2 |V|$. In particular, we have that $\mathcal{O}^*(c^{\text{td}(G)}) \leq \mathcal{O}^*(c^{|X|})$ for all $c \geq 1$. The decompositions can be transformed in polynomial time.*

Proof. Let X be a modulator to treewidth r for G . By Lemma 3.1, we see that $\text{td}(G - X) \leq (r + 1) \log_2 |V|$ for $G - X$. By repeatedly invoking the inequality $\text{td}(G) \leq \text{td}(G - v) + 1$ for $v \in X$, we obtain $\text{td}(G) \leq |X| + (r + 1) \log_2 |V|$. To see the claim regarding the \mathcal{O}^* -notation, we compute $\mathcal{O}^*(c^{\text{td}(G)}) = \mathcal{O}^*(c^{|X|} |V|^{(r+1) \log_2 c}) = \mathcal{O}^*(c^{|X|})$. ◀

► **Theorem 3.3.** *Let $G = (V, E)$ be a graph. We have the following two chains of inequalities:*

$$\begin{aligned} \text{cw}(G) &\leq \text{lin-cw}(G) \leq \text{tc-pw}(G) + 3 \leq \text{tc-td}(G) + 2 \leq \text{td}(G) + 2, \\ \text{cw}(G) &\leq \text{lin-cw}(G) \leq \text{tc-pw}(G) + 3 \leq \text{tc-ctw}(G) + 3 \leq \text{ctw}(G) + 3. \end{aligned}$$

Proof. Follows from [42, Lemma 2.1], Lemma 3.1 and the last inequalities in both rows follow from the fact that $G/\Pi_{tc}(G)$ is a subgraph of G and that treedepth and cutwidth are subgraph-monotone. ◀

► **Lemma 3.4.** *Suppose that G admits a TCM \mathcal{X} to treewidth r , then $tc\text{-}td(G) \leq |\mathcal{X}| + (r + 1)\log_2 |V(G/\Pi_{tc}(G))|$. In particular, we have for any $c \geq 1$ that $\mathcal{O}^*(c^{tc\text{-}td(G)}) \leq \mathcal{O}^*(c^{|\mathcal{X}|})$. The decompositions can be transformed in polynomial time.*

Proof. Since $G/\Pi_{tc}(G) - \mathcal{X}$ is an induced subgraph of $G - \bigcup(\mathcal{X})$, we see that $tw(G/\Pi_{tc}(G) - \mathcal{X}) \leq tw(G - \bigcup(\mathcal{X})) \leq r$. The remainder of the proof is analogous to Corollary 3.2 by working on the quotient graph $G/\Pi_{tc}(G)$. ◀

4 Outline of Main Result

We outline our two main results, i.e., tight lower bounds for DELETION TO r -COLORABLE parameterized by a (twinclass-)modulator to treewidth r . Conceptually, the constructions for the sparse setting and for the dense setting are similar. The most significant change is in the *structure gadget*, since we have to enforce a considerably more involved structure in the dense setting. We give an overview of both settings and go into more detail for the dense case.

We fix the number of colors $r \geq 2$. *Solutions* are functions $\varphi: V(G) \rightarrow [r] \cup \{\perp\}$ so that for every edge $\{u, v\} \in E(G)$ either $\varphi(u) = \varphi(v) = \perp$ or $\varphi(u) \neq \varphi(v)$. Hence, $\varphi^{-1}(\perp)$ is the set of deleted vertices, whereas $\varphi|_{V(G) \setminus \varphi^{-1}(\perp)}$ is an r -coloring of the remaining graph.

In both settings we want to simulate a *logical OR* constraint. For ODD CYCLE TRANSVERSAL, i.e. $r = 2$, we can use *odd cycles*. For $r \geq 3$, Theorem 4.1 provides an analogue, where a graph H is $(r + 1)$ -critical if $\chi(H) = r + 1$ and $\chi(H - v) = r$ for all $v \in V(H)$.

► **Theorem 4.1** (proof in full version). *There exists a family \mathcal{H}^r of $(r + 1)$ -critical graphs with treewidth r such that for every $s \in \mathbb{N}$, there exists a graph $H \in \mathcal{H}^r$ with $s \leq |V(H)| \leq s + r$.*

Setup. Given a q -SATISFIABILITY instance σ with n variables and m clauses, we start with the following standard step [43]: we partition the variables into $t = \lceil n/p_0 \rceil$ groups of size p_0 , where p_0 only depends on the running time base that we want to rule out. Furthermore, we pick an integer p depending on p_0 that represents the size of the groups in the graph.

4.1 Sparse Setting

Central vertices and solution structure. We construct a graph G that has a solution φ for DELETION TO r -COLORABLE with cost $|\varphi^{-1}(\perp)| \leq b$ if and only if σ is satisfiable. Converting from base $r + 1$ to base 2 implies that G should admit a modulator X to treewidth r of size roughly $n \log_{r+1}(2)$. Like Cygan et al. [9], we make the modulator slightly larger, thus picking a larger p . The modulator X consists of $t + 1$ vertex groups: the first t groups U_i , $i \in [t]$, are independent sets of size p each and correspond to the variable groups; the last group F is a clique of size r which simulates LIST COLORING constraints.

On each group U_i , we consider the set of partial solutions $\Phi_i = \{\varphi: U_i \rightarrow [r] \cup \{\perp\} : |\varphi^{-1}(\perp)| = p/(r + 1)\}$. By picking p large enough, Φ_i is sufficiently large to encode all assignments of the i -th variable group. Defining Φ_i in this way achieves two things: first, the solutions in Φ_i are pairwise non-dominating; secondly, this fixes the budget used on the modulator. The second point is important, because by also fixing the budget on the remaining graph via a vertex-disjoint packing \mathcal{P} of $(r + 1)$ -critical graphs, no vertex of F can be deleted, which allows us to simulate LIST COLORING constraints with the clique F .

Structure gadgets. The next step is to enforce that only the solutions in Φ_i can be attained on group U_i . By choosing the budget b appropriately, we obtain an upper bound on the number of deletions in U_i . To obtain a lower bound, we construct the *structure gadgets*. These are built by combining $(r+1)$ -critical graphs with the *arrow* gadget of Lokshtanov et al. [43]. A (thin) arrow simply propagates a deletion from a vertex u to another vertex v ; else if u is not deleted, then v is not deleted and the arrow does not affect the remaining graph.

The structure gadget works as follows: if φ deletes less than $p/(r+1)$ vertices in group U_i , then there is a subset $S \subseteq U_i$ of size $|S| = (|U_i| - p/(r+1)) + 1$ that avoids all deletions in U_i . For every subset of this size, G contains a $(r+1)$ -critical graph $L_{i,S}$ with an arrow from every $u \in S$ to a private vertex v in $L_{i,S}$, hence simulating an OR on the vertices in S . Since S avoids all deletions of φ , no deletion is propagated to $L_{i,S}$ and φ must pay extra to resolve $L_{i,S}$. By copying each $L_{i,S}$ sufficiently often, we can ensure that the existence of a deletion-avoiding S implies that φ must exceed our budget constraint.

Decode and verify. The remaining construction decodes the partial solution on the modulator X and verifies if the corresponding truth assignment satisfies all clauses of σ . One could generalize the gadgets of Lokshtanov et al. [43] to higher r , but this leads to an involved construction with a worse bound on the treewidth of the remainder: for ODD CYCLE TRANSVERSAL the construction of Lokshtanov et al. has treewidth 4, whereas the simpler construction we use has only treewidth 2. More details will be presented in the dense case.

4.2 Dense Setting

We now have a twinclass-modulator \mathcal{X} to treewidth r instead of a basic modulator and this changes the possible states as follows. Whereas φ could assume $r+1$ different states on a single vertex u , i.e., one of the r colors or deleting the vertex, there are 2^r possible states on a true twinclass U of size r ; each corresponds to a possible value of $\varphi(U) \setminus \{\perp\} \subseteq [r]$. Since U is a true twinclass, no color is used multiple times and the exact mapping $\varphi|_U$ is irrelevant.

Central twinclasses and setup. The twinclass-modulator \mathcal{X} of the constructed graph G consists of $t+1$ groups and each group is a family of twinclasses. The first t groups \mathcal{U}_i , $i \in [t]$, correspond to the variable groups and each consists of p true twinclasses of size r that are pairwise non-adjacent. The last group contains the clique F .

Solution structure. Our family Φ_i of considered partial solutions on group \mathcal{U}_i should achieve the same two things as before. First, consider the structure of states of φ on a twinclass $U \in \mathcal{U}_i$ precisely: fix a state $C = \varphi(U) \setminus \{\perp\}$ and note that all states $C' \subsetneq C$ *dominate* C if we disregard the budget constraint, i.e., φ remains a solution if we replace C by C' . After arranging the states into *levels* according to the number ℓ of deleted vertices, there is no domination between states on the same level. This motivates the following definition.

► **Definition 4.2 (informal).** Given rationals $0 < c_\ell < 1$, $\ell \in \{0\} \cup [r]$, with $\sum_{\ell=0}^r c_\ell = 1$, the set Φ_i consists of solutions φ on the family of twinclasses \mathcal{U}_i such that for every $\ell \in \{0\} \cup [r]$ there are exactly $c_\ell \cdot |\mathcal{U}_i|$ twinclasses $U \in \mathcal{U}_i$ where φ deletes exactly ℓ vertices in U .

Essentially, we are only restricting how the deletions can be distributed inside the modulator; there are no restrictions on the used colors. This again fixes the budget used on the modulator, allowing us to simulate LIST COLORING constraints with the clique F . By picking $c_\ell = \binom{r}{\ell} 2^{-r}$, $\ell \in \{0\} \cup [r]$, we ensure that Φ_i contains the solutions on \mathcal{U}_i where all 2^r states appear the

same number of times. This enables us to choose p small enough so that the time calculations work out and simultaneously large enough so that an injective mapping $\kappa_i: \{0, 1\}^{p_0} \rightarrow \Phi_i$, mapping truth assignments of the i -th variable group to solutions in Φ_i , exists.

Thick arrows and structure gadgets. To enforce the structure of Φ_i , we need a gadget to distinguish different number of deletions inside a twinclass. We can construct such a gadget $A_\ell(U, v)$, $\ell \in [r]$, also called *thick ℓ -arrow*. See Lemma 4.3 for the gadget's behavior.

► **Lemma 4.3 (informal).** *Let U be a set of r true twins and v be a vertex that is not adjacent to U and $\ell \in [r]$. There is a gadget $A = A_\ell(U, v)$ of treewidth r with the following properties:*

- *Any solution φ must delete at least ℓ vertices in $A - U$.*
- *If a solution φ deletes exactly ℓ vertices in $A - U$, then φ can only delete v if φ deletes at least ℓ vertices in U .*

We proceed by constructing the *structure gadgets* which enforce that the partial solution on \mathcal{U}_i belongs to Φ_i . Let $c_{<\ell} = c_0 + \dots + c_{\ell-1}$ for all $\ell \in \{0\} \cup [r]$. For every group $i \in [t]$, number of deletions $\ell \in [r]$, set of twinclasses $\mathcal{S} \subseteq \mathcal{U}_i$ with $|\mathcal{S}| = c_{<\ell} \cdot p + 1$, we add an $(r+1)$ -critical graph $L_{i,\ell,\mathcal{S}} \in \mathcal{H}^r$ consisting of at least $|\mathcal{S}|$ vertices. For every $U \in \mathcal{S}$, we pick a private vertex v in $L_{i,\ell,\mathcal{S}}$ and add the thick ℓ -arrow $A_\ell(U, v)$. We create a large number of copies of each $L_{i,\ell,\mathcal{S}}$ and the incident thick arrows.

The number of deletions in the central vertices is already bounded from above by the budget constraint. If too few deletions occur in the twinclasses of \mathcal{U}_i , then we can find an ℓ and an $\mathcal{S} \subseteq \mathcal{U}_i$ with $|\mathcal{S}| = c_{<\ell} \cdot p + 1$ such that less than ℓ vertices are deleted in each $U \in \mathcal{S}$. Hence, all thick ℓ -arrows leading to $L_{i,\ell,\mathcal{S}}$ and its copies cannot propagate deletions. To resolve all these $(r+1)$ -critical graphs, one extra vertex per copy must be deleted. Due to the large number of copies, this implies that we must violate our budget constraint.

Hence, for any $\mathcal{S} \subseteq \mathcal{U}_i$ with $|\mathcal{S}| = c_{<\ell} \cdot p + 1$ and any solution φ obeying the budget constraint there is at least one twinclass $U \in \mathcal{S}$ in which φ deletes at least ℓ vertices. Therefore, there are at least $(1 - c_{<\ell})p$ twinclasses in \mathcal{U}_i where φ deletes at least ℓ vertices. Since this holds for all $\ell \in \{0\} \cup [r]$ and the budget b is chosen appropriately, all inequalities have to be tight and the deletions inside \mathcal{U}_i follow the distribution imposed by Φ_i .

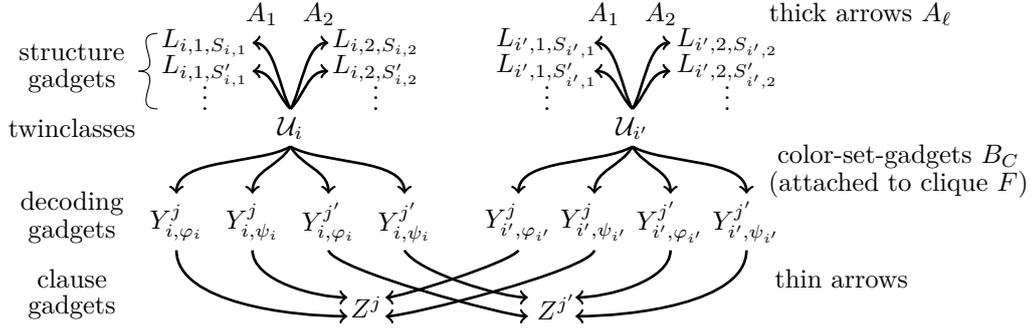
Color-set-gadgets and decoding gadgets. Next, we discuss the decoding part of the construction. Since gadgets cannot read the color of single vertices but only of a whole twinclass, we need *color-set-gadgets* to detect the colors used on a twinclass, cf. Lemma 4.4.

► **Lemma 4.4 (informal).** *Let U be a set consisting of r true twins and v be a vertex that is not adjacent to U and let $C \subsetneq [r]$. There is a gadget $B = B_C(U, v)$ of treewidth r such that:*

- *Any solution φ deletes at least $(r - |C|) + 1$ vertices in $B - U$.*
- *If φ deletes exactly $(r - |C|) + 1$ vertices in $B - U$, then $\varphi(v) = \perp$ only if $\varphi(U) \setminus \{\perp\} \subseteq C$.*

To construct the color-set-gadgets we rely on the LIST COLORING constraints that are simulated with the central clique F . Note that the color-set-gadgets only check for set inclusion and not set equality. Using the structure of solutions in Φ_i however, the color-set-gadgets will still be sufficient to distinguish the solutions in Φ_i from each other.

By using a complete $(r+1)$ -partite graph with all sets of the partition being singletons except for one large independent set, we can simulate a logical AND, see Lemma 4.5.



■ **Figure 1** An overview of the construction for the dense setting in case of $r = 2$. The arrows point in the direction that deletions are propagated by the corresponding gadget.

► **Lemma 4.5 (informal).** *Let n_Y be a positive integer. There is a gadget Y of treewidth r with a set of input vertices $V' \subseteq V(Y)$, $|V'| = n_Y$, and a vertex $\hat{y} \in V(Y) \setminus V'$ such that:*

- *Any solution φ has to delete at least one vertex in $Y - V'$.*
- *If φ deletes exactly one vertex in $Y - V'$, then $\varphi(\hat{y}) = \perp$ only if $\varphi(V') = \{\perp\}$.*

For the j -th clause, variable group $i \in [t]$, solution $\varphi_i \in \Phi_i$, we invoke Lemma 4.5 to create a gadget Y_{i,φ_i}^j for $n_Y = (1 - c_r)p = (1 - 2^{-r})p$ input vertices and with distinguished vertex \hat{y}_{i,φ_i}^j . For every twinclass $U \in \mathcal{U}_i$ with $\varphi_i(U) \neq [r]$, we pick a private input vertex v of Y_{i,φ_i}^j and add the color-set-gadget $B_{\varphi_i(U) \setminus \{\perp\}}(U, v)$. By Lemma 4.5, the vertex \hat{y}_{i,φ_i}^j can only be deleted if all input vertices of Y_{i,φ_i}^j are deleted. Due to Lemma 4.4 and the structure of Φ_i , this will only be the case if φ_i is the partial solution on \mathcal{U}_i .

Clause gadgets. For the j -th clause, we add an $(r + 1)$ -critical graph $Z^j \in \mathcal{H}^r$ consisting of at least $q2^{p_0}$ vertices. For every group $i \in [t]$ and solution $\varphi_i \in \Phi_i$ such that $\kappa_i^{-1}(\varphi_i)$ is a partial truth assignment satisfying the j -th clause, we pick a private vertex v in Z^j and add a thin arrow from \hat{y}_{i,φ_i}^j to v . The budget constraint will ensure that the only way to delete a vertex in Z^j is by propagating a deletion via a thin arrow from some \hat{y}_{i,φ_i}^j . By construction of the decoding and clause gadgets this is only possible if the partial solution on \mathcal{U}_i corresponds to a satisfying assignment of the j -th clause. This concludes the construction, cf. Figure 1.

Budget and packing. The budget $b = b_0 + \text{cost}_{\mathcal{P}}$ of the constructed instance (G, b) consists of two parts; $b_0 = trp/2$ is allocated to the central twinclasses and matches the number of deletions incurred by picking a partial solution $\varphi_i \in \Phi_i$ on \mathcal{U}_i for each group $i \in [t]$; the second part $\text{cost}_{\mathcal{P}}$ is due to a vertex-disjoint packing \mathcal{P} which we describe next. A part of each thin arrow in G is added to \mathcal{P} and for every thick arrow, color-set-gadget, or decoding gadget, we add the appropriate parts to \mathcal{P} given by Lemmas 4.3, 4.4, 4.5, respectively. Summing up the implied costs yields $\text{cost}_{\mathcal{P}}$. Hence, we know how the deletions are distributed throughout the various gadgets. In particular, this ensures that no vertex of the central clique F is deleted.

Theorem 1.4 follows by using these ideas and working out the remaining technical details.

5 Algorithm for Deletion to r -Colorable

In this section we describe how to solve DELETION TO r -COLORABLE in time $\mathcal{O}^*((2^r)^k)$ if we are given a k -expression μ for G . We perform bottom-up dynamic programming along the syntax tree \mathcal{T}_{μ} . We again view solutions to DELETION TO r -COLORABLE as functions $\varphi: V(G) \rightarrow [r] \cup \{\perp\}$ with the property discussed in the outline, cf. Section 4.

► **Theorem 5.1.** *Given a k -expression μ for G , DELETION TO r -COLORABLE on G can be solved in time $\mathcal{O}^*((2^r)^k)$.*

Proof. Let (G, b) be a DELETION TO r -COLORABLE instance and μ a k -expression for G . We can without loss of generality assume that μ consists of $\mathcal{O}(|V(G)|)$ union-operations and $\mathcal{O}(|V(G)|k^2)$ unary operations [6]. For every node $t \in V(\mathcal{T}_\mu)$ and label i , we store the set of colors used on G_t^i . After deleting the appropriate vertices, the remaining graph should be r -colorable, hence the possible color sets are precisely the subsets of $[r]$, where \emptyset indicates that all vertices are deleted. Since we use at most k labels at every node, this yields $(2^r)^k$ possible types of partial solutions at each node. If the work for each type is only polynomial, then the claimed running time immediately follows, since there are only a polynomial number of nodes in $V(\mathcal{T}_\mu)$.

For every $t \in V(\mathcal{T}_\mu)$ and $f: [k] \rightarrow \mathcal{P}([r])$, we consider the set of partial solutions

$$\mathcal{Q}_t[f] = \{\varphi: V(G_t) \rightarrow [r] \cup \{\perp\} : \varphi \text{ induces an } r\text{-coloring of } G_t - \varphi^{-1}(\perp) \text{ and} \\ \varphi(V(G_t^i)) \setminus \{\perp\} = f(i) \text{ for all } i \in [k]\}$$

and we want to compute the quantity $A_t[f] = \min\{|\varphi^{-1}(\perp)| : \varphi \in \mathcal{Q}_t[f]\}$. Let t_0 be the root node of the k -expression μ . We answer yes if there is an f such that $A_{t_0}[f] \leq b$; otherwise we answer no.

Note that $f(i) = \emptyset$ implies $\varphi(V(G_t^i)) = \{\perp\}$ for all $\varphi \in \mathcal{Q}_t[f]$, i.e., all vertices with label i are deleted. Furthermore, the definition of $\mathcal{Q}_t[f]$ implies that $\mathcal{Q}_t[f] = \emptyset$ and $A_t[f] = \infty$ whenever $|f(i)| > |V(G_t^i)|$ for some $i \in [k]$, we will not explicitly mention this edge case again in what follows and assume that the considered f satisfy $|f(i)| \leq |V(G_t^i)|$ for all $i \in [k]$. We proceed by presenting the recurrences to compute $A_t[f]$ for all t and f and afterwards show the correctness of these recurrences.

Base case. If $t = \text{in}_i(v)$ for some $i \in [k]$, then $A_t[f] = [f(i) = \emptyset]$, because the solution cost is 1 if v is deleted and 0 otherwise.

Relabel case. If $t = \text{lab}_{i \rightarrow j}(G_{t'})$ for some $i \neq j \in [k]$ and where t' is the child of t , then

$$A_t[f] = \min\{A_{t'}[f'] : f'(a) = f(a) \text{ for all } a \in [k] \setminus \{i, j\} \text{ and } f'(i) \cup f'(j) = f(j)\}.$$

By assumption, f' will always satisfy $f'(i) = \emptyset$ here, since there are no vertices with label i in $G_{t'}$. This recurrence goes over all ways how the colors $f'(j)$ used for vertices with label j in $G_{t'}$ can be split among the vertices with label i and j in the previous graph G_t . Observe that we are taking the minimum over at most $(2^r)^2 = \mathcal{O}(1)$ numbers on the right-hand side, hence this recurrence can be computed in polynomial time.

Join case. If $t = \text{join}_{i,j}(G_{t'})$ for some $i \neq j \in [k]$, where t' is the child of t , and assuming without loss of generality that $V(G_{t'}^i) \neq \emptyset$ and $V(G_{t'}^j) \neq \emptyset$, then

$$A_t[f] = \begin{cases} A_{t'}[f] & \text{if } f(i) \cap f(j) = \emptyset, \\ \infty & \text{else.} \end{cases}$$

This recurrence filters out all partial solutions where the coloring properties are not satisfied at some newly added edge. This happens precisely when $f(i) \cap f(j) \neq \emptyset$, because then there exists an edge in the join between label i and j whose endpoints get the same color.

Union case. If $t = \text{union}(G_{t_1}, G_{t_2})$ where t_1 and t_2 are the children of t , then

$$A_t[f] = \min\{A_{t_1}[f_1] + A_{t_2}[f_2] : f_1(a) \cup f_2(a) = f(a) \text{ for all } a \in [k]\}.$$

Here, we assume that $\infty + x = x + \infty = \infty + \infty = \infty$ for all $x \in \mathbb{N}$. This recurrence goes for each label $a \in [k]$ over all ways how the color set $f(a)$ can be split among the vertices with label a in the first graph G_{t_1} and in the second graph G_{t_2} .

This recurrence can be computed for all f simultaneously in time $\mathcal{O}^*((2^r)^k)$ by turning it into an appropriate cover product in the min-sum semiring as follows. We interpret the functions of the form $f: [k] \rightarrow \mathcal{P}([r])$ as subsets of $[k] \times [r]$ in the following way: $S(f) = \{(i, c) : i \in [k], c \in f(i)\}$. Observe that $f_1(a) \cup f_2(a) = f(a)$ for all $a \in [k]$ is equivalent to $S(f_1) \cup S(f_2) = S(f)$. Now, A_t can be considered as a function $\mathcal{P}([k] \times [r]) \rightarrow [n]$ and the recurrence of the union case is the $(\min, +)$ -cover product of A_{t_1} and A_{t_2} . By [10, Theorem 10.17] we can compute all values of A_t in time $2^{kr}(kr)^{\mathcal{O}(1)} \cdot \mathcal{O}(n \log n \log \log n) = \mathcal{O}^*((2^r)^k)$.

Correctness. We prove the correctness by bottom-up induction along the syntax tree \mathcal{T}_μ . In the base case G_t only consists of the single vertex v and we can either delete v or assign some color to v . Together with the edge case handling, this is implemented by the formula for the base case.

For the relabel case, notice that $G_t = G_{t'}$, $V(G_t^i) = \emptyset$, $V(G_t^j) = V(G_{t'}^i) \cup V(G_{t'}^j)$, and $V(G_t^a) = V(G_{t'}^a)$ for all $a \in [k] \setminus \{i, j\}$. Let f' be a candidate in the recurrence of $A_{t'}[f']$ and $\varphi' \in \mathcal{Q}_{t'}[f']$ be a minimizer in the definition of $A_{t'}[f']$, then we also have that $\varphi' \in \mathcal{Q}_t[f]$ since $\varphi'(V(G_t^j)) \setminus \{\perp\} = (\varphi'(V(G_{t'}^i)) \setminus \{\perp\}) \cup (\varphi'(V(G_{t'}^j)) \setminus \{\perp\}) = f'(i) \cup f'(j) = f(j)$. Hence, the recurrence is an upper bound on $A_t[f]$.

In the other direction, let φ be a minimizer in the definition of $A_t[f]$ and consider f' with $f'(a) = \varphi(V(G_t^a)) \setminus \{\perp\}$ for all $a \in [k]$. Then f' satisfies $f'(i) \cup f'(j) = f(j)$ and $\varphi \in \mathcal{Q}_{t'}[f']$, so f' is also considered in the recurrence and the recurrence is a lower bound on $A_t[f]$.

For the join case, notice that for $\varphi' \in \mathcal{Q}_{t'}[f] \supseteq \mathcal{Q}_t[f]$ it holds that $\varphi' \in \mathcal{Q}_t[f]$ if and only if $\varphi'(V(G_{t'}^i)) \cap \varphi'(V(G_{t'}^j)) \subseteq \{\perp\}$.

For the union case, a feasible solution φ of G_t induces feasible solutions φ_1 of G_{t_1} and φ_2 of G_{t_2} such that $\varphi_1(V(G_{t_1}^a)) \cup \varphi_2(V(G_{t_2}^a)) = \varphi(V(G_t^a))$ for all $a \in [k]$ and vice versa. \blacktriangleleft

This algorithm has a straightforward extension that can also handle polynomially large vertex costs in running time $\mathcal{O}^*((2^r)^k)$. For even larger costs it is not clear how to compute the table entries for the union nodes quickly enough.

6 Conclusion

Our main results are the two lower bounds for DELETION TO r -COLORABLE, which apply also to parameterization by treewidth resp. cliquewidth but use much more restrictive structure; this greatly refines what was known for ODD CYCLE TRANSVERSAL, i.e. $r = 2$, and gives new tight bounds for $r \geq 3$. In particular, beyond the above-mentioned examples, these are further natural problems where a small modulator to a simple graph class (of constant treewidth) is as hard as small treewidth. Surprisingly perhaps, something even stronger holds for clique-width: To get the tight lower bound, a modulator with few (true) twinclasses suffices, i.e., we need neither a sequence of disjoint separators nor complex dense structure. For DOMINATING SET, only the latter was established: twinclass-cutwidth rather than cliquewidth suffices to take us from base 3 in the running time to base 4.

Such results bring several benefits: (1) Rather than e.g. getting only the isolated result of (conditional) complexity of a problem relative to treewidth, we get a much larger range of input structure that exhibits the same tight complexity. (2) At the same time, by aiming for maximally restricted lower bound structure, we get a much better understanding of what structure makes a given problem hard. This in turn helps to focus efforts at faster algorithms through (even) stronger structural restrictions on the input.

An immediate follow-up question is whether there are improved algorithms for DELETION TO r -COLORABLE when $G - X$ has treewidth less than r ; so far, this is known only for ODD CYCLE TRANSVERSAL, but we think such algorithms exist in general. We observe that any construction relying on $(r + 1)$ -critical graphs must have treewidth at least r , hence improving upon the treewidth of our construction requires a fundamentally different idea.

Similarly, is there a meaningful restriction of (linear) clique-width, for which Lampis' [42] lower bound for r -COLORING already holds? Much more broadly, what other classes of problems exhibit the same lower bound as for treewidth already relative to deletion distance to a sparse graph class? Are there problems where this jump in complexity happens later, say, for treedepth, for some elimination distance, or only for treewidth/pathwidth? E.g., what is the complexity of DOMINATING SET relative to deletion distances, and the complexity relative to treedepth may be an interesting stepping stone? Similarly, to what generality do we get the same lower bound as for clique-width already relative to, e.g., twinclass-pathwidth?

References

- 1 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.
- 2 Glencora Borradaile and Hung Le. Optimal dynamic program for r -domination problems over tree decompositions. 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 8:1–8:23. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICs.IPEC.2016.8.
- 3 Jannis Bulian and Anuj Dawar. Graph isomorphism parameterized by elimination distance to bounded degree. *Algorithmica*, 75(2):363–382, 2016. doi:10.1007/s00453-015-0045-3.
- 4 Jannis Bulian and Anuj Dawar. Fixed-parameter tractable distances to sparse graph classes. *Algorithmica*, 79(1):139–158, 2017. doi:10.1007/s00453-016-0235-7.
- 5 Yijia Chen and Jörg Flum. Fo-definability of shrub-depth. In Maribel Fernández and Anca Muscholl, editors, *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*, volume 152 of *LIPICs*, pages 15:1–15:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CSL.2020.15.
- 6 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discret. Appl. Math.*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.
- 7 Radu Curticapean, Nathan Lindzey, and Jesper Nederlof. A tight lower bound for counting hamiltonian cycles via matrix rank. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1080–1099. SIAM, 2018. doi:10.1137/1.9781611975031.70.
- 8 Radu Curticapean and Dániel Marx. Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1650–1669. SIAM, 2016. doi:10.1137/1.9781611974331.ch113.
- 9 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016. doi:10.1145/2925416.

- 10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 11 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.
- 12 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.23.
- 13 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.
- 14 Matt DeVos, O-joung Kwon, and Sang-il Oum. Branch-depth: Generalizing tree-depth of graphs. *Eur. J. Comb.*, 90:103186, 2020. doi:10.1016/j.ejc.2020.103186.
- 15 Louis Dublois, Michael Lampis, and Vangelis Th. Paschos. New algorithms for mixed dominating set. *Discret. Math. Theor. Comput. Sci.*, 23(1), 2021. URL: <http://dmtcs.episciences.org/7407>.
- 16 Louis Dublois, Michael Lampis, and Vangelis Th. Paschos. Upper dominating set: Tight algorithms for pathwidth and sub-exponential approximation. *Theor. Comput. Sci.*, 923:271–291, 2022. doi:10.1016/j.tcs.2022.05.013.
- 17 László Egri, Dániel Marx, and Pawel Rzazewski. Finding list homomorphisms from bounded-treewidth graphs to reflexive graphs: a complete complexity characterization. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPICs*, pages 27:1–27:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.STACS.2018.27.
- 18 Eduard Eiben, Robert Ganian, Thekla Hamm, and O-joung Kwon. Measuring what matters: A hybrid approach to dynamic programming with treewidth. *J. Comput. Syst. Sci.*, 121:57–75, 2021. doi:10.1016/j.jcss.2021.04.005.
- 19 Eduard Eiben, Robert Ganian, and Stefan Szeider. Meta-kernelization using well-structured modulators. *Discret. Appl. Math.*, 248:153–167, 2018. doi:10.1016/j.dam.2017.09.018.
- 20 Eduard Eiben, Robert Ganian, and Stefan Szeider. Solving problems on graphs of high rank-width. *Algorithmica*, 80(2):742–771, 2018. doi:10.1007/s00453-017-0290-8.
- 21 Jacob Focke, Dániel Marx, and Pawel Rzazewski. Counting list homomorphisms from graphs of bounded treewidth: tight complexity bounds. 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 431–458. SIAM, 2022. doi:10.1137/1.9781611977073.22.
- 22 Jakub Gajarský and Stephan Kreutzer. Computing shrub-depth decompositions. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPICs*, pages 56:1–56:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.STACS.2020.56.
- 23 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In Gregory Z. Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation – 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, 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, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. The fine-grained complexity of graph homomorphism parameterized by clique-width. 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 66:1–66:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICALP.2022.66.
- 25 Robert Ganian, Petr Hlinený, Jaroslav Nesetril, Jan Obdržálek, and Patrice Ossona de Mendez. Shrub-depth: Capturing height of dense graphs. *Log. Methods Comput. Sci.*, 15(1), 2019. doi:10.23638/LMCS-15(1:7)2019.
- 26 Carla Groenland, Isja Mannens, Jesper Nederlof, and Krisztina Szilágyi. Tight bounds for counting colorings and connected edge sets parameterized by cutwidth. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPICs*, pages 36:1–36:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.STACS.2022.36.
- 27 Falko Hegerfeld and Stefan Kratsch. Towards exact structural thresholds for parameterized complexity. *CoRR*, abs/2107.06111, 2021. arXiv:2107.06111.
- 28 Petr Hlinený, O-joung Kwon, Jan Obdržálek, and Sebastian Ordyniak. Tree-depth and vertex-minors. *Eur. J. Comb.*, 56:46–56, 2016. doi:10.1016/j.ejc.2016.03.001.
- 29 Eva-Maria C. Hols, Stefan Kratsch, and Astrid Pieterse. Elimination distances, blocking sets, and kernels for vertex cover. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPICs*, pages 36:1–36:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.STACS.2020.36.
- 30 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.
- 31 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.
- 32 Yoichi Iwata and Yuichi Yoshida. On the equivalence among problems of bounded width. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms – ESA 2015 – 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 754–765. Springer, 2015. doi:10.1007/978-3-662-48350-3_63.
- 33 Ashwin Jacob, Fahad Panolan, Venkatesh Raman, and Vibha Sahlot. Structural parameterizations with modulator oblivion. *Algorithmica*, 84(8):2335–2357, 2022. doi:10.1007/s00453-022-00971-7.
- 34 Hugo Jacob, Thomas Bellitto, Oscar Defrain, and Marcin Pilipczuk. Close relatives (of feedback vertex set), revisited. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPICs*, pages 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.IPEC.2021.21.
- 35 Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. In Dimitris Fotakis, Aris Pagourtzis, and Vangelis Th. Paschos, editors, *Algorithms and Complexity – 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, volume 10236 of *Lecture Notes in Computer Science*, pages 345–356, 2017. doi:10.1007/978-3-319-57586-5_29.
- 36 Bart M. P. Jansen, Jari J. H. de Kroon, and Michal Włodarczyk. Vertex deletion parameterized by elimination distance and even less. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1757–1769. ACM, 2021. doi:10.1145/3406325.3451068.

- 37 Bart M. P. Jansen and Jesper Nederlof. Computing the chromatic number using graph decompositions via matrix rank. *Theor. Comput. Sci.*, 795:520–539, 2019. doi:10.1016/j.tcs.2019.08.006.
- 38 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r) -center. *Discrete Applied Mathematics*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.
- 39 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structurally parameterized d -scattered set. *Discret. Appl. Math.*, 308:168–186, 2022. doi:10.1016/j.dam.2020.03.052.
- 40 O-joung Kwon, Rose McCarty, Sang-il Oum, and Paul Wollan. Obstructions for bounded shrub-depth and rank-depth. *J. Comb. Theory, Ser. B*, 149:76–91, 2021. doi:10.1016/j.jctb.2021.01.005.
- 41 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/s00453-011-9554-x.
- 42 Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.
- 43 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.
- 44 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Trans. Algorithms*, 11(2):15:1–15:31, 2014. doi:10.1145/2566616.
- 45 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Degrees and gaps: Tight complexity results of general factor problems parameterized by treewidth and cutwidth. 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 95:1–95:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.95.
- 46 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Anti-factor is FPT parameterized by treewidth and list size (but counting is hard). *CoRR*, abs/2110.09369, 2022. To appear at IPEC 2022. arXiv:2110.09369.
- 47 Stefan Mengel. Parameterized compilation lower bounds for restricted CNF-formulas. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016 – 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2016. doi:10.1007/978-3-319-40970-2_1.
- 48 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.
- 49 Karolina Okrasa, Marta Piecyk, and Pawel Rzazewski. Full complexity classification of the list homomorphism problem for bounded-treewidth graphs. 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 74:1–74:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ESA.2020.74.
- 50 Karolina Okrasa and Pawel Rzazewski. Fine-grained complexity of the graph homomorphism problem for bounded-treewidth graphs. *SIAM J. Comput.*, 50(2):487–508, 2021. doi:10.1137/20M1320146.
- 51 Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for CNF formulas of bounded modular treewidth. *Algorithmica*, 76(1):168–194, 2016. doi:10.1007/s00453-015-0030-x.

- 52 Marta Piecyk and Pawel Rzazewski. Fine-grained complexity of the list homomorphism problem: Feedback vertex set and cutwidth. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 56:1–56:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.56.
- 53 Bas A. M. van Geffen, Bart M. P. Jansen, Arnoud A. W. M. de Kroon, and Rolf Morel. Lower bounds for dynamic programming on planar graphs of bounded cutwidth. *J. Graph Algorithms Appl.*, 24(3):461–482, 2020. doi:10.7155/jgaa.00542.

A Problem Definitions

VERTEX COVER

Input: An undirected graph $G = (V, E)$ and an integer b .

Question: Is there a set $Y \subseteq V$, $|Y| \leq b$, such that $G - Y$ contains no edges, i.e., $\chi(G - Y) \leq 1$?

ODD CYCLE TRANSVERSAL

Input: An undirected graph $G = (V, E)$ and an integer b .

Question: Is there a set $Y \subseteq V$, $|Y| \leq b$, such that $G - Y$ is bipartite, i.e., $\chi(G - Y) \leq 2$?

DELETION TO r -COLORABLE

Input: An undirected graph $G = (V, E)$ and an integer b .

Question: Is there a set $Y \subseteq V$, $|Y| \leq b$, such that $\chi(G - Y) \leq r$?

SATISFIABILITY

Input: A boolean formula σ in conjunctive normal form.

Question: Is there a satisfying assignment τ for σ ?

q -SATISFIABILITY

Input: A boolean formula σ in conjunctive normal form with clauses of size at most q .

Question: Is there a satisfying assignment τ for σ ?

q -HITTING SET

Input: An universe U and a set family \mathcal{F} over U of sets of size at most q and an integer t .

Question: Is there a set $H \subseteq U$, $|H| \leq t$, such that $H \cap S \neq \emptyset$ for all $S \in \mathcal{F}$?

 r -COLORING

Input: An undirected graph $G = (V, E)$.

Question: Is $\chi(G) \leq r$?

LIST r -COLORING

Input: An undirected graph $G = (V, E)$, lists $\Lambda(v) \subseteq [r]$ for all $v \in V$.

Question: Is there an r -Coloring $\varphi: V \rightarrow [r]$ of G such that $\varphi(v) \in \Lambda(v)$ for all $v \in V$?

MAXIMUM CUT

Input: An undirected graph $G = (V, E)$ and an integer b .

Question: Is there a set $Y \subseteq V$, such that $|\delta(Y)| \geq b$?

 H -FREE DELETION

Input: An undirected graph $G = (V, E)$ and an integer b .

Question: Is there a set $Y \subseteq V$, $|Y| \leq b$, such that $G - Y$ is H -free?

DOMINATING SET

Input: An undirected graph $G = (V, E)$ and an integer b .

Question: Is there a set $X \subseteq V$, $|X| \leq b$, such that $N[X] = V$?

TOTAL DOMINATING SET

Input: An undirected graph $G = (V, E)$ and an integer b .

Question: Is there a set $X \subseteq V$, $|X| \leq b$, such that $\bigcup_{v \in X} N(v) = V$?

 (b, r) -CENTER

Input: An undirected graph $G = (V, E)$ and an integers b and r .

Question: Is there a set $X \subseteq V$, $|X| \leq b$, such that every vertex $v \in V$ is at most at distance r to X ?

Hardness of Interval Scheduling on Unrelated Machines

Danny Hermelin ✉

Department of Industrial Engineering and Management,
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Yuval Itzhaki ✉

Department of Industrial Engineering and Management,
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Hendrik Molter ✉

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 provide new (parameterized) computational hardness results for INTERVAL SCHEDULING ON UNRELATED MACHINES. It is a classical scheduling problem motivated from *just-in-time* or *lean* manufacturing, where the goal is to complete jobs exactly at their deadline. We are given n jobs and m machines. Each job has a deadline, a weight, and a processing time that may be different on each machine. The goal is find a schedule that maximizes the total weight of jobs completed exactly at their deadline. Note that this uniquely defines a processing time interval for each job on each machine.

INTERVAL SCHEDULING ON UNRELATED MACHINES is closely related to coloring interval graphs and has been thoroughly studied for several decades. However, as pointed out by Mnich and van Bevern [Computers & Operations Research, 2018], the parameterized complexity for the number m of machines as a parameter remained open. We resolve this by showing that INTERVAL SCHEDULING ON UNRELATED MACHINES is W[1]-hard when parameterized by the number m of machines. To this end, we prove W[1]-hardness with respect to m of the special case where we have parallel machines with eligible machine sets for jobs. This answers Open Problem 8 of Mnich and van Bevern's list of 15 open problems in the parameterized complexity of scheduling [Computers & Operations Research, 2018].

Furthermore, we resolve the computational complexity status of the unweighted version of INTERVAL SCHEDULING ON UNRELATED MACHINES by proving that it is NP-complete. This answers an open question by Sung and Vlach [Journal of Scheduling, 2005].

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Theory of computation → W hierarchy; Theory of computation → Scheduling algorithms

Keywords and phrases Just-in-time scheduling, Parallel machines, Eligible machine sets, W[1]-hardness, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.18

Funding Supported by the ISF, grant No. 1070/20.

1 Introduction

In scheduling problems, we wish to assign jobs to machines in order to maximize a certain optimization objective while respecting certain constraints. In many traditional scheduling settings, jobs can be scheduled to start at any point in time and then need a given *processing*



© Danny Hermelin, Yuval Itzhaki, Hendrik Molter, and Dvir Shabtay;
licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 18; pp. 18:1–18:16

Leibniz International Proceedings in Informatics



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

time to be completed. However, in a typical *interval scheduling problem*, each job can be processed only in a fixed time interval, or sometimes in a set of time intervals, that may vary from machine to machine [16, 17]. Many different variations of interval scheduling have been considered and investigated [1, 2, 4, 5, 6, 7, 27].

In interval scheduling in its most basic form, we are given a set of n jobs and a set of m identical parallel machines that each can process one job at a time. Each job has a *processing time*, a *deadline*, and a *weight*, and shall be processed such that it finishes exactly at its deadline. This uniquely defines an interval for each job in which it can be processed. A *schedule* assigns a subset of the jobs to machines. Unassigned jobs are rejected. We call a schedule *feasible* if no two jobs with overlapping processing time intervals are assigned to the same machine. The goal is to find a feasible schedule that maximizes the weighted number of scheduled jobs.¹

This setting corresponds to the concept of *just-in-time (JIT)* or *lean manufacturing* that revolutionized industrial production processes in the 1980s and 1990s [18, 23, 26, 30, 31]. Herein, the main goal is to provide and receive goods precisely when they are needed in order to reduce storage costs and wastage. The first implementation of this manufacturing paradigm is attributed to the Japanese automobile company Toyota and is sometimes also called Toyota Production System (TPS) [23, 26]. Naturally, just-in-time and interval scheduling in many different variants has received much attention from the research community since the late 1980s until today [1, 3, 4, 5, 6, 7, 9, 13, 19, 20, 25, 27, 28].

The basic form of interval scheduling as described above is known to be solvable in polynomial time [2, 7, 8, 9, 13]. It is closely related to the classical problems of finding maximum independent sets in interval graphs and coloring interval graphs [8, 11, 24, 32]. The jobs of an interval scheduling instance naturally define an interval graph with vertex weights. For example, if there is only one machine, then interval scheduling is equivalent to finding a maximum weight independent set in an interval graph. Coloring an interval graph or, more specifically, computing its chromatic number is equivalent to determining the minimum number of machines necessary to schedule all jobs.

In our work, we investigate several natural generalizations and variants of interval scheduling and answer some longstanding open questions about their (parameterized) computational complexity.

The first problem we consider in this paper is INTERVAL SCHEDULING ON ELIGIBLE MACHINES, a natural generalization of the basic interval scheduling problem we introduced earlier. Here, each job additionally has a set of eligible machines and each job can only be assigned to a machine in this set in a feasible schedule. Arkin and Silverberg [2] proved in 1987 that INTERVAL SCHEDULING ON ELIGIBLE MACHINES is strongly NP-hard and can be solved in $O(mn^{m+1})$ time. In terms of parameterized complexity, Arkin and Silverberg [2] showed that INTERVAL SCHEDULING ON ELIGIBLE MACHINES is in XP when parameterized by the number m of machines. However, they left open whether INTERVAL SCHEDULING ON ELIGIBLE MACHINES also admits an FPT-algorithm for parameter m . Mnich and van Bevern [22] included this question as Open Problem 8 in their 2018 list of 15 open problems in the parameterized complexity of scheduling. We answer this question negatively in our first main contribution of this paper.

¹ In the standard three field notation for scheduling problems of Graham [12] this problem is sometimes denoted by $P \mid p_j = d_j - r_j \mid \sum_j w_j U_j$, or $P \parallel \sum_j w_j E_j$, or $P \parallel \text{JIT}$. We give a more formal definition in Section 2.

► **Theorem 1.** *INTERVAL SCHEDULING ON ELIGIBLE MACHINES is strongly² $W[1]$ -hard when parameterized by the number m of machines.*

A natural and well-studied generalization of INTERVAL SCHEDULING ON ELIGIBLE MACHINES is INTERVAL SCHEDULING ON UNRELATED MACHINES. In the latter, the processing time of each job can be machine-dependent whereas the deadline stays the same on all machines. Furthermore, each job is eligible on all machines. This definition stems from the just-in-time motivation, where each job should be finished exactly at its deadline but on different machines it may take different times to complete the job. We mention in passing that if both processing times and deadlines can be machine-dependent, the problem becomes NP-hard on two machines [17, 27]. Sung and Vlach [28] showed that INTERVAL SCHEDULING ON UNRELATED MACHINES can also be solved in $O(mn^{m+1})$ time, generalizing the result of Arkin and Silverberg [2]. Mnich and van Bevern [22] asked in Open Problem 8 for an FPT-algorithm for INTERVAL SCHEDULING ON ELIGIBLE MACHINES parameterized by the number m of machines as a first step towards finding an FPT-algorithm for INTERVAL SCHEDULING ON UNRELATED MACHINES parameterized by m . However, Theorem 1 naturally implies that INTERVAL SCHEDULING ON UNRELATED MACHINES presumably also does not admit an FPT-algorithm for the number m of machines as a parameter.

► **Corollary 2.** *INTERVAL SCHEDULING ON UNRELATED MACHINES is strongly $W[1]$ -hard when parameterized by the number m of machines.*

We point out that all known hardness reductions for INTERVAL SCHEDULING ON UNRELATED MACHINES require job weights, raising the question whether the weights play an integral role in the computational complexity of the problem. UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES is the natural special case of INTERVAL SCHEDULING ON UNRELATED MACHINES where all jobs have weight one. Sung and Vlach [28] asked in 2005 to resolve the computational complexity status of UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES. We give an answer to this in our second main contribution.

► **Theorem 3.** *UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES is NP-complete.*

We remark that our reduction for Theorem 3 does not imply hardness for the unweighted version of INTERVAL SCHEDULING ON ELIGIBLE MACHINES. We leave this open for future research. An additional immediate question that we leave open for future research is whether UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES admits an FPT-algorithm for the number m of machines as a parameter.

With Theorem 1, Corollary 2, and Theorem 3 we answer fundamental longstanding open questions concerning the (parameterized) computational complexity of natural interval scheduling problems. For INTERVAL SCHEDULING ON ELIGIBLE MACHINES and INTERVAL SCHEDULING ON UNRELATED MACHINES, our results together with the XP-containment results from Arkin and Silverberg [2] and Sung and Vlach [28], respectively, essentially resolve their parameterized complexity classification for the number m of machines as a parameter. We point out that all considered problem variants are known to be fixed-parameter tractable when parameterized by the number n of jobs. This can be shown with a simple reduction to MULTICOLORED INDEPENDENT SET ON INTERVAL GRAPHS parameterized

² A parameterized problem is *strongly* $W[1]$ -hard if it remains $W[1]$ -hard when all numbers are encoded unarily.

by the number of colors, which is known to be fixed-parameter tractable [4, 5]. Hence, we make an important further step towards fully understanding the parameterized complexity of several basic and natural interval scheduling problems. We remark that our results also imply that MULTICOLORED INDEPENDENT SET ON INTERVAL GRAPHS is W[1]-hard when parameterized by the maximum number of vertices of any color.

The rest of the paper is organized as follows: we give formal definitions of all problems in Section 2. We prove Theorem 1 and Corollary 2 in Section 3 and we prove Theorem 3 in Section 4. We conclude with future research directions in Section 5.

2 Problem Setting

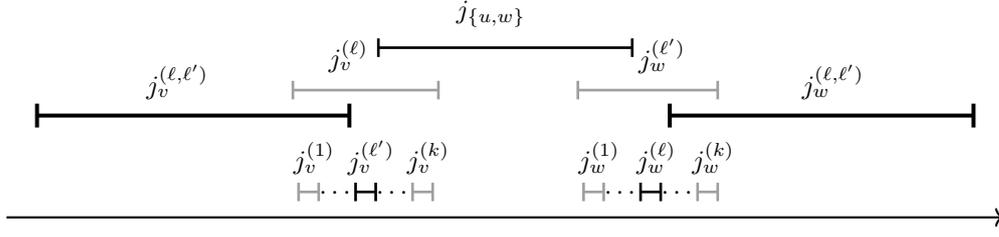
The first problem we consider is INTERVAL SCHEDULING ON ELIGIBLE MACHINES. Here, we have a set of n jobs $\{j_1, j_2, \dots, j_n\}$ and a set of m machines $\{i_1, i_2, \dots, i_m\}$ that each can process one job at a time. Each job j has a *processing time* p_j , a *deadline* d_j , a *weight* w_j , and a set of *eligible machines* $M_j \subseteq \{i_1, i_2, \dots, i_m\}$. Job j can be processed in exactly *one* fixed time interval $(d_j - p_j, d_j]$, specified by its processing time and deadline, that is the same on each of its eligible machines. A schedule is a mapping from jobs to machines. More formally, a *schedule* is a function $\sigma : \{j_1, j_2, \dots, j_n\} \rightarrow \{i_1, i_2, \dots, i_m, \perp\}$. If for job j we have $\sigma(j) = i$ (with $i \neq \perp$), then job j is scheduled to be processed on machine i . If for job j we have $\sigma(j) = \perp$, then job j is not scheduled, that is, it is not assigned to any machine. We say that two jobs j, j' are *in conflict* on a machine i if $(d_j - p_j, d_j] \cap (d_{j'} - p_{j'}, d_{j'}] \neq \emptyset$, that is, the processing time intervals corresponding to jobs j and j' on machine i overlap. A schedule σ is *feasible* if there is no pair of jobs j, j' with $\sigma(j) = \sigma(j') = i \neq \perp$ that is in conflict on machine i and each job is mapped to one of its eligible machines. The goal is to find a feasible schedule that maximizes the weighted number of scheduled jobs $W = \sum_{j|\sigma(j) \neq \perp} w_j$. In the standard three field notation for scheduling problems of Graham [12] INTERVAL SCHEDULING ON ELIGIBLE MACHINES is sometimes denoted by $P \mid M_j, p_j = d_j - r_j \mid \sum_j w_j U_j$, or $P \mid M_j \mid \sum_j w_j E_j$, or $P \mid M_j \mid \text{JIT}$.

The second problem we consider is INTERVAL SCHEDULING ON UNRELATED MACHINES. Here, for each job j the processing time $p_{i,j}$ can depend on machine i whereas the deadline d_j is the same on all machines. Hence, the processing time interval of job j on machine i is $(d_j - p_{i,j}, d_j]$. Moreover, the all jobs are eligible on all machines, that is, $M_j = \{i_1, i_2, \dots, i_m\}$ for all jobs j . In the standard three field notation for scheduling problems of Graham [12] INTERVAL SCHEDULING ON UNRELATED MACHINES is sometimes denoted by $R \mid p_j = d_j - r_j \mid \sum_j w_j U_j$, or $R \parallel \sum_j w_j E_j$, or $R \parallel \text{JIT}$.

Finally, the third problem we consider is UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES, the unweighted version of INTERVAL SCHEDULING ON UNRELATED MACHINES. Here, we have that $w_j = 1$ for all jobs j . In the standard three field notation for scheduling problems of Graham [12] UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES is sometimes denoted by $R \mid p_j = d_j - r_j \mid \sum_j U_j$, or $R \parallel \sum_j E_j$, or $R \mid w_j = 1 \mid \text{JIT}$.

3 W[1]-Hardness of Interval Scheduling on Eligible Machines

In this section, we prove Theorem 1 from which Corollary 2 follows directly. To prove Theorem 1, we present a parameterized polynomial-time reduction from MULTICOLORED CLIQUE parameterized by the number of colors to INTERVAL SCHEDULING ON ELIGIBLE MACHINES parameterized by the number m of machines. In MULTICOLORED CLIQUE, we



■ **Figure 1** Illustration of the edge selection machine for color combination ℓ, ℓ' with $\ell < \ell'$. Depicted are intervals of jobs relating to $v \in V_\ell$, $w \in V_{\ell'}$, and $e = \{v, w\} \in E$. Gray intervals correspond to jobs that are not eligible on the machine.

are given a k -partite graph $G = (V_1 \uplus V_2 \uplus \dots \uplus V_k, E)$, we are asked whether G contains a clique of size k . The k vertex parts V_1, V_2, \dots, V_k are called *colors*. MULTICOLORED CLIQUE parameterized by k is known to be W[1]-hard [10].

Given an instance of MULTICOLORED CLIQUE, we construct an instance of INTERVAL SCHEDULING ON ELIGIBLE MACHINES as follows.

► **Construction 1.** Let $G = (V_1 \uplus V_2 \uplus \dots \uplus V_k, E)$ be a k -partite graph with n_G vertices. Assume we have some total ordering $<_\pi$ over $V := V_1 \uplus V_2 \uplus \dots \uplus V_k$ such that for all $v \in V_\ell$ and $w \in V_{\ell'}$ we have that if $\ell < \ell'$ then $v <_\pi w$. Let $\pi(v)$ denote the ordinal position of $v \in V$ in the ordering $<_\pi$.

In the following, we describe the jobs and specify their processing times, deadlines and weights. Then we describe the machines and the eligible machine sets for the jobs. In order to describe the weights more easily, we introduce the following three values: $c_1 = n_G + 1$, $c_2 = (k - 1)n_G c_1 + n_G + 1$, and $c_3 = (kn_G + k^2 n_G)n_G c_2 + 1$. We create the following jobs:

- For each vertex $v \in V$, we create k vertex jobs $j_v^{(1)}, j_v^{(2)}, \dots, j_v^{(k)}$, where one of the vertex jobs corresponds to the color of v and the $k - 1$ other vertex jobs correspond to the other $k - 1$ colors.

Let $v \in V_\ell$. The processing time of $j_v^{(\ell)}$ (the vertex job corresponding to the same color as v) is $k + 2$, the deadline of $j_v^{(\ell)}$ is $(k + 2)\pi(v) + 1$, and the weight of $j_v^{(\ell)}$ is one.

The processing time of $j_v^{(\ell')}$ with $\ell' \neq \ell$ (vertex jobs corresponding to a different color than v) is one, the deadline of $j_v^{(\ell')}$ with $\ell' \neq \ell$ is $(k + 2)\pi(v) - \ell'$, and the weight of $j_v^{(\ell')}$ with $\ell' \neq \ell$ is c_1 .

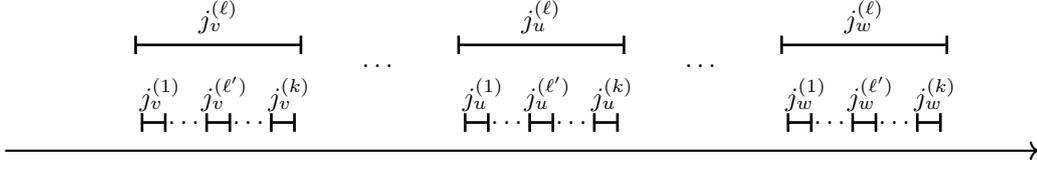
- For each edge $e = \{v, w\} \in E$ with $v \in V_\ell$, $w \in V_{\ell'}$, and $\ell < \ell'$, we create one edge job j_e with processing time $(k + 2)(\pi(w) - \pi(v)) - \ell + \ell'$, deadline $(k + 2)\pi(w) - \ell - 1$, and weight $c_2(\pi(w) - \pi(v)) + c_3$.
- For each color combination ℓ, ℓ' with $\ell < \ell'$ we create $|V_\ell| + |V_{\ell'}|$ color combination jobs, one for each $v \in V_\ell$ and one for each $w \in V_{\ell'}$.

Let $v \in V_\ell$, we create a job $j_v^{(\ell, \ell')}$ with processing time $(k + 2)\pi(v) - \ell' - 2$, deadline $(k + 2)\pi(v) - \ell' - 1$, and weight $c_2\pi(v)$.

Let $w \in V_{\ell'}$, we create a job $j_w^{(\ell, \ell')}$ with processing time $(k + 2)(n_G - \pi(w)) + \ell + 2$, deadline $(k + 2)n_G + 2$, and weight $c_2(n_G - \pi(w))$.

We create $m = \binom{k}{2} + 1$ machines $i_1, i_2, \dots, i_{\binom{k}{2}+1}$. We call the first $\binom{k}{2}$ machines *edge selection machines* (one machine for each color combination) and we call the remaining machine *validation machine*.

Consider color combination ℓ, ℓ' with $\ell < \ell'$ and let i be the corresponding edge selection machine.



■ **Figure 2** Illustration of the validation machine. Depicted are intervals of jobs corresponding to vertices $v, u, w \in V_\ell$ with $v <_\pi u <_\pi w$.

- For each vertex $v \in V_\ell$ we add machine i to the set of eligible machines of job $j_v^{(\ell)}$ and of job $j_v^{(\ell, \ell')}$.
- For each vertex $w \in V_{\ell'}$ we add machine i to the set of eligible machines of job $j_w^{(\ell)}$ and of job $j_w^{(\ell, \ell')}$.
- For each edge $e = \{v, w\} \in E$ with $v \in V_\ell$ and $w \in V_{\ell'}$ we add machine i to the set of eligible machines of job j_e .

We give an illustration of the edge selection machines in Figure 1. Finally, consider the validation machine $i_{\binom{k}{2}+1}$. We add the validation machine to the set of eligible machines of all vertex jobs. We give an illustration of the validation machine in Figure 2.

This finishes our construction of the INTERVAL SCHEDULING ON ELIGIBLE MACHINES instance. We first show that given a clique of size k in G , we can create a feasible schedule for the constructed instance such that the total weight of scheduled jobs attains at least a certain value.

► **Lemma 4.** *Let G be an instance of MULTICOLORED CLIQUE. Let I be the INTERVAL SCHEDULING ON ELIGIBLE MACHINES instance computed from G as specified by Construction 1. If G contains a clique of size k , then there is a feasible schedule σ for I such that for the total weight W of scheduled jobs we have*

$$W \geq \binom{k}{2}c_3 + \binom{k}{2}n_Gc_2 + (k-1)n_Gc_1 + k.$$

Proof. Let $G = (V_1 \uplus V_2 \uplus \dots \uplus V_k, E)$ be an instance of MULTICOLORED CLIQUE and consider the corresponding INTERVAL SCHEDULING ON ELIGIBLE MACHINES instance specified by Construction 1. Assume there is a clique X of size k in G . Then we schedule the following jobs.

- For color combination ℓ, ℓ' with $\ell < \ell'$ let $\{v\} = X \cap V_\ell$ and $\{w\} = X \cap V_{\ell'}$. Since X is a clique in G , we know that $e = \{v, w\} \in E$. On edge selection machine i corresponding to color combination ℓ, ℓ' we schedule the following jobs: $j_e, j_v^{(\ell, \ell')}, j_w^{(\ell, \ell')}, j_v^{(\ell')}$, and $j_w^{(\ell)}$. By construction of the instance, the intervals of the jobs are non-intersecting on the machine i , and machine i is in the eligible set of the four jobs. Hence, scheduling these jobs on machine i yields a feasible schedule. Furthermore, it accounts for weight $c_3 + n_Gc_2 + 2c_1$ of scheduled jobs per color combination. Summing over all color combinations, we obtain weight $\binom{k}{2}c_3 + \binom{k}{2}n_Gc_2 + k(k-1)c_1$. Note that for each $\{v\} = X \cap V_\ell$ we have scheduled all vertex jobs $j_v^{(\ell')}$ with $\ell \neq \ell'$.
- On the validation machine $i_{\binom{k}{2}+1}$ we schedule the following jobs. Let $\{v\} = X \cap V_\ell$, then we schedule vertex job $j_v^{(\ell)}$. Note that this job is only in conflict with vertex jobs $j_v^{(\ell')}$ with $\ell \neq \ell'$, which are scheduled on the edge selection machines. Furthermore, we

schedule all jobs $j_w^{(\ell')}$ with $v \neq w \in V_\ell$ and $\ell \neq \ell'$. By construction, all these jobs can be scheduled on machine $i_{\binom{k}{2}+1}$ without conflicts and all these jobs have machine $i_{\binom{k}{2}+1}$ in their set of eligible machines.

For all colors, this accounts for weight $(n_G - k)(k - 1)c_1 + k$ of scheduled jobs.

Clearly, we have that the constructed schedule is feasible. Furthermore, it is straightforward to check that the total weight of scheduled jobs in this constructed schedule is W . ◀

Before we show a similar statement for the opposite direction, we make an observation about feasible schedules in INTERVAL SCHEDULING ON ELIGIBLE MACHINES instances from Construction 1. We show that we can assume that any feasible schedule where the total weight of scheduled jobs is at least $\binom{k}{2}c_3 + \binom{k}{2}n_Gc_2 + (k - 1)n_Gc_1 + k$ schedules exactly one edge job on each edge selection machine.

► **Observation 5.** *Let I be an instance of INTERVAL SCHEDULING ON ELIGIBLE MACHINES resulting from applying Construction 1 to some k -partite graph G . Let σ be a feasible schedule such that for the total weight W of scheduled jobs we have*

$$W \geq \binom{k}{2}c_3 + \binom{k}{2}n_Gc_2 + (k - 1)n_Gc_1 + k.$$

Then exactly $\binom{k}{2}$ edge jobs are scheduled, one on each edge selection machine.

Proof. We first show that no feasible schedule with total weight $W \geq \binom{k}{2}c_3 + \binom{k}{2}n_Gc_2 + (k - 1)n_Gc_1 + k$ of scheduled jobs can schedule more than $\binom{k}{2}$ edge jobs. Let ℓ, ℓ' with $\ell < \ell'$ be a color combination. On the edge selection machine corresponding to color combination ℓ, ℓ' we have that all edge jobs corresponding to edges that do *not* connect vertices of colors ℓ and ℓ' are not eligible. Furthermore, all edge jobs corresponding to edges that connect vertices of colors ℓ and ℓ' are pairwise in conflict. It follows that on each edge selection machine, at most one edge job can be scheduled. Hence, any feasible schedule can schedule at most $\binom{k}{2}$ edge jobs, one on each edge selection machine.

We next show that any feasible schedule with $W \geq \binom{k}{2}c_3 + \binom{k}{2}n_Gc_2 + (k - 1)n_Gc_1 + k$ needs to schedule at least $\binom{k}{2}$ edge jobs. Assume we have a feasible schedule that schedules (strictly) less than $\binom{k}{2}$ edge jobs. Note that each edge job has weight at least c_3 . Furthermore, there are at most $kn_G + k^2n_G$ jobs that are not edge jobs and those jobs each have weight at most n_Gc_2 . Let σ be a feasible schedule that does not schedule all edge jobs and let W be the total weight of all jobs scheduled by σ . Let W^* denote the sum of weights of all jobs that are not edge jobs. Then we have

$$W < \left(\binom{k}{2} - 1\right)c_3 + W^* < \binom{k}{2}c_3.$$

Hence, the observation follows. ◀

Now we are ready to show how to construct a clique of size k from a feasible schedule where the total weight of scheduled jobs is at least $\binom{k}{2}c_3 + \binom{k}{2}n_Gc_2 + (k - 1)n_Gc_1 + k$.

► **Lemma 6.** *Let G be an instance of MULTICOLORED CLIQUE. Let I be the INTERVAL SCHEDULING ON ELIGIBLE MACHINES instance computed from G as specified by Construction 1. If there is a feasible schedule σ for I such that for the total weight W of scheduled jobs we have*

$$W \geq \binom{k}{2}c_3 + \binom{k}{2}n_Gc_2 + (k - 1)n_Gc_1 + k,$$

then G contains a clique of size k .

Proof. Let $G = (V_1 \uplus V_2 \uplus \dots \uplus V_k, E)$ be an instance of MULTICOLORED CLIQUE and consider the corresponding INTERVAL SCHEDULING ON ELIGIBLE MACHINES instance specified by Construction 1. Assume we have a feasible schedule σ such that the for the total weight W of scheduled jobs we have $W \geq \binom{k}{2}c_3 + \binom{k}{2}n_Gc_2 + (k-1)n_Gc_1 + k$. We construct a clique of size k in G as follows.

By Observation 5 we know that σ schedules one edge job on each edge selection machine. We can also observe that on the validation machine, only vertex jobs are eligible and can be scheduled. Note that the sum of weights of all vertex jobs is $(k-1)n_Gc_1 + n_G$, which is strictly smaller than c_2 .

Assume that edge job j_e is scheduled on the edge selection machine i corresponding to color combination ℓ, ℓ' with $\ell < \ell'$. Then by construction, $e = \{v, w\} \in E$ with $v \in V_\ell$ and $w \in V_{\ell'}$. Now by construction of the instance, two color combination jobs can be scheduled on machine i , one for a vertex of color ℓ and one for a vertex of color ℓ' . In order to obtain a weight of scheduled jobs of at least $c_3 + n_Gc_2$ it is necessary that jobs $j_v^{(\ell, \ell')}$ and $j_w^{(\ell, \ell')}$ are scheduled (note that the weights of j_e , $j_v^{(\ell, \ell')}$, and $j_w^{(\ell, \ell')}$ sum up to exactly $c_3 + n_Gc_2$). Any other selection of color combination jobs to schedule either results in a weight that is lower by at least c_2 or in an infeasible schedule. Now, by construction, the only further jobs that can be scheduled are $j_v^{(\ell')}$ and $j_w^{(\ell)}$. It follows that the maximum weight achievable on any edge selection machine is $c_3 + n_Gc_2 + 2c_1$. Since $\binom{k}{2}2c_1 < c_2$, it follows that for each edge selection machine corresponding to color combination ℓ, ℓ' with $\ell < \ell'$ we have the following: one edge job j_e for $e = \{v, w\}$ with $v \in V_\ell$ and $w \in V_{\ell'}$ is scheduled and the two color combination jobs $j_v^{(\ell, \ell')}$ and $j_w^{(\ell, \ell')}$ are scheduled.

We can conclude that the jobs scheduled on all edge selection machines have a total weight of at least $\binom{k}{2}c_3 + \binom{k}{2}n_Gc_2$. Hence, there are additional jobs scheduled that have a total weight of $(k-1)n_Gc_1 + k$ on the validation machine.

Since no additional edge jobs or color combination jobs can be scheduled, we have that all $(k-1)n_G$ vertex jobs $j_v^{(\ell')}$ with $v \in V_\ell$ and $\ell \neq \ell'$ (having weight $c_1 > n_G$) are scheduled. Furthermore, at least k vertex jobs $j_v^{(\ell)}$ with $v \in V_\ell$ (having weight one) are scheduled.

Let X be the set of vertices in G such that if $v \in X$ and $v \in V_\ell$, the job $j_v^{(\ell)}$ is scheduled. We claim that X is a clique of size at least k in G .

By construction we have that $|X| \geq k$, assume for contradiction that X is not a clique in G . Then there are two vertices $v, w \in X$ such that $e = \{v, w\} \notin E$. Let $v \in V_\ell$ and $w \in V_{\ell'}$. Then, in particular, vertex jobs $j_v^{(\ell')}$ and $j_w^{(\ell)}$ cannot be scheduled on the validation machine, since they are in conflict with vertex jobs $j_v^{(\ell)}$ and $j_w^{(\ell')}$, respectively. However, as observed above, vertex jobs $j_v^{(\ell')}$ and $j_w^{(\ell)}$ can only be scheduled on the edge selection machine corresponding to color combination ℓ, ℓ' if there is edge job j_e with $e = \{v, w\}$ scheduled on that machine, a contradiction to the assumption that $e = \{v, w\} \notin E$. ◀

Finally, we have all ingredients to prove Theorem 1.

Proof of Theorem 1. To prove Theorem 1, we show that Construction 1 is parameterized polynomial-time reduction from MULTICOLORED CLIQUE parameterized by the number of colors to INTERVAL SCHEDULING ON ELIGIBLE MACHINES parameterized by the number m of machines. First, it is easy to observe that given an instance of MULTICOLORED CLIQUE, the INTERVAL SCHEDULING ON ELIGIBLE MACHINES instance specified by Construction 1 can be computed in polynomial time. Furthermore, if k is the number of colors in the MULTICOLORED CLIQUE instance, then the number of machines in the constructed INTERVAL

SCHEDULING ON ELIGIBLE MACHINES instance is $m = \binom{k}{2} + 1$. Lastly, observe that all weights in the constructed INTERVAL SCHEDULING ON ELIGIBLE MACHINES instance are in $n_G^{O(1)}$ (where n_G is the number of vertices in the MULTICOLORED CLIQUE instance).

The correctness of the reduction follows from Lemmas 4 and 6. Since MULTICOLORED CLIQUE parameterized by the number of colors is W[1]-hard [10], we have that Theorem 1 follows. ◀

4 NP-Hardness of Unweighted Interval Scheduling on Unrelated Machines

In this section we prove Theorem 3. The containment of UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES in NP is easy to see, hence we focus on proving NP-hardness. To this end, we present a polynomial-time many-one reduction from EXACT (3,4)-SAT to UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES. In EXACT (3,4)-SAT we are given a Boolean formula ϕ in conjunctive normal form where every clause has exactly three literals and every variable appears in exactly four clauses, and are asked whether ϕ has a satisfying assignment. EXACT (3,4)-SAT is known to be NP-hard [29].

Given such a formula ϕ , we construct an instance I of UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES as follows.

► **Construction 2.** Let ϕ be a Boolean formula in conjunctive normal form where every clause has exactly three literals and every variable appears in exactly four clauses. Let α be the number of variables in ϕ and let β be the number of clauses in ϕ . We construct an instance I of UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES as follows.

We first describe the jobs, then we define an ordering of the jobs and use it to specify their deadlines. Lastly, we describe the processing times of the jobs on the different machines. We create the following jobs.

- For every variable x , we create two *variable jobs*: x^T and x^F .
- For every clause c , we create three *clause jobs*: c_1 , c_2 , and c_3 .
- We create $2\alpha + 2\beta$ *dummy jobs*.

We next define an ordering π of the jobs, which we will use to define the deadlines of the jobs. To this end, we partition the jobs into the following sets.

- Let $T = \{x^T \mid x \text{ is a variable in } \phi\}$.
- Let $F = \{x^F \mid x \text{ is a variable in } \phi\}$.
- Let $P = \{c_\ell \mid \text{the } \ell\text{th literal of clause } c \text{ of } \phi \text{ is non-negated}\}$.
- Let $N = \{c_\ell \mid \text{the } \ell\text{th literal of clause } c \text{ of } \phi \text{ is negated}\}$.
- Let D be the set of dummy jobs.

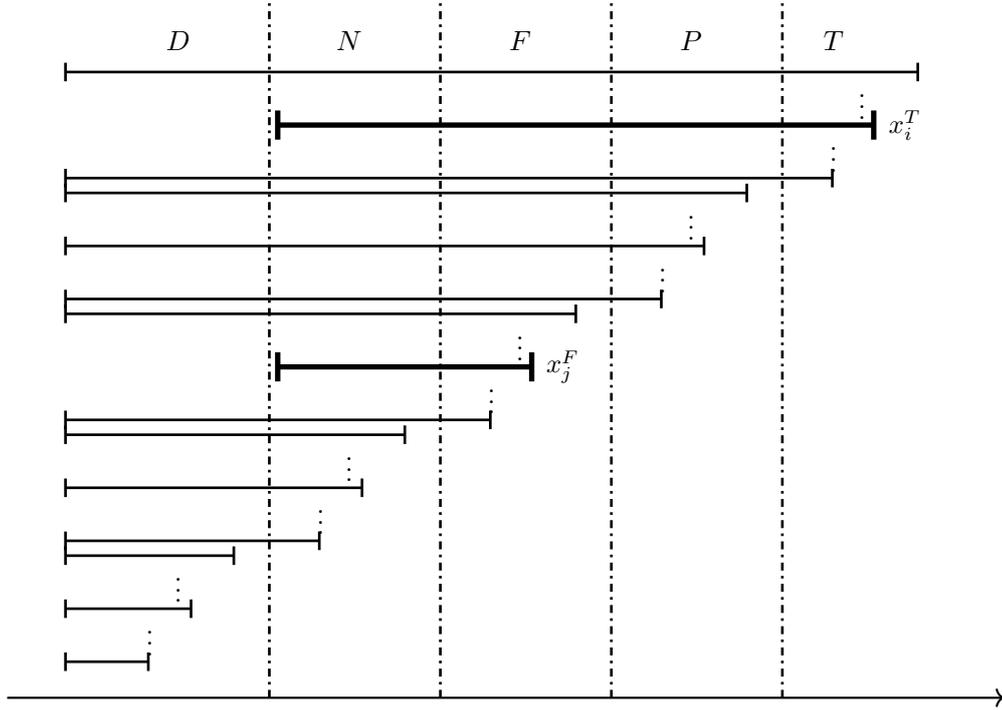
Now we define π as a total ordering of the jobs such that

$$D \prec N \prec F \prec P \prec T,$$

and the jobs within the sets are ordered in an arbitrary but fixed way. Let $\pi(j)$ denote the ordinal position of job j in π . For each job j , we set

$$d_j = \pi(j).$$

We next describe the machines, more specifically, the processing times of all jobs on each of the machines. We first introduce α *variable selection machines*, one for each variable in ϕ . Let x be a variable in ϕ , then we introduce a machine where the processing time of variable



■ **Figure 3** Illustration of the job intervals on the variable selection machine for variable x . On this machine only one of jobs x^T and x^F (bold) can be scheduled alongside with one dummy job.

job x^T is $\pi(x^T) - 2\alpha - 2\beta$ and the processing time of variable job x^F is $\pi(x^F) - 2\alpha - 2\beta$. We set the processing times of all other jobs to their respective deadlines. We give an illustration of the variable selection machines in Figure 3.

Next, we introduce 2β *clause selection machines*, two for each clause in ϕ . Let c be a clause in ϕ , then we introduce two machines where the processing times of c_1 , c_2 , and c_3 are $\pi(c_1) - 2\alpha - 2\beta$, $\pi(c_2) - 2\alpha - 2\beta$, and $\pi(c_3) - 2\alpha - 2\beta$, respectively. We set the processing times of all other jobs to their respective deadlines. We give an illustration of the clause selection machines in Figure 4.

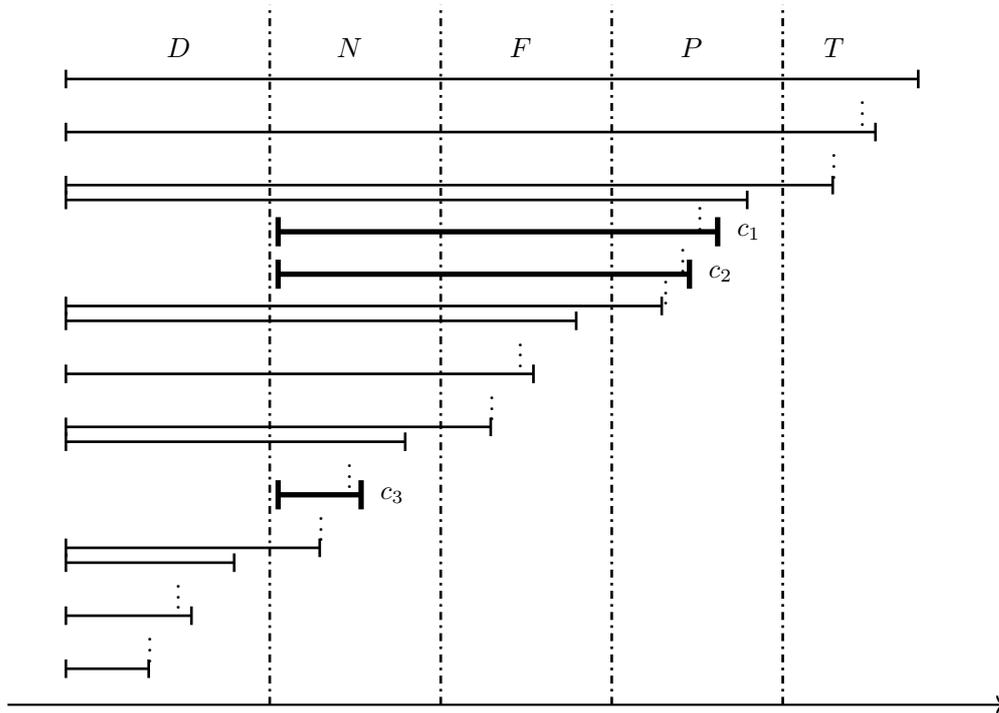
Furthermore, we have α *validation machines*, one for each variable in ϕ . Let x be a variable in ϕ , then we introduce a machine where

- the processing time of x^T is $\pi(x^T) - \pi(x^F) + 1$,
- the processing time of x^F is $\pi(x^F) - 2\alpha - 2\beta$,
- if x appears in the ℓ th literal of clause c , then the processing time of c_ℓ is one, and
- processing times of all other jobs are set to their respective deadlines.

We give an illustration of the validation machines in Figure 5.

This finishes our construction of the UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES instance. We first show that given a satisfying assignment for ϕ , we can create a feasible schedule for the constructed instance such all jobs are scheduled.

► **Lemma 7.** *Let ϕ be an instance of EXACT (3,4)-SAT. Let I be the UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES instance computed from ϕ as specified by Construction 2. If ϕ is satisfiable, then there is a feasible schedule σ for I such that all jobs are scheduled.*



■ **Figure 4** Illustration of the job intervals on the clause selection machine for clause c . On this machine only one of the jobs c_1 , c_2 and c_3 (bold) can be scheduled alongside with one dummy job.

Proof. Let ϕ be an instance of EXACT (3,4)-SAT and consider the corresponding UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES instance specified by Construction 2. Assume there is a satisfying assignment for ϕ . Then we schedule the jobs as follows.

We first describe on which machine we schedule each variable job. Let x be a variable in ϕ . If x is set to true in the satisfying assignment, we schedule variable job x^T on the variable selection machine corresponding to x and we schedule variable job x^F on the validation machine corresponding to x . Otherwise, we schedule variable job x^F on the variable selection machine corresponding to x and we schedule variable job x^T on the validation machine corresponding to x .

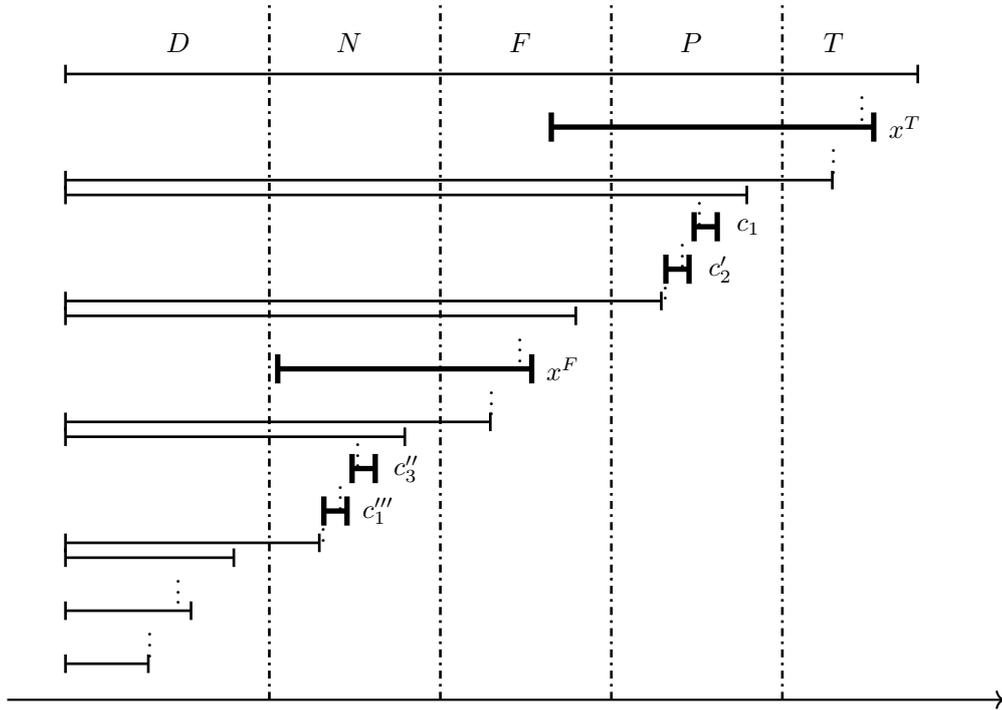
Next, we describe on which machine to schedule each clause job. Let c be a clause in ϕ . Let clause c be satisfied by its ℓ th literal (if multiple literals satisfy the clause, pick one of them arbitrarily). Let x be the variable appearing in the ℓ th literal of c . We schedule clause job c_ℓ on the validation machine corresponding to x . We schedule clause jobs $c_{\ell'}$ with $\ell' \in \{1, 2, 3\} \setminus \{\ell\}$ on the two clause selection machines corresponding to c , respectively.

Lastly, notice that the number of dummy jobs equals the number of machines. For each dummy job we arbitrarily choose a distinct machine and schedule it on this machine.

In the constructed schedule, we clearly schedule each job. We next show that the schedule is feasible.

Notice that on each variable selection machine and each clause selection machine we schedule exactly two jobs, one dummy job and one variable job of the variable corresponding to the variable selection machine or, respectively, one clause job of the clause corresponding to the clause selection machine. Since the processing time intervals of the variable jobs of

18:12 **Hardness of Interval Scheduling on Unrelated Machines**



■ **Figure 5** Illustration of the job intervals on the validation machine for variable x for the case that x appears in clauses c and c' non-negated in positions one and two, respectively, and that x appears in clauses c'' and c''' negated in positions three and one, respectively. Processing time intervals of jobs that do not conflict with dummy jobs on this machine are depicted in bold.

variables corresponding to the variable selection machines start at $2\alpha + 2\beta$ and the deadline of each dummy job is at most $2\alpha + 2\beta$, the schedules for the variable selection machines are feasible. Analogously, the schedules for the clause selection machines are feasible.

It remains to show that the schedules for the validation machines are feasible. Notice that by construction, the variable jobs and clause jobs that are potentially scheduled on a validation machine cannot conflict with any dummy job. Furthermore, the variable jobs that are potentially scheduled on a validation machine cannot conflict with each other. We have the same for the clause jobs.

Hence, the only way to obtain an infeasible schedule is if a variable job and a clause job are in conflict. Assume the variable x^T is scheduled (the case of variable job x^F is analogous) and clause job c_ℓ is scheduled and the two jobs are in conflict. Note that this implies that we are dealing with the validation machine for variable x . By construction of the schedule, this means that variable x is set to false in the satisfying assignment. However, the jobs c_ℓ and x^T are in conflict on the validation machine for x (and the job of c_ℓ is not in conflict with the dummy jobs) if x appears non-negated in the ℓ th literal of clause c . Furthermore, by construction of the schedule, we have that clause c is satisfied by its ℓ th literal. This is a contradiction to x being set to false in the satisfying assignment. ◀

Now we show how to construct a satisfying assignment from a feasible schedule where all jobs are scheduled.

► **Lemma 8.** *Let ϕ be an instance of EXACT (3,4)-SAT. Let I be the UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES instance computed from ϕ as specified by Construction 2. If there is a feasible schedule σ for I such that all jobs are scheduled, then ϕ is satisfiable.*

Proof. Let ϕ be an instance of EXACT (3,4)-SAT and consider the corresponding UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES instance specified by Construction 2. Assume we have a feasible schedule for the constructed instance such that all jobs are scheduled. We construct a satisfying assignment for ϕ as follows.

First, observe that by construction, the at most one dummy job can be scheduled on each machine. Since the number of dummy jobs equals the number of machines, we have that on each machine exactly one dummy job is scheduled. This means that on each machine no non-dummy jobs that conflict with a dummy job (that is, jobs with processing time equal to their deadline) can be scheduled.

We can further observe that on the variable selection machine of variable x , apart from a dummy job, only the variable job x^T or the variable job x^F can be scheduled. Since the two jobs conflict, they cannot both be scheduled. We assume w.l.o.g. that exactly one of the two jobs is scheduled. If the variable job x^T is scheduled, we set variable x to true, otherwise we set variable x to false. In the remainder, we show that this yields a satisfying assignment for ϕ .

Assume for contradiction that ϕ is not satisfied by the constructed assignment. Then there is a clause c in ϕ such that none of its literals are satisfied. Consider the three clause jobs c_1 , c_2 , and c_3 associated with the three literals in clause c . Each of these three jobs can only be scheduled (without creating a conflict with a dummy job) on the clause selection machines corresponding to c , and the validation machine corresponding to the variable appearing in the respective literal of the clause c . Since we only have two clause selection machines, at least one of the clause jobs c_1 , c_2 , and c_3 has to be scheduled on a validation machine. Assume c_1 is scheduled on a validation machine (the case of c_2 and c_3 is symmetric). Let x be the variable appearing in the first literal of c . Assume the variable job x^T is scheduled on the variable selection machine corresponding to x (the case where the variable job x^F is scheduled is symmetric). Then the variable job x^F has to be scheduled on the validation machine corresponding to x , since on all other machines it is in conflict with all dummy jobs. However, by construction of the validation machines, the clause job c_1 and the variable job x^F can only both be scheduled on the validation machine corresponding to x if setting x to true satisfies the first literal of c , a contradiction to the assumption that c is not satisfied. ◀

Finally, we have all ingredients to prove Theorem 3.

Proof of Theorem 3. To prove Theorem 3, we show that Construction 2 is polynomial-time many-one reduction from EXACT (3,4)-SAT to UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES. First, it is easy to observe that given an instance of EXACT (3,4)-SAT, the UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES instance specified by Construction 2 can be computed in polynomial time. The correctness of the reduction follows from Lemmas 7 and 8. Since EXACT (3,4)-SAT is NP-hard [29], we have that Theorem 3 follows. ◀

5 Conclusion

We proved that INTERVAL SCHEDULING ON ELIGIBLE MACHINES and its generalization INTERVAL SCHEDULING ON UNRELATED MACHINES are $W[1]$ -hard when parameterized by the number m of machines, and that UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES is NP-complete, answering two open questions by Mnich and van Bevern [22] and Sung and Vlach [28], respectively. With this, we contribute to the understanding of the (parameterized) computational complexity of basic and natural interval scheduling problems.

Our results leave two main open questions. Our NP-hardness proof for UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES does not imply NP-hardness of UNWEIGHTED INTERVAL SCHEDULING ON ELIGIBLE MACHINES, which leaves the following question.

► **Open Question 1.** *What is the computational complexity of UNWEIGHTED INTERVAL SCHEDULING ON ELIGIBLE MACHINES?*

Furthermore, the reduction used in our NP-hardness proof for UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES uses an unbounded number of machines, and hence does not imply $W[1]$ -hardness of UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES when parameterized by the number m of machines. Hence, we have the following question.

► **Open Question 2.** *What is the parameterized complexity of UNWEIGHTED INTERVAL SCHEDULING ON UNRELATED MACHINES when parameterized by the number m of machines?*

Lastly, we want to point out that the parameterized reduction we use to prove that INTERVAL SCHEDULING ON ELIGIBLE MACHINES and INTERVAL SCHEDULING ON UNRELATED MACHINES are $W[1]$ -hard when parameterized by the number m of machines roughly squares the parameter. More precisely, we reduce from MULTICOLORED CLIQUE parameterized by the number k of colors and for the number m of machines in the produced INTERVAL SCHEDULING ON ELIGIBLE MACHINES / INTERVAL SCHEDULING ON UNRELATED MACHINES instances we have $m \in O(k^2)$. This implies that assuming the Exponential Time Hypothesis (ETH) [14, 15], that there are no $f(m)(n+m)^{o(\sqrt{m})}$ algorithms for INTERVAL SCHEDULING ON ELIGIBLE MACHINES and INTERVAL SCHEDULING ON UNRELATED MACHINES for any function f [21]. However, the best known algorithms have running time $(n+m)^{O(m)}$ [2, 28]. Hence, there is still a gap between the upper and lower bound.

References

- 1 Enrico Angelelli, Nicola Bianchessi, and Carlo Filippi. Optimal interval scheduling with a resource constraint. *Computers & Operations Research*, 51:268–281, 2014.
- 2 Esther M. Arkin and Ellen B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18(1):1–8, 1987.
- 3 Kenneth R Baker and Gary D Scudder. Sequencing with earliness and tardiness penalties: a review. *Operations Research*, 38(1):22–36, 1990.
- 4 Matthias Bentert, René van Bevern, and Rolf Niedermeier. Inductive k -independent graphs and c -colorable subgraphs in scheduling: a review. *Journal of Scheduling*, 22(1):3–20, 2019.
- 5 René van Bevern, Matthias Mnich, Rolf Niedermeier, and Mathias Weller. Interval scheduling and colorful independent sets. *Journal of Scheduling*, 18(5):449–469, 2015.
- 6 René van Bevern, Rolf Niedermeier, and Ondřej Suchý. A parameterized complexity view on non-preemptively scheduling interval-constrained jobs: few machines, small looseness, and small slack. *Journal of Scheduling*, 20(3):255–265, 2017.

- 7 Khalid I Bouzina and Hamilton Emmons. Interval scheduling on identical machines. *Journal of Global Optimization*, 9(3):379–393, 1996.
- 8 Martin C Carlisle and Errol L Lloyd. On the k -coloring of intervals. *Discrete Applied Mathematics*, 59(3):225–235, 1995.
- 9 Ondřej Čepek and Shao Chin Sung. A quadratic time algorithm to maximize the number of just-in-time jobs on identical parallel machines. *Computers & operations research*, 32(12):3265–3271, 2005.
- 10 Michael R. Fellows, Danny Hermelin, Frances Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009.
- 11 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974.
- 12 Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.
- 13 Kunihiro Hiraiishi, Eugene Levner, and Milan Vlach. Scheduling of parallel identical machines to maximize the weighted number of just-in-time jobs. *Computers & Operations Research*, 29(7):841–848, 2002.
- 14 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k -SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.
- 15 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.
- 16 Antoon WJ Kolen, Jan Karel Lenstra, Christos H Papadimitriou, and Frits CR Spieksma. Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543, 2007.
- 17 Mikhail Y. Kovalyov, Chi To Ng, and T.C. Edwin Cheng. Fixed interval scheduling: Models, applications, computational complexity and algorithms. *European Journal of Operational Research*, 178(2):331–342, 2007.
- 18 John F Krafcik. Triumph of the lean production system. *Sloan Management Review*, 30(1):41–52, 1988.
- 19 Avital Lann and Gur Mosheiov. Single machine scheduling to minimize the number of early and tardy jobs. *Computers & Operations Research*, 23(8):769–781, 1996.
- 20 Yaron Leyvand, Dvir Shabtay, George Steiner, and Liron Yedidsion. Just-in-time scheduling with controllable processing times on parallel machines. *Journal of Combinatorial Optimization*, 19(3):347–368, 2010.
- 21 Daniel Lokshantov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of EATCS*, 3(105), 2013.
- 22 Matthias Mnich and René van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, 100:254–261, 2018.
- 23 Taiichi Ohno and Norman Bodek. *Toyota production system: beyond large-scale production*. Productivity Press, 1988.
- 24 Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.
- 25 Dvir Shabtay. The just-in-time scheduling problem in a flow-shop scheduling system. *European Journal of Operational Research*, 216(3):521–532, 2012.
- 26 Shigeo Shingo and Andrew P Dillon. *A revolution in manufacturing: the SMED system*. Productivity Press, 1985.
- 27 Frits C.R. Spieksma. On the approximability of an interval scheduling problem. *Journal of Scheduling*, 2(5):215–227, 1999.
- 28 Shao Chin Sung and Milan Vlach. Maximizing weighted number of just-in-time jobs on unrelated parallel machines. *Journal of Scheduling*, 8(5):453–460, 2005.
- 29 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.

18:16 Hardness of Interval Scheduling on Unrelated Machines

- 30 James P Womack and Daniel T Jones. Lean thinking – banish waste and create wealth in your corporation. *Journal of the Operational Research Society*, 48(11):1148–1148, 1997.
- 31 James P Womack, Daniel T Jones, and Daniel Roos. *The machine that changed the world: The story of lean production – Toyota’s secret weapon in the global car wars that is now revolutionizing world industry*. Simon and Schuster, 1990.
- 32 Mihalis Yannakakis and Fănică Gavril. The maximum k -colorable subgraph problem for chordal graphs. *Information Processing Letters*, 24(2):133–137, 1987.

Vertex Cover and Feedback Vertex Set Above and Below Structural Guarantees

Leon Kellerhals  

Faculty IV, Institute of Software Engineering and Theoretical Computer Science, Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Tomohiro Koana  

Faculty IV, Institute of Software Engineering and Theoretical Computer Science, Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Pascal Kunz  

Faculty IV, Institute of Software Engineering and Theoretical Computer Science, Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Abstract

VERTEX COVER parameterized by the solution size k is the quintessential fixed-parameter tractable problem. FPT algorithms are most interesting when the parameter is small. Several lower bounds on k are well-known, such as the maximum size of a matching. This has led to a line of research on parameterizations of VERTEX COVER by the difference of the solution size k and a lower bound. The most prominent cases for such lower bounds for which the problem is FPT are the matching number or the optimal fractional LP solution. We investigate parameterizations by the difference between k and other graph parameters including the feedback vertex number, the degeneracy, cluster deletion number, and treewidth with the goal of finding the border of fixed-parameter tractability for said difference parameterizations. We also consider similar parameterizations of the FEEDBACK VERTEX SET problem.

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

Keywords and phrases parameterized complexity, vertex cover, feedback vertex set, above guarantee parameterization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.19

Related Version *Full Version:* <https://arxiv.org/abs/2203.05887>

Funding *Tomohiro Koana:* Supported by the DFG Project DiPa, NI 369/21.

Pascal Kunz: Supported by DFG Research Training Group 2434 “Facets of Complexity”.

Acknowledgements This work was initiated at the research retreat of the Algorithmics and Computational Complexity group, TU Berlin, in 2021.

1 Introduction

Given an undirected graph G and an integer k , the VERTEX COVER problem asks whether there is a set of at most k vertices that contains at least one endpoint of each edge. VERTEX COVER is arguably the most well-studied problem in parameterized complexity. After significant efforts, the state-of-the-art FPT algorithm parameterized by the solution size k runs in time $\mathcal{O}(1.2738^k + kn)$ [4], where n is the number of vertices. Very recently, Harris and Narayanaswamy [22] have announced an even faster algorithm with running time $\mathcal{O}(1.2540^k \cdot n^{\mathcal{O}(1)})$.

The aforementioned FPT algorithm is only useful when the parameter k is small. In practice, however, the minimum vertex cover size is often large. For this reason, many recent studies look into VERTEX COVER where the parameterization is k minus a lower bound on k .



© Leon Kellerhals, Tomohiro Koana, and Pascal Kunz;
licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 19; pp. 19:1–19:14

Leibniz International Proceedings in Informatics



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

For instance, if the maximum matching size m is greater than k , then this would be a trivial no instance, since a vertex cover must contain at least one endpoint of each edge in any matching. This naturally gives rise to the “above guarantee” [33] parameter $k - m$. VERTEX COVER is FPT with respect to $k - m$ [37]. It has also been shown that VERTEX COVER is FPT for even smaller above guarantee parameters such as $k - r$ [8, 31] and $k - 2r + m$ [19], where r is the optimal LP relaxation value of VERTEX COVER. Kernelization with respect to these parameters has also been studied [28, 29].

This work considers above guarantee parameterizations of VERTEX COVER where the lower bounds are structural parameters not related to the matching number, such as feedback vertex number, degeneracy, and cluster vertex deletion number. We also study similar above guarantee parameterizations of the FEEDBACK VERTEX SET problem: Given a graph G and an integer k , it asks whether there is a set of at most k vertices whose deletion from G results in a forest. In this work, we do not deeply look into the “below guarantee” parameterization (where the number of vertices n is the most obvious upper bound) because VERTEX COVER and FEEDBACK VERTEX SET are known to be W[1]-hard when parameterized by $n - k$.¹

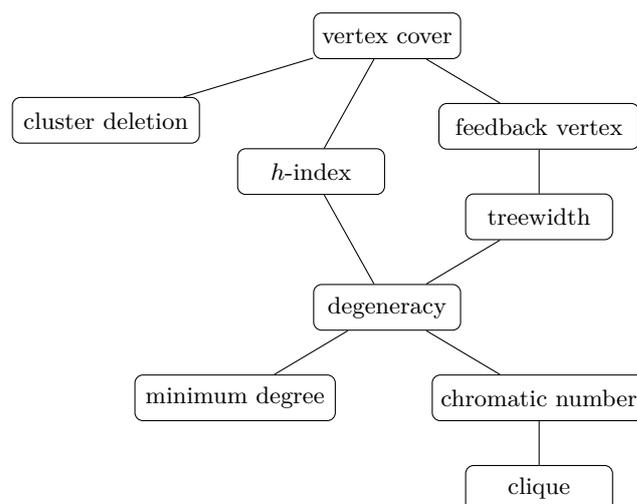
Motivation. We believe that FPT algorithms with above guarantee parameterizations may help explain the efficiency of some branching algorithms in practice. Consider an instance $I = (G, k)$ of VERTEX COVER for a complete graph G . This instance is trivial to solve: I has a solution if and only if $k \geq n - 1$. This triviality, however, is overlooked by the worse-case running time bound of FPT algorithms parameterized by the solution size k or the aforementioned smaller parameters $k - m$, $k - \ell$, or $k - 2\ell + m$, all of which amount to $n/2$ (for even n). Now consider another above guarantee parameter $k - (\omega - 1)$, where ω is the maximum clique size. (Note that for a clique C , a vertex cover contains at least $|C| - 1$ vertices of C .) As we will see, VERTEX COVER is FPT parameterized by this parameter (even if a maximum clique is not given). This gives us a theoretical reasoning as to why VERTEX COVER is indeed trivial to solve on complete graphs.

We also believe that above structural guarantee parameterizations are of theoretical interest, because they are closely related to identifying graphs in which two of its parameters coincide. Structural characterizations of such graphs have been extensively studied where the two parameters are maximum matching size and minimum vertex cover size [32], maximum matching size and minimum edge dominating set size [30], maximum matching size and induced matching size [3], maximum independent set size and minimum dominating set [35, 36], and minimum dominating set and minimum independent dominating set [21]. The corresponding computational complexity questions, i.e. whether these graphs can be recognized in polynomial time, have been studied as well [5, 11, 12, 16, 20, 40].

Finally, our parameterizations can be seen as “dual” of parameters studied in literature. There has been significant work (especially in the context of kernelization) on VERTEX COVER parameterized by structural parameters smaller than the solution size k such as feedback vertex set number [2, 7, 23, 26, 34].

Our contribution. In Section 3, we show that VERTEX COVER is FPT when parameterized by $k - h$ for the h -index h . This parameter is greater than or equal to many graph parameters such as degeneracy d , chromatic number χ , and clique number ω (See Figure 1). Thus, VERTEX COVER is FPT for $k - d$, $k - \chi$, and $k - \omega$ as well. Using a similar approach, we

¹ These two parameterized problem are essentially the W[1]-hard problems INDEPENDENT SET [14] and MAXIMUM INDUCED FOREST [27], respectively.



■ **Figure 1** A Hasse diagram of graph parameters. There is a line between two parameters p (above) and q (below) if $p + 1 \geq q$ holds for any graph G .

show in Section 4 that FEEDBACK VERTEX SET is FPT for $k - d$. We also show that on planar graphs, fixed-parameter tractability of VERTEX COVER with respect to $k - d$ can be strengthened: VERTEX COVER is FPT parameterized by $k - tw$ for the treewidth tw (Section 5). In the remaining sections, we prove hardness results. In Section 6, we show that VERTEX COVER admits no kernel of size polynomial in $k - \delta$ (δ is the minimum degree) and neither VERTEX COVER nor FEEDBACK VERTEX SET admit a kernel of size polynomial in $k - \omega$. We also show that VERTEX COVER is W[1]-hard for $k - fvs$ (Section 7) and NP-hard for $k - cd = 0$ (Section 8), where fvs and cd are the size of a minimum feedback vertex set and of a minimum cluster deletion set, respectively. Finally, we prove that FEEDBACK VERTEX SET NP-hard for $vc - k = 2$ in Section 9 where vc is the size of a minimum vertex cover.

2 Preliminaries

Graphs. For standard graph terminology, we refer to Diestel [13]. All graphs we consider are finite, undirected, and loopless. We call a function p that maps any graph G to an integer $p(G)$ a *graph parameter*. In the following, we will define several graph parameters that are of interest in this work. Let G be a graph. We denote the vertex set and edge set of G by $V(G)$ and $E(G)$, respectively. We denote the *minimum degree* of G by $\delta(G)$ and the *maximum degree* by $\Delta(G)$. The *vertex cover number* $vc(G)$ of G is the size of a smallest set $X \subseteq V(G)$ such that $G - X$ is edgeless. The *feedback vertex number* $fvs(G)$ of G is the size of a smallest set $X \subseteq V(G)$ such that $G - X$ is acyclic. The *h -index* $h(G)$ of a graph G is the largest integer k such that G contains at least k vertices each of degree at least k . The *degeneracy* of a graph G is $d(G) := \max_{V' \subseteq V(G)} \delta(G[V'])$. A subset V' of $V(G)$ that maximizes $\delta(G[V'])$ is a *core* of G . The *clique number* $\omega(G)$ of G is the size of a largest clique in G . The *chromatic number* $\chi(G)$ is the minimum integer k such that G can be properly k -colored. The *cluster deletion number* $cd(G)$ of G is the size of a smallest set $X \subseteq V(G)$ such that $G - X$ does not contain a P_3 as an induced subgraph. A pair $(T = (W, F), \beta)$ where T is a tree and $\beta: W \rightarrow 2^{V(G)}$ is a *tree decomposition* of G if (i) $\bigcup_{w \in W} \beta(w) = V(G)$, (ii) $\{w \in W \mid v \in \beta(w)\}$ induces

a connected subgraph of T for all $v \in V(G)$ and (iii) for all $\{u, v\} \in E(G)$, there exists a $w \in W$ with $u, v \in \beta(w)$. The *width* of $(T = (W, F), \beta)$ is $\max_{w \in W} |\beta(w)| - 1$. The *treewidth* $\text{tw}(G)$ of G is the minimum width over all tree decompositions of G .

If p and q are graph parameters, then we will say that p is *smaller* than q (and write $p \preceq q$), if there is a constant c such that $p(G) \leq q(G) + c$ for all graphs G . This differs from the way the boundedness relation between graph parameters is usually defined [25, 42], but this stricter definition is necessary in the context of difference parameterizations. This is because with this stricter definition the following is true (and easy to prove): If p, q, r are graph parameters such that $p \preceq q \preceq r$, then $r - q \preceq r - p$. Figure 1 depicts the graph parameters relevant to this work and the relationships between them.

Parameterized complexity. A *parameterized problem* is a pair (L, κ) where $L \subseteq \Sigma^*$ for a finite alphabet Σ and $\kappa: \Sigma^* \rightarrow \mathbb{N}$ is the parameter. The problem is *fixed-parameter tractable* (FPT) if it can be decided by an algorithm with running time $\mathcal{O}(f(\kappa(I)) \cdot |I|^c)$ where $I \in \Sigma^*$, f is a computable function and c is a constant. Note that if (L, κ) is FPT and $\kappa \preceq \kappa'$, then (L, κ') is also FPT. A *kernel* for this problem is a polynomial-time algorithm that takes the instance I and outputs a second instance I' such that (i) $I \in (L, \kappa) \iff I' \in (L, \kappa)$ and (ii) $|I'| \leq f(\kappa(I))$ for a computable function f . The *size* of the kernel is f . There is a hierarchy of computational complexity classes for parameterized problems: $\text{FPT} \subseteq \text{W}[1] \subseteq \dots \subseteq \text{XP}$. To show that a parameterized problem (L, κ) is (presumably) not FPT one may use a *parameterized reduction* from a W[1]-hard problem to L . A parameterized reduction from a parameterized problem (L, κ) to another parameterized problem (L', κ') is a function that acts as follows: For computable functions f and g , given an instance I of L , it computes in $f(\kappa(I)) \cdot |I|^{\mathcal{O}(1)}$ time an instance I' of L' so that $I \in (L, \kappa) \iff I' \in (L', \kappa')$ and $\kappa(I') \leq g(\kappa(I))$. For more details on parameterized algorithms and complexity, we refer to the standard literature [6, 15, 17].

Problem definitions. We are interested in above guarantee parameterizations of VERTEX COVER of the following form. Let $p \preceq \text{vc}$ be a graph parameter. Then, we define:

VERTEX COVER ABOVE p
Input: A graph G and an integer k .
Question: Does G contain a vertex cover of order at most k ?
Parameter: $\ell := k - p(G)$.

Similarly, we also consider above (below) guarantee parameterizations of FEEDBACK VERTEX SET. Now, let p be a graph parameter with $p \preceq \text{fvs}$ ($\text{fvs} \preceq p$). We consider the following problem:

FEEDBACK VERTEX SET ABOVE (BELOW) p
Input: A graph G and an integer k .
Question: Does G contain a feedback vertex set of order at most k ?
Parameter: $\ell := k - p(G)$ ($\ell := p(G) - k$).

3 Vertex Cover above h -Index

We start by proving that VERTEX COVER is FPT when parameterized by the difference between k and the h -index of the graph. Recall that the state-of-the-art algorithm for VERTEX COVER parameterized by the solution size k has running time $\mathcal{O}(1.274^k + kn)$ [4].

► **Theorem 1.** *VERTEX COVER ABOVE h -INDEX is FPT.*

Proof. Let (G, k) be an instance of VERTEX COVER where G is a graph with an h -index of h . Let $v_1, \dots, v_h \in V(G)$ with $\deg(v_i) \geq h$. We branch into the following $h + 1$ cases:

- (1) The solution contains all of the vertices v_1, \dots, v_h . Hence, we test the instance $(G - \{v_1, \dots, v_h\}, k - h)$ in time $\mathcal{O}(1.274^{k-h} + (k - h)n)$.
- (2) The solution does not contain v_i for some $i \in \{1, \dots, h\}$. Then, the solution must contain all of v_i 's neighbors. Hence, we test the instance $(G - N(v_i), k - |N(v_i)|)$. Since $|N(v_i)| \geq h$, this is possible in time $\mathcal{O}(1.274^{k-h} + (k - h)n)$.

In all, we get a running time of $\mathcal{O}(1.274^{k-h}h + (k - h)hn)$. ◀

This algorithm can also be used to obtain a Turing kernelization (cf. [18, Ch. 22]) by simply computing a kernel for each of the $h + 1$ instances of VERTEX COVER parameterized by the solution size that the algorithm branches into.

4 Feedback Vertex Set above Degeneracy

A similar approach to the one used in the previous section, branching once to lower k and then applying a known algorithm for the standard parameterization, can also be employed to show that FEEDBACK VERTEX SET ABOVE DEGENERACY is FPT. The fastest presently known deterministic algorithm for the standard parameterization of FEEDBACK VERTEX SET runs in time $\mathcal{O}(3.460^k \cdot n)$ [24].

► **Theorem 2.** *FEEDBACK VERTEX SET ABOVE DEGENERACY is FPT.*

Proof. Let (G, k) be an instance for FEEDBACK VERTEX SET where d is the degeneracy of G . It is well-known that the degeneracy and the core of a graph can be computed in polynomial time by iteratively deleting a minimum-degree vertex and storing the largest degree of a vertex at the time it is deleted. We start by computing a core V' of G . We branch into the following $|V'|^2 + 1$ cases.

- (1) The entire core is contained in the minimum feedback vertex set. The core must contain at least $d + 1$ vertices. Hence, we test the instance $(G - V', k - |V'|)$ in time $\mathcal{O}(3.460^{k-|V'|} \cdot n) = \mathcal{O}(3.460^{k-d} \cdot n)$.
- (2) The entire core is not contained in the minimum feedback vertex set. Let X denote the minimum feedback vertex and let $F := V(G) \setminus X$ be the maximum induced forest.
 - a. If $G[V' \cap F]$ contains an isolated vertex u , then all neighbors of u in $G[V']$, of which there are at least d , must be in X . Hence, for each $u \in V'$, we test the instance $(G - N_{G[V']}(u), k - \deg_{G[V']}(u))$ in time $\mathcal{O}(3.460^{k-\deg_{G[V']}(u)} \cdot n) = \mathcal{O}(3.460^{k-d} \cdot n)$.
 - b. If $G[V' \cap F]$ does not contain an isolated vertex, it must still contain a leaf u , since it is a forest. Then, all but one of the neighbors of u in $G[V']$ must be in X . Hence, for each pair $u \in V'$ and $v \in N_{G[V']}(u)$, we test the instance $(G - (N_{G[V']}(u) \setminus \{v\}), k - \deg_{G[V']}(u) + 1)$ in time $\mathcal{O}(3.460^{k-\deg_{G[V']}(u)+1} \cdot n) = \mathcal{O}(3.460^{k-d} \cdot n)$.

In all, this makes for a running time of $\mathcal{O}(3.460^{k-d} \cdot n^3)$. ◀

Like the algorithm in the previous section, this one can also be easily converted to a Turing kernelization.

5 Vertex Cover above Treewidth

In this section, we show that on planar graphs VERTEX COVER is also FPT with respect to $\ell = k - \text{tw}$, which is smaller than $k - d$.

► **Theorem 3.** *VERTEX COVER ABOVE TREewidth on planar graphs is FPT.*

Proof. Given a planar graph G , we compute the branchwidth β of G . This is possible in polynomial time, because G is planar [41]. Moreover, $\beta \leq \text{tw}(G) + 1 \leq \frac{3}{2}\beta$ [39, Theorem 5.1]. Any planar graph with treewidth w contains a $g \times g$ -grid with $g \geq \frac{w+4}{6}$ as a minor [38, Theorem 6.2]. Hence, having computed β , we know that G must contain a $g \times g$ -grid with $g \geq \frac{\beta+3}{6}$ as a minor. Any vertex cover of the $g \times g$ -grid must contain at least $\lfloor \frac{g}{2} \rfloor$ in each row, for a total of at least $g \cdot \lfloor \frac{g}{2} \rfloor \geq \frac{g(g-1)}{2}$ vertices. Since $\text{vc}(H_1) \leq \text{vc}(H_2)$, if H_1 is a minor of H_2 , it follows that $\text{vc}(G) \geq \frac{\beta^2-9}{72} =: r$. Hence, if $k < r$, we may reject the input. Otherwise, $\ell = k - \text{tw}(G) \geq k - \frac{3}{2}\beta \geq k - \frac{3}{2}\sqrt{72k-9}$. This means that ℓ is bounded from below by a function in k and, therefore, fixed-parameter tractability with respect to k implies fixed-parameter tractability with respect to ℓ . ◀

This algorithm relies on two properties of planar graphs: (i) large treewidth guarantees the existence of a $g \times g$ -grid where $g \in \Omega(\text{tw}^{1/2+\varepsilon})$ for $\varepsilon > 0$ and (ii) branchwidth can be computed in polynomial time on planar graphs. In any graph class that excludes a minor, (i) still holds true [9]. Although it is not clear that (ii) can be generalized, we remark that a constant approximation algorithm is known for graphs excluding single-crossing graphs as minors [10]. In fact, our result can be extended to any class of graphs that do not contain a single-crossing graph as a minor.

We leave open whether or not VERTEX COVER ABOVE TREewidth is FPT on graph classes that exclude a minor (other than planar graphs) or even on arbitrary graphs.

6 Kernelization Lower Bounds

In this section we show that, while there is a Turing kernel when parameterized above h -index, VERTEX COVER presumably does not admit a polynomial kernel when parameterized above the minimum degree or the clique number.

► **Theorem 4.** *VERTEX COVER ABOVE MINIMUM DEGREE and VERTEX COVER ABOVE CLIQUE NUMBER do not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP/poly}$.*

Proof. We prove the statement by giving a linear parametric transformation from CLIQUE parameterized by maximum degree and parameterized by the vertex cover number. Unless $\text{NP} \subseteq \text{coNP/poly}$, under neither parameterization does CLIQUE admit a polynomial kernel. This is folklore for maximum degree and was shown by Bodlaender et al. [1] for vertex cover number. The underlying reduction takes the CLIQUE instance (G, k) and transforms it into the instance (\bar{G}, \bar{k}) of VERTEX COVER where \bar{G} is the complement graph of G , that is $V(\bar{G}) := V(G)$ and $E(\bar{G}) := \binom{V(G)}{2} \setminus E(G)$, and $\bar{k} := |V| - k$. The reduction is obviously correct and computable in $\mathcal{O}(|V|^2)$ time. As for the parameterizations, observe that $\bar{k} - \delta(\bar{G}) = (|V| - k) - (|V| - 1 - \Delta(G)) \leq \Delta(G)$. Since $\omega(\bar{G}) \geq |V(G)| - \text{vc}(G)$, we also have $\bar{k} - \omega(\bar{G}) \leq (|V(G)| - k) - (n - \text{vc}(G)) \leq \text{vc}(G)$. This yields the claimed transformations. ◀

Using a standard reduction from VERTEX COVER to FEEDBACK VERTEX SET, we obtain the following.

► **Corollary 5.** *FEEDBACK VERTEX SET ABOVE CLIQUE NUMBER does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.*

Proof. We provide a linear parametric transformation from VERTEX COVER ABOVE CLIQUE NUMBER. Given an instance (G, ℓ) of VERTEX COVER, we use the following folklore construction to obtain an instance (G', ℓ) of FEEDBACK VERTEX SET. After initializing G' as a copy of G , we add for each edge $\{u, v\} = e \in E(G)$ the vertex w_{uv} and the edges $\{u, w_{uv}\}$ and $\{v, w_{uv}\}$ to G' , so that for each edge $e \in E(G)$ there exists a unique triangle in G' . Clearly, unless $\omega(G) = 2$, we have $\omega(G') = \omega(G)$. Hence, the parameter $\ell - \omega(G')$ is upper-bounded by the parameter of the input problem, and we are done. ◀

We leave open whether FEEDBACK VERTEX SET ABOVE MINIMUM DEGREE admits a polynomial kernel.

7 Vertex Cover above Feedback Vertex Number

In this section we prove that, when parameterizing above feedback vertex number, VERTEX COVER is $W[1]$ -hard. First, we prove $W[1]$ -hardness with respect to a related parameter, namely above the *distance to K_r -free* for every constant $r \geq 3$, that is, the minimum number of vertices one needs to remove such that the remaining graph does not contain a clique of order r . Distance to K_3 -free is a lower bound on the feedback vertex number, so our proof also implies hardness for VERTEX COVER ABOVE FEEDBACK VERTEX NUMBER.

► **Theorem 6.** *VERTEX COVER ABOVE DISTANCE TO K_r -FREE is $W[1]$ -hard.*

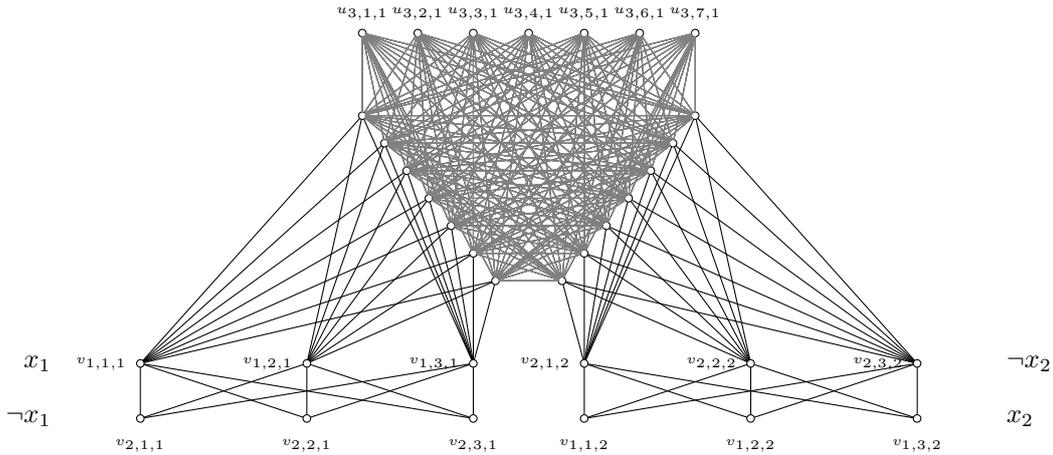
Proof. We provide a parameterized reduction from INDEPENDENT SET parameterized by the solution size k . Let $I = (G, k)$ be an instance of INDEPENDENT SET with $V(G) = [n]$ and $E(G) = \{e_1, \dots, e_m\}$. We create an instance (G', ℓ) as follows. First, for each $i \in [k]$ we add a clique on the vertex set $V_i = \{w_j^i \mid j \in [n]\}$ to G' . For each $i, j \in [k]$, we add the edge between w_q^i and w_q^j for each $q \in [n]$ and the edge between w_p^i and w_q^i for each $\{p, q\} \in E$. Next, for each $i \in [k]$ we add a set A_i of $r - 2$ vertices which form a clique, attach a leaf to each vertex in A_i , and make each vertex in A_i adjacent to each vertex in V_i . Let $A := \bigcup_{i \in [k]} A_i$. Then we add $k + 1$ cliques on $r - 1$ vertices. Call the set of these vertices B , attach a leaf to each $v \in B$, and make each $v \in B$ adjacent to each vertex in $\bigcup_{i \in [k]} V_i$. Denote by L the set of leaves in G' . Lastly, set $\ell := (n - 1)k + |A| + |B| = (n - 1)k + |L|$.

For the correctness, observe that G' contains a vertex cover of size k' if and only if G' contains an independent set of size

$$|V'| - \ell = nk + |A| + |B| + |L| - ((n - 1)k + |A| + |B|) = |L| + k.$$

Let $Y \subseteq V(G')$ be an independent set in G' . As it is always optimal to take leaves into an independent set, we may assume that Y contains all of L . Hence, we may assume that $Y \cap (A \cup B) = \emptyset$. Furthermore, Y contains at most one vertex of each clique on V_i , $i \in [k]$, and, by the construction of the edges in G' , Y can contain only such vertices in the cliques whose corresponding vertices in G are pairwise nonadjacent. Hence, G' contains an independent set of size $|L| + k$ if and only if G contains an independent set of size k , and the reduction is correct.

Finally, we will show that the distance to K_r -free of G' is exactly nk . This implies that the parameter of the output instance, if d is the distance to K_r -free of G' , is $k' = \ell - d = (n - 1)k + |A| + |B| - nk = |A| + |B| - k = (r - 2)k + (r - 1)(k + 1) - k$. Since r is a constant, this implies that k' is bounded in k . Let $D \subseteq V'$ be of minimum set such



■ **Figure 2** An excerpt of the graph G output by Construction 9: At the bottom are the vertex gadgets for the variables x_1 and x_2 . At the top is a clause gadget representing a clause that contains both x_1 and $\neg x_2$.

that $G' - D$ is K_r -free. Clearly, $D \cap L = \emptyset$ as L does not intersect any K_r . Furthermore, as every vertex $u \in A_i$ intersects a subset of the cliques that any vertex $v \in V_i$ intersects, we may exchange each vertex in $D \cap A_i$ with a vertex in V_i . As there are fewer than r vertices in each A_i we may assume that $D \cap A_i = \emptyset$. But then D must contain $n - 1$ vertices of each set V_i , hence, D contains all but k vertices from $\bigcup_{i \in [k]} V_i$. If however $v \notin D$ for one such $v \in \bigcup_{i \in [k]} V_i$, then D must contain at least one vertex from each clique in B . As there are $k + 1$ such cliques, $v \notin D$ contradicts D being minimum. Hence, $D := \bigcup_{i \in [k]} V_i$ is a K_r -deletion set of size nk . ◀

For $r = 3$, the deletion set D in the proof above is also a feedback vertex set. Hence, we obtain the following.

► **Corollary 7.** VERTEX COVER ABOVE FEEDBACK VERTEX NUMBER is $W[1]$ -hard.

Observe that in the proof of Theorem 6 we can specify a minimum deletion set. Hence, our hardness results also hold if a minimum deletion set is given as part of the input.

8 Vertex Cover above Cluster Deletion Number

Recall that the cluster deletion number is the minimum size of a set X such that $G - X$ is a cluster graph, i.e., every connected component of $G - X$ is a clique. We show that VERTEX COVER ABOVE CLUSTER DELETION NUMBER is NP-hard even if the parameter is zero.

► **Theorem 8.** VERTEX COVER ABOVE CLUSTER DELETION is NP-hard even if $\ell = 0$.

We will prove this theorem by reduction from 3-SAT. In fact, we prove a slightly stronger claim: VERTEX COVER is NP-hard when restricted to graphs G with $\text{cd}(G) = \text{vc}(G)$. The following construction is illustrated in Figure 2.

► **Construction 9.** Let φ be a Boolean formula in 3-CNF consisting of the clauses C_1, \dots, C_m over the variables x_1, \dots, x_n . We may assume that each clause of φ contains exactly three literals. For each $i \in \{1, \dots, m\}$, let $C_i = (L_i^1 \vee L_i^2 \vee L_i^3)$ where L_i^1, L_i^2 , and L_i^3 are literals.

We construct a graph G and an integer $k := 14m + 3n$ such that $\text{cd}(G) = \text{vc}(G)$ and $\text{vc}(G) \leq k$ if and only if φ is satisfiable. Each variable x_j is represented by a *variable gadget* consisting of six vertices $(v_{r,s,j})_{r \in \{1,2\}, s \in \{1,2,3\}}$ that induce a complete bipartite graph with three vertices in each color class. Each clause C_i is represented by a *clause gadget* consisting of twenty-one vertices $(u_{r,s,i})_{r \in \{1,2,3\}, s \in \{1, \dots, 7\}}$ that induce a complete tripartite graph with seven vertices in each color class. The two sides of the bipartition of a variable gadget correspond to its positive and negative literals and the three sides of the tripartition of a clause gadget correspond to the three literals the clause contains. All vertices in a side of a clause gadget are connected to all vertices in the side of a variable gadget if these two sides correspond to the literal of opposite sign. Formally, we let:

$$\begin{aligned} V(G) &:= \{u_{r,s,i} \mid r \in \{1,2,3\}, s \in \{1, \dots, 7\}, i \in \{1, \dots, m\}\} \\ &\quad \cup \{v_{r,s,j} \mid r \in \{1,2\}, s \in \{1,2,3\}, j \in \{1, \dots, n\}\} \text{ and} \\ E(G) &:= \{\{u_{r,s,i}, u_{r',s',i}\} \mid r, r' \in \{1,2,3\}, r \neq r', s, s' \in \{1, \dots, 7\}, i \in \{1, \dots, m\}\} \\ &\quad \cup \{\{v_{1,s,j}, v_{2,s',j}\} \mid s, s' \in \{1,2,3\}, j \in \{1, \dots, n\}\} \\ &\quad \cup \{\{u_{r,s,i}, v_{1,s',j}\} \mid s \in \{1, \dots, 7\}, s' \in \{1,2,3\}, L_i^r = x_j\} \\ &\quad \cup \{\{u_{r,s,i}, v_{2,s',j}\} \mid s \in \{1, \dots, 7\}, s' \in \{1,2,3\}, L_i^r = \neg x_j\}. \end{aligned}$$

► **Lemma 10.** *Let G be the graph output by Construction 9 and $X \subseteq V(G)$ be a minimum cluster deletion set. Then, in each clause gadget of G , X contains all vertices in two of the sides of the gadget and none of the vertices of the third side.*

Proof. For every $i \in \{1, \dots, m\}$, the deletion set X must contain either (i) all vertices from two sides of the clause gadget for C_i or (ii) all but one vertex from each side of this clause gadget, because if X omits two vertices from one side and an additional vertex from a second side these three vertices induce a P_3 .

In case (i), it only remains to show that X does not contain any of the vertices in the third side. Let $A := \{u_{r,1,i}, \dots, u_{r,7,i}\}$ with $r \in \{1,2,3\}$ be the vertices in this side. Let $\alpha_r := 1$, if $L_i^r = x_j$, and $\alpha_r := 2$, if $L_i^r = \neg x_j$. If X contains all of the vertices in $B := \{v_{\alpha_r,1,j}, v_{\alpha_r,2,j}, v_{\alpha_r,3,j}\}$, then $X \setminus A$ is a cluster deletion set strictly smaller than X . If X does not contain all vertices in B , then it must contain all but one of the vertices in A (i.e., $|X \cap A| \geq 6$). Then $(X \setminus A) \cup B$ is a cluster deletion set and $|(X \setminus A) \cup B| \leq |X| - 6 + 3 < |X|$.

For case (ii), let r, r', r'' be the three sides of the clause gadget and let $X_{r''}$ be the set of vertices in X that lie in side r'' . Assume that X does not contain all vertices from two sides r, r' . Since X must contain all but one vertex from each side, we may assume without loss of generality that $X_r = \{u_{r,1,i}, \dots, u_{r,6,i}\}$, $|X_{r'}| = \{u_{r',1,i}, \dots, u_{r',6,i}\}$, and $|X_{r''}| \geq 6$ (note that it may contain all vertices in the third side r''). Let $\alpha_{r''} := 1$, if $L_i^{r''} = x_j$, and $\alpha_{r''} := 2$, if $L_i^{r''} = \neg x_j$ and let

$$X' := (X \setminus X_{r''}) \cup \{u_{r,7,i}, u_{r',7,i}\} \cup \{v_{\alpha_{r''},s,j} \mid s \in \{1,2,3\}\}.$$

Then, $|X'| \leq |X| - 6 + 2 + 3 = |X| - 1$. Moreover, X' is a cluster deletion set in G , because $X \setminus X' \subseteq \{u_{r'',s,i} \mid s \in \{1, \dots, 7\}\}$, but all of these vertices are isolated and, therefore, not part of any P_3 in $G - X'$. Hence, X is not minimum. ◀

A similar statement is also true for vertex gadgets:

► **Lemma 11.** *Let G be the graph output by Construction 9 and $X \subseteq V(G)$ be a minimum cluster deletion set. Then, in each variable gadget of G , X contains all vertices in one of the two sides of the gadget.*

19:10 Vertex Cover and Feedback Vertex Set Above and Below Structural Guarantees

Proof. If X does not contain all vertices in either side of a vertex gadget corresponding to the variable x_j , it must contain all but one vertex from each side. Let C_{i_1}, \dots, C_{i_t} be the clauses that contain the literal x_j and let r_1, \dots, r_t be the sides of each of the corresponding clause gadgets whose vertices are adjacent to the side in the vertex gadget of x_j . Then, X must contain all but one vertex in the side $r_{t'}$ of the clause gadget for $C_{i_{t'}}$, for each $t' \in \{1, \dots, t\}$. By Lemma 10, it follows that X contains all vertices in each of those sides. Hence, removing from X all vertices in the $r = 1$ side of x_j 's vertex gadget and adding the remaining vertex in the $r = 2$ side yields a smaller cluster deletion set. \blacktriangleleft

► **Lemma 12.** *Let G be the graph output by Construction 9 and $C \subseteq V(G)$ be a minimum cluster deletion set. Then, C is a vertex cover of G . Hence, $\text{cd}(G) = \text{vc}(G)$.*

Proof. Clearly $\text{cd}(G) \leq \text{vc}(G)$. We show that $\text{cd}(G) \geq \text{vc}(G)$. By Lemmas 10 and 11, C covers all edges within each clause and each variable gadget. It remains to show that edges between these gadgets are covered. The only such edges are between sides of a clause gadget and sides of a variable gadget when these two sides correspond to the same literal. Then, C must contain all vertices in one of these two sides, since, otherwise, $G - C$ would contain an induced P_3 . Hence, C also covers all edges between those two sides. \blacktriangleleft

► **Lemma 13.** *Let φ be a formula in 3-CNF and G the graph output by Construction 9 on input φ . Then, φ is satisfiable if and only if $\text{vc}(G) \leq \ell$.*

Proof. First, suppose that φ is satisfiable and that $\alpha: \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$ is a satisfying assignment. We extend α to literals on this variable set in the natural way. Since α satisfies every clause in φ , there is an $\alpha_i \in \{1, 2, 3\}$ for every $i \in \{1, \dots, m\}$ such that $\alpha(L_i^{\alpha_i}) = 1$. Let

$$\begin{aligned} C := & \{u_{r,s,i} \mid r \in \{1, 2, 3\} \setminus \{\alpha_i\}, s \in \{1, \dots, 7\}, i \in \{1, \dots, m\}\} \\ & \cup \{v_{1,s,j} \mid s \in \{1, 2, 3\}, j \in \{1, \dots, n\}, \alpha(x_j) = 1\} \\ & \cup \{v_{2,s,j} \mid s \in \{1, 2, 3\}, j \in \{1, \dots, n\}, \alpha(x_j) = 0\}. \end{aligned}$$

First, note that $|C| = 14m + 3n = k$. Secondly, we claim that C is a vertex cover of G . Clearly, all edges within clause gadgets and all edges within vertex gadgets are covered, because C contains all but one side in each of those gadgets. Edges between a vertex gadget and a clause gadget are covered because C contains vertices in sides of vertex gadgets, unless the literal this side corresponds to is not satisfied by α , but if this is the case, then C contains all sides of clause gadgets that correspond to this literal.

Now, suppose that $C \subseteq V(G)$, $|C| \leq k$, is a vertex cover of G . We may assume that C is minimum. Hence, by Lemmas 10–12, it contains all vertices in at two sides of every clause gadget and all vertices in exactly one side of every variable gadget. Let $\alpha: \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$ with:

$$\alpha(x_j) := \begin{cases} 1, & \text{if } \{v_{1,s,j} \mid s \in \{1, 2, 3\}\} \subseteq C, \\ 0, & \text{if } \{v_{2,s,j} \mid s \in \{1, 2, 3\}\} \subseteq C. \end{cases}$$

We claim that α satisfies φ . Let C_i be a clause in φ . One of the three sides of the gadget representing C_i is not contained in C . This side corresponds to the literal $L_i^r \in \{x_j, \neg x_j\}$. If $L_i^r = x_j$, then all vertices in $\{u_{r,s,i} \mid s \in \{1, \dots, 7\}\}$ are adjacent to all vertices in $\{v_{1,s,j} \mid s \in \{1, 2, 3\}\}$. Since $\{u_{r,s,i} \mid s \in \{1, \dots, 7\}\} \not\subseteq C$, it follows that $\{v_{1,s,j} \mid s \in \{1, 2, 3\}\} \subseteq C$ and, therefore, $\alpha(x_j) = 1$. Hence, α satisfies the clause C_i . The case where $L_i^r = \neg x_j$ is analogous. \blacktriangleleft

Theorem 8 follows from the preceding lemmas.

Proof of Theorem 8. Clearly, Construction 9 can be computed in polynomial time. The claim follows by Lemmas 12 and 13. ◀

We remark that the NP-hardness of VERTEX COVER ABOVE CLUSTER DELETION NUMBER holds even if we are given a minimum cluster deletion set as part of the input. To show this, we slightly adapt Construction 9. Let (G, k) be an instance given in Construction 9. We further introduce $7m/3 + n$ complete graphs on three vertices (we may assume that m is divisible by 3, otherwise we add dummy clauses). Denote these vertices by T and observe that $|T| = 7m + 3n$. We add an edge between each vertex in $V(G)$ and each vertex in T . Let H denote the resulting graph. Observe that $V(G)$ is a cluster vertex deletion set of size $21m + 6n$ of H . Moreover, $\text{cd}(H) \geq \text{cd}(G) + |T| \geq 21m + 6n$. It is not difficult to show that any vertex cover of size at most $21m + 6n$ of G contains every vertex of T . Thus, H has a vertex cover of size at most $21m + 6n$ if and only if G has a vertex cover of size $21m + 6n - |T| = k$. Lemma 13 establishes the correctness of the reduction.

9 Feedback Vertex Set below Vertex Cover

FEEDBACK VERTEX SET BELOW n is generally known as the MAXIMUM INDUCED FOREST problem and is known to be $W[1]$ -hard with respect to the solution size [27]. In the following, we consider FEEDBACK VERTEX SET BELOW VERTEX COVER, essentially the same problem but with a slightly smaller parameter. We show that this change is sufficient to make the problem NP-hard even if the parameter is fixed at two.

► **Theorem 14.** FEEDBACK VERTEX SET BELOW VERTEX COVER is NP-hard even if $\ell = \text{vc}(G) - \text{fvs}(G) = 2$.

Proof. We reduce from FEEDBACK VERTEX SET. Let $I = (G, k)$ be an instance of the latter problem. We assume without loss of generality that G has at least one edge. Let $\lambda = |V(G)| - k - 2$. We construct a graph H with $\text{vc}(H) = |V(G)|$ that contains a feedback vertex set of size at most $k' = |V(G)| - 2$ if and only if G contains a feedback vertex set of size at most k . Note that then $\ell = \text{vc}(H) - k' = 2$.

The construction of H is as follows: Add a copy of G to H . Attach a leaf to every vertex $v \in V(G)$, that is, add a new vertex u_v and the edge $\{v, u_v\}$ to H . Add a set V^* of λ vertices to $V(H)$ and make each $u \in V^*$ adjacent to every vertex in $V(G)$.

Suppose G contains a feedback vertex set S of size at most k . Then the set $S \cup V^*$ is a feedback vertex set of size at most $k + \lambda = |V(G)| - 2 = k'$ for H , as $H - (S \cup V^*)$ is isomorphic to the forest $G - S$ with a leaf attached to every vertex.

Conversely, suppose that H contains a feedback vertex set S' of size at most k' . We claim that there exists a feedback vertex set of size at most k' in H that contains none of the leaves u_v and all vertices in V^* . Clearly, if S' contains a leaf u_v attached to some $v \in V(G)$, then $(S' \setminus \{u_v\}) \cup \{v\}$ is also a feedback vertex set of size at most $|V(G)| - 2$ in H . Hence, we may assume that $V^* \subseteq S' \subseteq V^* \cup V(G)$.

Next, suppose that $V^* \setminus S' \neq \emptyset$. If $|V^* \setminus S'| \geq 2$, then S' contains at least $|V(G)| - 1$ vertices of $V(G)$ since otherwise two vertices in $V^* \setminus S'$ and two vertices in $V(G)$ form a cycle of length four. Note, however, that $|S'| \leq k' = |V(G)| - 2$. Thus, we may assume that $|V^* \setminus S'| \leq 1$. Towards showing that $V^* \setminus S' = \emptyset$, suppose that there is a vertex $w \in V^* \setminus S'$. Then, for every edge $\{u, v\} \in E(G)$, we have $u \in S'$ or $v \in S'$, as otherwise u, v , and w induce a cycle in $H - S'$. Thus, $S' \setminus V^*$ is a vertex cover of G . Let $x \in S' \cap V(G)$ be

arbitrary. Such a vertex exists by the assumption that G has at least one edge. We claim that $S^* = (S' \setminus \{x\}) \cup \{w\}$ is a feedback vertex set for H of size at most k' . If it is not, then $H - S^*$ has a cycle that contains x . Let y and z be two neighbors of x in this cycle. Note that $y, z \in V(G) \setminus S'$. Since $S' \cap V(G)$ is a vertex cover of G , it follows that the neighborhoods of y and z in $H - S^*$ are $\{x, u_y\}$ and $\{x, u_z\}$, respectively. The vertices u_y and u_z have degree one, and thus we have a contradiction to the existence of the aforementioned cycle. Thus, the claim follows. Lastly, having a solution S' with $V^* \subseteq S' \subseteq V^* \cup V(G)$ of size at most k' implies that $S = S' \setminus V^*$ is a feedback vertex set of size at most k for G .

Finally, to show that $\ell = \text{fvs}(H) - \text{vc}(H) = 2$, we need to show that $\text{vc}(H) = \text{fvs}(H) + 2 = |V(G)|$. As $V(G)$ is a vertex cover of H , we have $\text{vc}(H) \leq |V(G)|$. Since $\{\{v, u_v\} \mid v \in V(G)\}$ is a matching of size $|V(G)|$ in H , we have $\text{vc}(H) \geq |V(G)|$. \blacktriangleleft

10 Conclusion

The goal of this work is to extend the above guarantee paradigm in parameterized complexity beyond the previously considered lower bounds on vertex cover, namely the maximum matching size and the optimal LP relaxation solution. We approached this issue by considering various structural graph parameters that are upper-bounded by the vertex cover number. This work sketches a rough contour of the parameterized complexity landscape of these kinds of parameterizations of both VERTEX COVER and FEEDBACK VERTEX SET. It raises a number of immediate open questions, of which we highlight four:

- (i) Is VERTEX COVER ABOVE TREEWIDTH also fixed-parameter tractable on arbitrary graphs? We also leave this question open for FEEDBACK VERTEX SET.
- (ii) In Section 7, we showed that VERTEX COVER ABOVE FEEDBACK VERTEX NUMBER is W[1]-hard. A natural question to ask is whether this problem is NP-hard for a constant parameter value or whether it is in XP, that is, whether it can be decided by an algorithm with running time $\mathcal{O}(n^{f(\ell)})$ for an arbitrary computable function f .
- (iii) One can naturally generalize graph parameters like feedback vertex number or cluster deletion number by fixing a graph class \mathcal{F} and defining the \mathcal{F} -free deletion number of any graph G as the size of a smallest set $X \subseteq V(G)$ such that $G - X$ does not contain any $H \in \mathcal{F}$ as an induced subgraph. If \mathcal{F} contains a graph that is not edgeless, then vertex cover number upper-bounds the \mathcal{F} -free deletion number. It would be interesting to find a graph class \mathcal{F} such that VERTEX COVER ABOVE \mathcal{F} -FREE DELETION is FPT or to rule out the existence of such a class. We have only answered this question (always in the negative) if \mathcal{F} is any of the following classes: all cycles, $\{P_3\}$, all complete graphs.
- (iv) Moving beyond parameterized complexity, can graphs in which the difference parameters we have considered are small be characterized in an elegant way? For instance, one can easily prove that $\text{vc}(G) = \text{fvs}(G)$ if and only if G is edgeless. We are not aware of any simple characterization of graphs where $\text{vc}(G) - \text{fvs}(G) = 1$ or $\text{vc}(G) - \text{fvs}(G) \leq c$ for a larger constant c . Such a characterization could be useful for answering (ii).

References

- 1 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014. doi:10.1137/120880240.
- 2 Marin Bougeret, Bart M. P. Jansen, and Ignasi Sau. Bridge-depth characterizes which structural parameterizations of vertex cover admit a polynomial kernel. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 16:1–16:19, 2020. doi:10.4230/LIPIcs.ICALP.2020.16.

- 3 Kathie Cameron and Tracy Walker. The graphs with maximum induced matching and maximum matching the same size. *Discrete Mathematics*, 299(1-3):49–55, 2005. doi:10.1016/j.disc.2004.07.022.
- 4 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010. doi:10.1016/j.tcs.2010.06.026.
- 5 Václav Chvátal and Peter J. Slater. A note on well-covered graphs. In John Gimbel, John W. Kennedy, and Louis V. Quintas, editors, *Quo Vadis, Graph Theory?*, pages 179–181. Elsevier, 1993. doi:10.1016/S0167-5060(08)70387-X.
- 6 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 7 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. On the hardness of losing width. *Theory of Computing Systems*, 54(1):73–82, 2014. doi:10.1007/s00224-013-9480-1.
- 8 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *ACM Transactions on Computation Theory*, 5(1):3:1–3:11, 2013. doi:10.1145/2462896.2462899.
- 9 Erik D. Demaine and MohammadTaghi HajiAghayi. Linearity of grid minors in treewidth with applications through bidimensionality. *Combinatorica*, 28(1):19–36, 2008. doi:10.1007/s00493-008-2140-4.
- 10 Erik D Demaine, MohammadTaghi Hajiaghayi, Naomi Nishimura, Prabhakar Ragde, and Dimitrios M Thilikos. Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. *Journal of Computer and System Sciences*, 69(2):166–195, 2004. doi:10.1016/j.jcss.2003.12.001.
- 11 Marc Demange and Tınaz Ekim. Efficient recognition of equimatchable graphs. *Information Processing Letters*, 114(1-2):66–71, 2014. doi:10.1016/j.ipl.2013.08.002.
- 12 Zakir Deniz, Tınaz Ekim, Tatiana Romina Hartinger, Martin Milanic, and Mordechai Shalom. On two extensions of equimatchable graphs. *Discrete Optimization*, 26:112–130, 2017. doi:10.1016/j.disopt.2017.08.002.
- 13 Reinhard Diestel. *Graph Theory*. Springer, 5th edition, 2016. doi:10.1007/978-3-662-53622-3.
- 14 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. doi:10.1016/0304-3975(94)00097-3.
- 15 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 16 Márcio Antônio Duarte, Felix Joos, Lucia Draque Penso, Dieter Rautenbach, and Uéverton S. Souza. Maximum induced matchings close to maximum matchings. *Theoretical Computer Science*, 588:131–137, 2015. doi:10.1016/j.tcs.2015.04.001.
- 17 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006. doi:10.1007/3-540-29953-X.
- 18 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.
- 19 Shivam Garg and Geevarghese Philip. Raising the bar for vertex cover: Fixed-parameter tractability above a higher guarantee. In *Proceedings of the 2016 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1152–1166, 2016. doi:10.1137/1.9781611974331.ch80.
- 20 Fanica Gavril. Testing for equality between maximum matching and minimum node covering. *Information Processing Letters*, 6(6):199–202, 1977. doi:10.1016/0020-0190(77)90068-0.

- 21 Wayne Goddard and Michael A. Henning. Independent domination in graphs: A survey and recent results. *Discrete Mathematics*, 313(7):839–854, 2013. doi:10.1016/j.disc.2012.11.031.
- 22 David G. Harris and N. S. Narayanaswamy. A faster algorithm for vertex cover parameterized by solution size, 2022. arXiv:2205.08022.
- 23 Eva-Maria C. Hols, Stefan Kratsch, and Astrid Pieterse. Elimination distances, blocking sets, and kernels for vertex cover. In *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 36:1–36:14, 2020. doi:10.4230/LIPIcs.STACS.2020.36.
- 24 Yoichi Iwata and Yusuke Kobayashi. Improved analysis of highest-degree branching for feedback vertex set. *Algorithmica*, 83(8):2503–2520, 2021. doi:10.1007/s00453-021-00815-w.
- 25 Bart M. P. Jansen. *The Power of Data Reduction: Kernels for Fundamental Graph Problems*. PhD thesis, Utrecht University, 2013. URL: <http://dspace.library.uu.nl/handle/1874/276438>.
- 26 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.
- 27 Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002. doi:10.1016/S0304-3975(01)00414-5.
- 28 Stefan Kratsch. A randomized polynomial kernelization for vertex cover with a smaller parameter. *SIAM Journal on Discrete Mathematics*, 32(3):1806–1839, 2018. doi:10.1137/16M1104585.
- 29 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *Journal of the ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.
- 30 M. Lesk, M. D. Plummer, and W. R. Pulleyblank. Equi-matchable graphs. *Graph theory and combinatorics*, pages 239–254, 1984.
- 31 Daniel Lokshtanov, 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.
- 32 László Lovász and Michael D. Plummer. *Matching theory*. North-Holland, 1986.
- 33 Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31(2):335–354, 1999. doi:10.1006/jagm.1998.0996.
- 34 Diptapriyo Majumdar, Venkatesh Raman, and Saket Saurabh. Polynomial kernels for vertex cover parameterized by small degree modulators. *Theory of Computing Systems*, 62(8):1910–1951, 2018. doi:10.1007/s00224-018-9858-1.
- 35 Michael D. Plummer. Some covering concepts in graphs. *Journal of Combinatorial Theory*, 8(1):91–98, 1970. doi:10.1016/S0021-9800(70)80011-4.
- 36 Michael D. Plummer. Well-covered graphs: A survey. *Quaestiones Mathematicae*, 16(3):253–287, 1993. doi:10.1080/16073606.1993.9631737.
- 37 Igor Razgon and Barry O’Sullivan. Almost 2-SAT is fixed-parameter tractable. *Journal of Computer and System Sciences*, 75(8):435–450, 2009. doi:10.1016/j.jcss.2009.04.002.
- 38 N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory Series B*, 62(2):323–348, 1994. doi:10.1006/jctb.1994.1073.
- 39 Neil Robertson and P. D. Seymour. Graph minors X: Obstructions to tree-decomposition. *Journal of Combinatorial Theory Series B*, 52(2):153–190, 1991. doi:10.1016/0095-8956(91)90061-N.
- 40 Ramesh S. Sankaranarayanan and Lorna K. Stewart. Complexity results for well-covered graphs. *Networks*, 22(3):247–262, 1992. doi:10.1002/net.3230220304.
- 41 P. D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994. doi:10.1007/BF01215352.
- 42 Manuel Sorge and Mathias Weller. The graph parameter hierarchy. Unpublished manuscript, 2019. URL: <https://manyu.pro/assets/parameter-hierarchy.pdf>.

Parameterized Local Search for Vertex Cover: When Only the Search Radius Is Crucial

Christian Komusiewicz ✉ 

Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, Germany

Nils Morawietz ✉

Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, Germany

Abstract

A k -swap W for a vertex cover S of a graph G is a vertex set of size at most k such that $S' = (S \setminus W) \cup (W \setminus S)$, the symmetric difference of S and W , is a vertex cover of G . If $|S'| < |S|$, then W is *improving*. In LS-VERTEX COVER, one is given a vertex cover S of a graph G and wants to know if there is an improving k -swap for S in G . In applications of LS-VERTEX COVER, k is a very small parameter that can be set by a user to determine the trade-off between running time and solution quality. Consequently, k can be considered to be a constant. Motivated by this and the fact that LS-VERTEX COVER is W[1]-hard with respect to k , we aim for algorithms with running time $\ell^{f(k)} \cdot n^{\mathcal{O}(1)}$ where ℓ is a structural graph parameter upper-bounded by n . We say that such a running time *grows mildly with respect to ℓ and strongly with respect to k* . We obtain algorithms with such a running time for ℓ being the h -index of G , the treewidth of G , or the modular-width of G . In addition, we consider a novel parameter, the maximum degree over all quotient graphs in a modular decomposition of G . Moreover, we adapt these algorithms to the more general problem where each vertex is assigned a weight and where we want to find a d -improving k -swap, that is, a k -swap which decreases the weight of the vertex cover by at least d .

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Computing methodologies \rightarrow Discrete space search

Keywords and phrases Local Search, Structural parameterization, Fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.20

Funding *Nils Morawietz*: Supported by the Deutsche Forschungsgemeinschaft (DFG), project OPERAH, KO 3669/5-1.

1 Introduction

Local search is one of the most successful heuristic strategies to tackle hard optimization problems [6, 16, 20]. Consequently, understanding when local search yields good results and improving local search approaches is of utmost importance. In its easiest form, local search follows a hill-climbing approach on the space of feasible solutions of the optimization problem at hand. In this setting, one chooses some initial feasible solution and then iteratively replaces the current solution by a better one in its local neighborhood until reaching a local optimum, that is, a solution that has no better solution in its neighborhood. Intuitively, it is clear that the larger the local search neighborhood, the better the final solution will be. At the same time, searching a larger neighborhood takes longer. In particular, for hard optimization problems, the running time will be superpolynomial when the neighborhood is too large. As a consequence, there is a trade-off between running time and solution quality that is governed by the size of the local search neighborhood.

Parameterized local search offers a framework that may guide the design process for algorithms that attempt to search larger local neighborhoods. When applying parameterized local search to an optimization problem, the first step is to define a measure of distance



© Christian Komusiewicz and Nils Morawietz;

licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 20; pp. 20:1–20:18

Leibniz International Proceedings in Informatics



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

between solutions. The local search neighborhood of a solution is then the set of solutions within distance at most k . Here, k is an *operational* parameter that can be set by the user and that does not depend on the input data. The hope is now that the superpolynomial part of the running time for searching the local neighborhood depends mostly on k . More precisely, the ultimate goal of parameterized local search is to devise an algorithm that determines in $f(k) \cdot n^{\mathcal{O}(1)}$ time whether there exists a better solution within distance at most k of the current one. Often such a running time is not possible, since most local search problems turn out to be W[1]-hard with respect to the parameter k [5, 10, 15, 14, 21, 24].

For example, when applying local search to VERTEX COVER, the set of feasible solutions of a graph $G = (V, E)$ is naturally defined as the collection of vertex covers of G , that is, vertex sets $S \subseteq V$ that cover all edges of the graph. The most obvious choice for a local search neighborhood is the k -swap neighborhood. Here, two vertex sets S and S' are k -swap neighbors if and only if $(S \setminus S') \cup (S' \setminus S)$ has size at most k . The problem of deciding whether a given vertex cover S of a graph G has a smaller vertex cover in its k -swap neighborhood, called LS-VERTEX COVER, is W[1]-hard with respect to k [10]. Thus, at first it may seem unlikely that parameterized local search can be successfully applied to LS-VERTEX COVER. There are, however, some positive results for LS-VERTEX COVER. In particular, LS-VERTEX COVER admits an FPT-algorithm for $\Delta(G) + k$, where $\Delta(G)$ is the maximum degree of the input graph [10]. That is, it can be solved in $f(\Delta(G), k) \cdot n^{\mathcal{O}(1)}$ time. While this running time bound is certainly interesting for bounded-degree graphs, it does not necessarily deliver on the promise of parameterized local search that the superpolynomial part of the running time depends mostly on k : for example $f(\Delta(G), k)$ could be $2^{\Delta(G) \cdot k}$. It is also known, however, that LS-VERTEX COVER can be solved in time $\mathcal{O}(2^k \cdot (\Delta(G) - 1)^{k/2} \cdot k^3 \cdot n)$ [18]. In this running time only k appears in the exponent, while $\Delta(G)$ appears only in the base of the exponential function. Consequently, for small values of k the running time guarantee can still be practically relevant, even when Δ is not too small. In particular, the running time is polynomial for every fixed k . The practical usefulness of the algorithm with this worst-case running time bound was confirmed by experiments which showed that LS-VERTEX COVER can be solved efficiently for k up to 25 [18].

In this work, we aim to find further algorithms for LS-VERTEX COVER that achieve such running times which can be considered practical even though the superpolynomial running time part depends not only on the operational parameter k . Before describing our results, let us formalize the class of running time functions that we aim to achieve.

► **Definition 1.1.** *Let $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ be a function. We say that f grows mildly with respect to ℓ and strongly with respect to k if $f(\ell, k) \in \mathcal{O}(\ell^{g(k)})$ for some computable function g depending only on k .*

We are interested in obtaining FPT-algorithms whose running time grows strongly only with respect to k and mildly with respect to some other parameters. In our opinion, the usefulness of this setting is not limited to local search problems. Instead, it may be useful whenever

- two parameters k and ℓ are studied,
- k is known to be very small on relevant input instances,
- k is known to be much smaller than ℓ on these instances,
- and the problem is W[1]-hard with respect to k .

Our Results. We provide FPT-algorithms for LS-VERTEX COVER parameterized by k and several structural parameters of G . Besides k , we consider the treewidth of the input graph G , denoted by $\text{tw}(G)$, the h -index of the input graph G , denoted by $h(G)$, the modular-width

of G , denoted by $\text{mw}(G)$, and a novel parameter, the maximum degree over all quotient graphs in a minimum-width modular decomposition, denoted by $\Delta_{\text{md}}(G)$. In all our FPT-algorithms, the running time grows strongly with respect to k and only mildly with respect to the particular structural parameter. Moreover, for all these algorithms, the running time depends only linearly on the size of the input graph.

The most general of our algorithms actually solve GAP LS-WEIGHTED VERTEX COVER where the input graph is vertex-weighted, the cost of a vertex cover is the sum of its vertex weights, and we search for a swap that improves the current solution by at least d for some input value d . Local search approaches for WEIGHTED VERTEX COVER have been studied from a more practical perspective which motivates our study of weighted variants of LS-VC. In addition, for weighted local search problems, there may be exponentially long chains of local improvements before one finds a local optimum [17] even for swaps of constant size [19]. Here, using a gap-variant of local search could reduce the number of necessary steps by increasing the improvement per step. We now discuss the results in detail.

The h -index of a graph G is the largest number h such that G has at least h vertices with degree at least h [9]. For GAP LS-WEIGHTED VERTEX COVER, we obtain an algorithm with running time $\mathcal{O}(k! \cdot (h(G) - 1)^k \cdot n)$. This can be seen as an improvement over the FPT-algorithm for $\Delta(G)$ and k [18] since $h(G)$ is never larger than $\Delta(G)$. In fact, in many real-world instances the input graphs are scale-free, and on scale-free graphs $h(G)$ is drastically smaller than $\Delta(G)$. Even in such graphs, in order to speak of an improvement, it is imperative that the running time of the FPT-algorithm grows mildly with respect to $h(G)$ and strongly with respect to k : a running time of $\mathcal{O}(2^{h(G) \cdot k} \cdot n)$ would be less desirable than the previous one for $\Delta(G)$ and k since the exponent would not be confined to the operational parameter k .

The FPT-algorithm for $\text{tw}(G)$ and k has running time $\mathcal{O}((\text{tw}(G)^{3k} + k^2) \cdot n)$. It is based on dynamic programming on the tree decomposition. For LS-VERTEX COVER, we show that, for each bag of the tree decomposition, it is sufficient to consider intersections of size at most $\lceil \frac{k}{2} \rceil$ with potential improving swaps. This reduces the running time for LS-VERTEX COVER to $\mathcal{O}((\text{tw}(G)^{3 \cdot \lceil \frac{k}{2} \rceil} + k^2) \cdot n)$. Hence, compared to the algorithm for GAP LS-WEIGHTED VERTEX COVER, we are able to consider swaps of double the size.

We then consider parameters that are related to modular decompositions. These parameters measure a different structural aspect, the similarity of neighborhoods in the graph, than treewidth or the degree-related parameterizations. In particular, they can be very small in dense graphs. For GAP LS-WEIGHTED VERTEX COVER we develop an FPT-algorithm with running time $\mathcal{O}(\text{mw}(G)^k \cdot k \cdot (\text{mw}(G) + k) \cdot n + m)$ based on dynamic programming on a modular decomposition, where $\text{mw}(G)$ is the modular-width of G , the size of the largest vertex set of any quotient graph of the modular decomposition. We then show an improvement of this algorithm in terms of the structural parameter. More precisely, we show that when processing a node of the decomposition in the dynamic programming algorithm, one may apply a branching algorithm to determine how the swap interacts with the current node. Superficially, this branching algorithm resembles the one for $\Delta(G)$ and k but including the information computed for other nodes of the decomposition requires to combine the branching with KNAPSACK DP-algorithms. This gives an FPT-algorithm for the parameters $\Delta_{\text{md}}(G)$ and k . Recall that $\Delta_{\text{md}}(G)$ is the maximum degree over all quotient graphs of a modular decomposition of minimum width which is upper-bounded by $\text{mw}(G)$. We believe that this novel parameter can be useful in further algorithmic applications of modular decompositions. We remark that the presented algorithm for $\Delta_{\text{md}}(G)$ and k only solves the unweighted gap version of LS-VC; an extension to GAP LS-WEIGHTED VERTEX COVER seems possible but somewhat tedious.

In a second improvement, we show that instead of modular-width one can also obtain FPT-algorithms when using the smaller splitwidth; the results for this parameter are deferred to the full version of this article. Another candidate parameterization would be cliquewidth. We do not consider cliquewidth here, since there is no polynomial-time polynomial-factor approximation of cliquewidth and thus the desired type of FPT running times can only be achieved when a clique decomposition is given as input.

We complement these algorithms by conditional lower bounds that are based on the assumption that matrix-multiplication-based algorithms for the CLIQUE-problem are running-time-optimal [1]. We show that under this assumption, we may not expect a very large improvement of the previous known and new algorithms.

The proofs of statements marked with a (*) are deferred to a full version. For further details regarding parameterized algorithms, refer to the textbook of Cygan et al. [7].

2 Preliminaries

For integers i and j with $i \leq j$, we define $[i, j] := \{k \in \mathbb{N} \mid i \leq k \leq j\}$. For a set A , we denote with $\binom{A}{2} := \{\{a, b\} \mid a \in A, b \in A, a \neq b\}$ the collection of all size-two subsets of A . For two sets A and B , we denote with $A \oplus B := (A \setminus B) \cup (B \setminus A)$ the *symmetric difference* of A and B .

Graph Notation. An (undirected) graph $G = (V, E)$ consists of a set of vertices V and a set of edges $E \subseteq \binom{V}{2}$. For vertex sets $S \subseteq V$ and $T \subseteq V$, we denote with $E_G(S, T) := \{\{s, t\} \in E \mid s \in S, t \in T\}$ the edges between S and T and we use $E_G(S) := E_G(S, S)$ as a shorthand. Moreover, we define $G[S] := (S, E_G(S, S))$ as the *subgraph of G induced by S* . For a vertex $v \in V$, we denote with $N_G(v) := \{w \in V \mid \{v, w\} \in E\}$ the *open neighborhood* of v in G and with $N_G[v] := \{v\} \cup N_G(v)$ the *closed neighborhood* of v in G . Analogously, for a vertex set $S \subseteq V$, we define $N_G[S] := \bigcup_{v \in S} N_G[v]$ and $N_G(S) := \bigcup_{v \in S} N_G(v) \setminus S$. If G is clear from the context, we may omit the subscript.

Modular Decompositions. A *modular decomposition* of a graph $G = (V, E)$ is a pair (\mathcal{T}, β) consisting of a rooted tree $\mathcal{T} = (\mathcal{V}, \mathcal{A}, x^*)$ with root $x^* \in \mathcal{V}$ and a function β that maps each node $x \in \mathcal{V}$ to a graph $\beta(x)$. If x is a leaf of \mathcal{T} , then $\beta(x)$ contains a single vertex of V and for each vertex $v \in V$, there is exactly one leaf ℓ of \mathcal{T} such that the graph $\beta(\ell)$ consists only of v . If x is not a leaf node, then the vertex set of $\beta(x)$ is exactly the set of child nodes of x in \mathcal{T} . Moreover, let V_x denote the set of vertices of V contained in leaf nodes of the subtree rooted in x . Formally, V_x is recursively defined as $V(\beta(\ell))$ for leaf nodes ℓ and defined as $\bigcup_{y \in V(\beta(x))} V_y$ for each non-leaf node x . Moreover, we define $G_x = (V_x, E_x) := G[V_x]$. A modular decomposition has the property that for each non-leaf node x and any pair of distinct nodes $y \in V(\beta(x))$ and $z \in V(\beta(x))$, y and z are adjacent in $\beta(x)$ if there is an edge in G between each pair of vertices of V_y and V_z and y and z are not adjacent if there is no edge in G between any pair of vertices of V_y and V_z . Hence, it is impossible that there are vertex pairs $(v_1, w_1) \in V_y \times V_z$ and $(v_2, w_2) \in V_y \times V_z$ such that v_1 is adjacent with w_1 and v_2 is not adjacent with w_2 .

We call $\beta(x)$ the *quotient graph* of x . A quotient graph is *prime* if there is no set $A \subseteq V(\beta(x))$ with $2 \leq |A| < |V(\beta(x))|$ such that all vertices of A have the same neighborhood in $V(\beta(x)) \setminus A$. The *width of a modular decomposition* is the size of the largest vertex set of any quotient graph and the *modular-width* of a graph G is the minimal width of any modular decomposition of G denoted by $\text{mw}(G)$.

The formal definition of treewidth and tree decompositions is deferred to the appendix.

Vertex Cover Local Search. A vertex set $S \subseteq V$ is a *vertex cover* of G if at least one endpoint of each edge in E is contained in S . Let S be a vertex cover of G . A k -*swap*, for $k \in \mathbb{N}$, is a vertex set W of size at most k and W is said to be *valid for S in G* if $S \oplus W$ is also a vertex cover of G . For each valid swap W for S in G , both $W \cap S$ and $W \setminus S$ are independent sets and $N(W) \setminus S = N(W \cap S) \setminus S \subseteq W$. A swap W is *connected* if $G[W]$ is connected. Let $\omega : V \rightarrow \mathbb{N}$. For some $X \subseteq V$, we set $\omega(X) = \sum_{x \in X} \omega(x)$. The *improvement* of W is defined as $\alpha_\omega^S(W) := \omega(W \cap S) - \omega(W \setminus S)$. Moreover, W is *improving* if $\alpha_\omega^S(W) > 0$ and *d -improving* for some $d \in \mathbb{N}$ if $\alpha_\omega^S(W) \geq d$. If S or ω are clear from the context, we may omit them. In this work, we study the following local search problems for VERTEX COVER.

LS-WEIGHTED VERTEX COVER (LS-WVC)

Input: A graph $G = (V, E)$, a weight function $\omega : V \rightarrow \mathbb{N}$, a vertex cover S of G , and $k \in \mathbb{N}$.

Question: Is there a valid improving k -swap $W \subseteq V$ for S in G ?

GAP LS-WEIGHTED VERTEX COVER (GLS-WVC)

Input: A graph $G = (V, E)$, a weight function $\omega : V \rightarrow \mathbb{N}$, a vertex cover S of G , $k \in \mathbb{N}$, and $d \in \mathbb{N}$.

Question: Is there a valid d -improving k -swap $W \subseteq V$ for S in G ?

Moreover, we define GAP LS-VERTEX COVER (GLS-VC) as the special case of GLS-WVC where $\omega(v) = 1$ for each $v \in V$ and $d \in [1, k]$ and LS-VERTEX COVER (LS-VC) as the special case of GLS-VC, where $d = 1$. Let $I = (G = (V, E), S, \omega, k, d)$ be an instance of GLS-WVC. We say that $W \subseteq V$ is a *solution for I* , if W is a valid d -improving k -swap for S in G .

3 Basic Observations and Lower Bounds

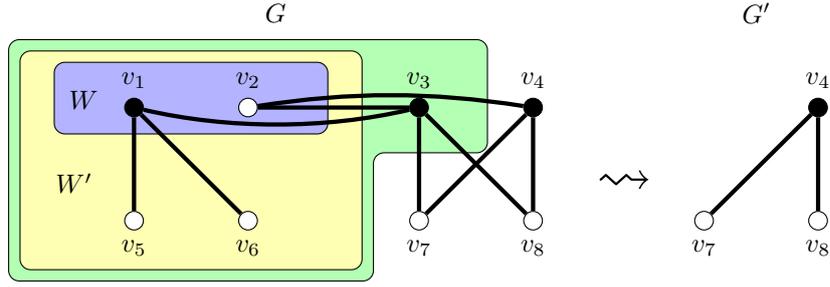
In this section, we first define swap-instances which are instances obtained from applying some partial swap. Swap-instances will be useful for describing certain parts of our algorithms such as branching rules. We then make some observations on certain useful properties of improving swaps. Finally, we present our running time lower bounds for the considered parameters.

Swap-Instances. In our algorithms, we may change instances by performing some partial swaps, for example during branching. We call the instance obtained by such an operation a *swap-instance*. Intuitively, the swap-instance $\text{swap}(I, W)$ for an instance I of GLS-WVC and a (partial) swap W is the GLS-WVC-instance obtained as follows: First, swap W . Then, swap further vertices to make the swap W valid, that is, to maintain that S is a vertex cover. To simplify the instance, the set $W' \supseteq W$ of swapped vertices is then removed from the instance. Finally, to maintain equivalence, the remaining budget k is decreased by the number of swapped vertices and the required improvement d is decreased by the improvement of W' . Formally, this reads as follows.

► **Definition 3.1.** Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS-WVC and let $W \subseteq V$ be a k -swap. Define $W' := W \cup (N(W) \setminus S)$. The instance

$$\text{swap}(I, W) := (G', \omega', S' := S \setminus W, k', d')$$

with $G' := G - (N(W \cap S) \cup W')$, $k' := k - |W'|$, $d' := d - \alpha(W')$, and $\omega'(v) := \omega(v)$ for each $v \in V(G')$ is the swap-instance for I and W .



■ **Figure 1** An instance $I := (G, S, k, d)$ (left) and the swap-instance $\text{swap}(I, W) := (G', S', k', d')$ (right) obtained from the swap $W := \{v_1, v_2\}$. The vertex cover vertices are black, the independent set vertices are white. The green area contains the vertices of $N(W \cap S) \cup W'$ which are in G but not in G' . Since W' has size 4 and contains only one vertex of S , $k' := k - 4$ and $d' := d + 2$. Moreover, the vertex v_3 is not contained in G' since v_1 is adjacent to v_3 and leaves the vertex cover, which implies that v_3 cannot leave the vertex cover afterwards.

An example of a swap-instance can be seen in Figure 1. Note that W' is a subset of each valid swap W^* for S in G where $W \subseteq W^*$.

► **Lemma 3.2** (*). *Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS-WVC and let $W \subseteq V$ such that $W \cap S$ is an independent set. There is a solution W^* for I with $W \subseteq W^*$ if and only if $\text{swap}(I, W)$ is a yes-instance of GLS-WVC.*

If I is an instance of GLS-VC, then $k' + d' = k + d - 2 \cdot |W \cap S|$, since $\alpha(W') = -|W| + 2 \cdot |W \cap S|$. Let I be an instance of GLS-WVC and let W be the subset of some valid swap. When we replace the instance I by $\text{swap}(I, W)$ we may say that we *swap* W in I .

Properties of Improving Swaps. Next, we show that it is sufficient to consider instances of GLS-VC where $k + d$ is even.

► **Lemma 3.3** (*). *Let $I = (G, S, k, d)$ be an instance of GLS-VC where $k + d$ is odd. If I is a yes-instance of GLS-VC, then $I' := (G, S, k - 1, d)$ is a yes-instance of GLS-VC.*

Consider some improving swap W for S in G . Then, each connected component in $G[W]$ is a valid swap and since W is improving, at least one connected component in $G[W]$ is an improving swap for S in G . Hence, the following holds.

► **Observation 3.4.** *Let $I = (G, \omega, S, k)$ be a yes-instance of LS-WVC. There is some valid improving k -swap W for S in G such that W is connected.*

Some of our algorithms branch over all possible intersections of a d -improving k -swap W with a given vertex set X . The following lemma shows that for GLS-VC, we only have to consider intersections of size at most $\frac{k+d}{2}$ of X with potential improving swaps.

► **Lemma 3.5.** *Let $I = (G = (V, E), S, k, d)$ be an instance of GLS-VC, let W be a solution for I , and let $S_X := W \cap X \cap S$ and $C_X := W \cap X \setminus N[S_X]$ for some $X \subseteq V$. If $|S_X \cup C_X| > \frac{k+d}{2}$, then there is a solution W' for I such that W' is a proper subset of W .*

Proof. First, we show that $W^* := S_X \cup (N(S_X) \setminus S)$ is a solution for I . Note that each d -improving k -swap contains at most $\frac{k-d}{2}$ vertices of $V \setminus S$. Since W is valid, it follows that $W^* \subseteq W$ and, thus, $|C_X| + |N(S_X) \setminus S| \leq \frac{k-d}{2}$. Moreover, since $|S_X \cup C_X| > \frac{k+d}{2}$, $|S_X| > d + \frac{k-d}{2} - |C_X| \geq d + |(N(S_X) \setminus S)|$. Hence, W^* is a solution for I .

If W^* is a proper subset of W , then the statement already holds. Hence, assume that $W = W^*$. As a consequence, $C_X = \emptyset$ and S_X has size more than $\frac{k+d}{2}$. Let S'_X be an arbitrary subset of S_X of size $\frac{k+d}{2}$ and let ℓ denote the size of the difference $S_X \setminus S'_X$. We show that $W' := S'_X \cup (N(S'_X) \setminus S)$ is a solution for I . Since S'_X is a subset of both S_X and S , and W^* is a valid swap, W' is a valid swap as well. Moreover, since W^* has size at most k and W' is a subset of W^* , W' is a k -swap. Finally, since $|S_X| = \frac{k+d}{2} + \ell$ and W^* is a k -swap, $W^* \setminus S_X = N(S_X) \setminus S$ has size at most $\frac{k-d}{2} - \ell$. Hence, W' is d -improving since $W' \setminus S'_X = N(S'_X) \setminus S$ is a subset of $N(S_X) \setminus S$ and thus has size at most $\frac{k-d}{2} - \ell$. ◀

To obtain linear FPT running times, we handle instances with small values of k separately.

► **Lemma 3.6 (*)**. *GLS-WVC can be solved in $\mathcal{O}(n+m)$ time if $k \leq 2$ and GLS-VC can be solved in $\mathcal{O}(n+m)$ time if $k+d \leq 4$.*

Lower Bounds. Let $\omega < 2.373$ be the matrix multiplication constant [3]. Using a reduction to matrix multiplication, one can solve the CLIQUE problem, which asks whether an n -vertex graph has a clique of size k , in $\mathcal{O}(n^{\omega \cdot k/3})$ time [23]. It is a long-standing question whether this running time can be improved to $\mathcal{O}(n^{(\omega/3-\varepsilon)k})$ [1, 25]. Assuming that this is not the case, we obtain the following lower bounds for our considered problem.

► **Theorem 3.7**. *For every $\varepsilon > 0$ and every $d \in [1, k]$, GLS-VC cannot be solved in $\mathcal{O}(\ell^{(\omega/3-\varepsilon) \cdot \frac{k+d}{2}})$ time where $\ell = \max\{n - \frac{k-d}{2}, \Delta(G), \text{vc}(G), |S|, \text{mw}(G)\}$, unless CLIQUE can be solved in $\mathcal{O}(n^{(\omega/3-\varepsilon)k})$ time.*

Proof. Let $\varepsilon > 0$ be a constant. We assume in the following that $\varepsilon < \omega/3$, since the statement follows directly for $\varepsilon \geq \omega/3$. Moreover, let $I^* = (G^* = (V, E^*), k)$ be an instance of CLIQUE with $k \geq \frac{2}{(\omega/3)-\varepsilon}$ and, let n denote the size of V , and let d be an arbitrary value between 1 and k . We show that we can compute in $\mathcal{O}(n^2)$ time an equivalent instance $I' = (G' = (V', E'), S, k', d)$ of GLS-VC such that $\ell := \max\{n' - \frac{k'-d}{2}, \Delta(G'), \text{vc}(G'), |S|, \text{mw}(G')\}$ is at most n . First, let $G = (V, E)$ be the complement graph of G^* , that is, $E := \binom{V}{2} \setminus E^*$. Note that a set $X \subseteq V$ is a clique in G^* if and only if X is an independent set in G and that one can compute G in $\mathcal{O}(n^2)$ time. We can assume that the maximum degree of G is at most $|V| - k$, since vertices in G of degree at least $|V| - k + 1$ are contained in no independent set of size k . We obtain G' by adding a set V^* of $k - d$ new vertices to G such that $N_{G'}(v) = V$ for all $v \in V^*$. Finally, we set $k' := 2k - d$ and $S := V$, which completes the construction of I' . Note that this takes at most $\mathcal{O}(n^2)$ time, since $k \leq n$. Next, we show that I^* is a yes-instance of CLIQUE if and only if I' is a yes-instance of GLS-VC.

(\Rightarrow) Let $C \subseteq V$ be an independent set of size k in G , then $S' := (V \setminus C) \cup V^*$ is a vertex cover for G' such that $|S \oplus S'| = k'$ and $|S'| \leq |S| - d$. Consequently, I' is a yes-instance of GLS-VC.

(\Leftarrow) Let $S' \subseteq V'$ such that $|S \oplus S'| \leq k'$ and $|S'| < |S| - d$. Consequently, $C := S \setminus S'$ is non-empty. We show that C is an independent set of size k in G . Since S' is a vertex cover for G' and every vertex of V^* is adjacent to every vertex of V , it follows that $V^* \subseteq S'$. By the fact that $|S \oplus S'| \leq 2k - d$, $S' \setminus S = V^*$, and V^* has size $k - d$, C has size at most k . Moreover, since $|S'| \leq |S| - d$, C has size at least $k' - |V^*| = k$. As a consequence, C has size k . Moreover, since S' is a vertex cover for G' , no two vertices of C are adjacent. Consequently, C is an independent set of size k in G and, thus, I^* is a yes-instance of CLIQUE.

Next, we show that $\ell := \max\{n' - \frac{k'-d}{2}, \Delta(G'), \text{vc}(G'), |S|, \text{mw}(G')\}$ is at most n . By construction, $n' = n + k - d = n + \frac{k'-d}{2}$. Since the maximum degree of G is at most $n - k$, the maximum degree of G' is at most n . Moreover, since S is a vertex cover of size n

for G' , $\text{vc}(G') \leq n$. Next, we show that the modular-width of G' is at most n . Let (\mathcal{T}_1, β_1) be a modular decomposition of G and let (\mathcal{T}_2, β_2) be a modular decomposition of $G'[V' \setminus V]$. Since there is an edge between any pair of vertices of V and $V' \setminus V$, a modular decomposition (\mathcal{T}, β) of G' can be obtained by combining (\mathcal{T}_1, β_1) and (\mathcal{T}_2, β_2) in the following way: We add a new root x^* where $\beta(x^*)$ is a graph consisting of a single edge and the vertices of $\beta(x^*)$ are the roots of the two modular decompositions (\mathcal{T}_1, β_1) and (\mathcal{T}_2, β_2) . Note that $\text{mw}(G) \leq n$. Moreover, since $V' \setminus V$ is an independent set, we have $\text{mw}(G'[V' \setminus V]) = 2$. Hence, $\text{mw}(G') \leq n$.

Now, if we have an algorithm A solving GLS-VC in $\mathcal{O}(\ell^{(\omega/3-\varepsilon) \cdot \frac{k+d}{2}})$ time for $\varepsilon > 0$, then CLIQUE can be solved in $\mathcal{O}(n^{(\omega/3-\varepsilon)k})$ time as well: Since $k \geq \frac{2}{(\omega/3)-\varepsilon}$, the running time $\mathcal{O}(n^{(\omega/3-\varepsilon) \cdot k})$ dominates the time used to construct the instance I' of GLS-VC. Now the running time bound for solving CLIQUE using A follows directly from $\ell \leq n$ and $\frac{k'+d}{2} = k$. ◀

For the cases LS-VC and GLS-WVC, we obtain the following.

► **Corollary 3.8 (*)**. *For every $\varepsilon > 0$, LS-VC cannot be solved in $\mathcal{O}(\ell^{(\omega/3-\varepsilon) \cdot \lceil \frac{k}{2} \rceil})$ time for $\ell := \max\{n - k + 1, \Delta(G), \text{vc}(G), |S|, \text{mw}(G)\}$ and GLS-WVC cannot be solved in $\mathcal{O}(n^{(\omega/3-\varepsilon)k})$ time, unless CLIQUE can be solved in $\mathcal{O}(n^{(\omega/3-\varepsilon)k})$ time.*

4 Parameterization by Treewidth

In this section, we present FPT-algorithms for k and the treewidth of G .

Intuitively, the algorithms are obtained by a dynamic programming algorithm on a given tree decomposition of width r where each entry of the dynamic programming table considers the intersection of the current bag of size $r + 1$ with an improving swap W of size at most k .

► **Theorem 4.1**. *Let $G = (V, E)$ be an undirected graph, let $\omega : V \rightarrow \mathbb{N}$ be a weight function, let $S \subseteq V$ be a vertex cover in G , and let k be a natural number. Given a nice tree decomposition of width r for G with $\mathcal{O}(n)$ bags, one can compute in $\mathcal{O}((r^k + k^2) \cdot n)$ time a valid k -swap W for S in G such that $\alpha(W)$ is maximal under all valid k -swaps for S in G .*

Proof. Due to Lemma 3.6, the statement holds for $k \leq 2$. In the following, we show the running time by describing a dynamic programming algorithm for $k \geq 3$.

Let $N_x(U) := N(U) \cap \beta(x)$ denote the neighbors of U in the bag of $x \in \mathcal{V}$. For a node $x \in \mathcal{V}$, we define with V_x the union of all bags $\beta(y)$, where y is reachable from x in \mathcal{T} . Moreover, we set $G_x := G[V_x]$ and $E_x := E_G(V_x)$. For each node $x \in \mathcal{V}$ in the tree decomposition, the dynamic programming table D_x has entries of type $D_x[S_x, C_x, k']$ with, $S_x \subseteq S \cap \beta(x)$, $C_x \subseteq \beta(x) \setminus (N(S_x) \cup S)$ and $k' \in [0, k]$, such that $|W_x| \leq k'$ where $W_x := S_x \cup C_x \cup (N_x(S_x) \setminus S)$. Hence, $|S_x \cup C_x| \leq k$.

Each entry stores the maximal improvement $\alpha_S(W)$ of a valid k' -swap $W \subseteq V_x$ for $S \cap V_x$ in G_x such that $W \cap S \cap \beta(x) = S_x$ and $W \cap \beta(x) \setminus (N(S_x) \cup S) = C_x$. In other words, W intersects with the vertices of S of the current bag exactly in S_x and W intersects with the vertices of $V \setminus S$ of the current bag (minus the vertices that are contained in each valid swap containing S_x) exactly in C_x .

To ensure that we do not have to evaluate entries where $|S_x \cup C_x| > k$, we define for all $x \in \mathcal{V}$, $S_x \subseteq \beta(x) \cap S$, $C_x \subseteq \beta(x) \setminus S$, and $k' \in [0, k]$, $f_x(S_x, C_x, k') := D_x[S_x, C_x, k']$ if $|S_x \cup C_x| \leq k$ and $f_x(S_x, C_x, k') := -\infty$, otherwise.

For each leaf node ℓ of \mathcal{T} , we fill the table D_ℓ by setting $D_\ell[\emptyset, \emptyset, k'] := 0$ for each $k' \in [0, k]$.

For all non-leaf nodes x of \mathcal{T} , we set $D_x[S_x, C_x, k'] := -\infty$ if

- S_x is not an independent set in G ,
- $|S_x \cup C_x \cup (N_x(S_x) \setminus S)| > k'$, or
- $N(S_x) \cap C_x \neq \emptyset$.

Note that this is correct since in all three cases, there is no swap fulfilling the constraints of the table definition. To compute the remaining entries $D_x[S_x, C_x, k']$, we distinguish between the three types of non-leaf nodes. For each type, we give only an informal proof of the correctness; the formal proof is omitted.

Forget Nodes. Let x be a forget node, let y be the unique child of x in \mathcal{T} , and let v be the unique vertex in $\beta(y) \setminus \beta(x)$. The entries for x can be computed as follows:

$$D_x[S_x, C_x, k'] := \begin{cases} \max(f_y(S_x, C_x, k'), f_y(S_x \cup \{v\}, C_x \setminus N(v), k')) & v \in S \text{ and} \\ \max(f_y(S_x, C_x, k'), f_y(S_x, C_x \cup \{v\}, k')) & v \notin S. \end{cases}$$

Informally, we chose the larger improvement of the best swap containing v and the best swap not containing v . To consider the best swap containing v , we remove v from the corresponding set (S_x or C_x). If v is a vertex of S , we also have to remove the vertices of $N(v) \setminus S$ from C_x , since these vertices are implicitly stored in the corresponding entry of D_y and, by definition, $N(S_y) \cap C_y = \emptyset$.

Introduce Nodes. Let x be an introduce node, let y be the unique child of x in \mathcal{T} , and let v be the unique vertex in $\beta(x) \setminus \beta(y)$. The entries for x can be computed as follows:

$$D_x[S_x, C_x, k'] := \begin{cases} f_y(S_x \setminus \{v\}, C_x \cup C^*, k' - 1) + \omega(v) & v \in W_x \cap S, \\ f_y(S_x, C_x \setminus \{v\}, k' - 1) - \omega(v) & v \in W_x \setminus S, \\ f_y(S_x, C_x, k') & \text{otherwise,} \end{cases}$$

where $W_x := S_x \cup C_x \cup (N_x(S_x) \setminus S)$ and $C^* := (N_x(v) \setminus S) \setminus N(S_x \setminus \{v\})$.

Informally, if v is a vertex of W_x , we have to consider the entry D_y where v is removed from the corresponding set (S_x or C_x) and adding the improvement we obtain from having v in the considered swap (increasing by $\omega(v)$ if $v \in S$ and decreasing by $\omega(v)$ if $v \notin S$). If v is a vertex of S , the vertices of C^* are not stored implicitly in D_y , so we have to consider the entry of D_y where we also explicitly swap C^* . Otherwise, if v is not a vertex of W_x , we consider the entry of D_y with the same subsets S_x and C_x and the same budget k' .

Join Nodes. Let x be a join node, let y and z be the unique children of x in \mathcal{T} . The entries for x can be computed as follows:

$$D[x, S_x, C_x, k'] := \max_{0 \leq k'' \leq k' - |W_x|} D_y[S_x, C_x, k'' + |W_x|] + D_z[S_x, C_x, k' - k''] - \alpha(W_x)$$

where $W_x := S_x \cup C_x \cup (N_x(S_x) \setminus S)$.

Informally, we divide the budget k' into two parts. One for the subset of vertices of W contained in the subtree rooted in y and one for the subset of vertices of W contained in the subtree rooted in z . Note that W_x is contained on both these vertex sets. Hence, we consider all possible ways to divide k' into two parts, such that both entries have at least enough budget to swap all vertices of W_x . Since the improvement of W_x is added twice, we have to remove $\alpha(W_x)$ from the obtained sum.

The maximal improvement of any valid k -swap for S in G can then be found in $D_{x^*}[\emptyset, \emptyset, k]$. Moreover, the corresponding swap can be found via traceback.

It remains to show the running time. Recall that $(\mathcal{T} = (\mathcal{V}, \mathcal{A}, x^*), \beta)$ is a nice tree decomposition of width r for G with $\mathcal{O}(n)$ bags. The number of entries of the table D_x is upper bounded by $k + 1$ times the number of subsets of $\beta(x)$ of size at most k . Since

20:10 Parameterized Local Search for Vertex Cover

for each $x \in \mathcal{V}$, $\beta(x) \leq r + 1$, we have that all dynamic programming tables together contain $\mathcal{O}(\binom{r+1}{\leq k} \cdot k \cdot n)$ entries, where $\binom{r+1}{\leq k}$ denotes the number of different subsets of size at most k of a set of size $r + 1$. By the following claim, we can compute each of them efficiently.

▷ **Claim (*)**. After a preprocessing running in $\mathcal{O}(\binom{r+1}{\leq k} \cdot k^2 + \binom{r+1}{\leq k-1} \cdot r \cdot k + r^2) \cdot n$ time, one can compute each entry of each table D_x in $\mathcal{O}(k)$ time.

Note that there are $\mathcal{O}(\binom{r+1}{\leq k} \cdot k \cdot n)$ entries in total. Hence, the whole algorithm runs in $\mathcal{O}(\binom{r+1}{\leq k} \cdot k^2 \cdot n)$ time which is $\mathcal{O}(r^k \cdot n)$ time if $r \geq 2$ (the proof of this fact is deferred to Appendix B) and in $\mathcal{O}(2^r \cdot k^2 \cdot n) = \mathcal{O}(k^2 \cdot n)$ time, otherwise. ◀

The dynamic programming algorithm deviates from the simple idea mentioned above in the following detail: it considers only a) the intersection W_x^S of $W \cap S$ with the vertices of the current bag and b) the intersection of W with those vertices of $V \setminus S$ in the current bag that are not contained in $N(W_x^S)$. This is more technical but has the following benefit: The intersection of W with $N(S_x) \setminus S$ is stored implicitly which decreases the factor r^k to $r^{\frac{k+d}{2}}$ for GLS-VC due to Lemma 3.5. In particular, for the case of $d = 1$, that is, for LS-VC, this gives a substantial improvement of the exponential part of the running time from r^k to $r^{\frac{k+1}{2}}$.

► **Theorem 4.2 (*)**. Let $I = (G = (V, E), S, k, d)$ be an instance of GLS-VC. Given a nice tree decomposition of width r for G with $\mathcal{O}(n)$ bags. One can solve I in $\mathcal{O}((r^{\frac{k+d}{2}} + k^2) \cdot n)$ time.

Since computing a tree decomposition of minimal width is NP-hard, we cannot directly obtain a running time of $\mathcal{O}((\text{tw}(G)^k + k^2) \cdot n)$ and $\mathcal{O}((\text{tw}(G)^{\frac{k+d}{2}} + k^2) \cdot n)$, respectively. We can, however, compute a nice tree decomposition of width $\text{tw}(G)$ in $\mathcal{O}(n + m)$ time if $\text{tw}(G) \leq 1800$ [4]. Moreover, for each $r \geq 0$, one can compute a nice tree decomposition of G of width $1800 \cdot r^2$ or correctly output that $\text{tw}(G) > r$ in $\mathcal{O}(r^7 \cdot n \cdot \log(n))$ time [11]. Hence, we can compute in $\mathcal{O}(\text{tw}(G)^8 \cdot n \cdot \log(n))$ time [11] a nice tree decomposition of G of width at most $1800 \cdot \text{tw}(G)^2$. If $\text{tw}(G) \geq 1800$, then the width of the latter tree decomposition is smaller than $\text{tw}(G)^3$. Altogether, we obtain the following.

► **Corollary 4.3**. GLS-VC can be solved in $\mathcal{O}((\text{tw}(G)^{\frac{3 \cdot (k+d)}{2}} + k^2) \cdot n \cdot \log(n))$ time and GLS-WVC can be solved in $\mathcal{O}((\text{tw}(G)^{3k} + k^2) \cdot n \cdot \log(n))$ time.

Note that even if a tree decomposition of width $\text{tw}(G)$ is given, one cannot improve much on the running time due to the lower bound of Theorem 3.7, since $\text{tw}(G) \leq \text{vc}(G)$.

5 Degree-Related Parameterizations

In this section, we present FPT-algorithms for the parameters maximum degree $\Delta(G)$ and k and for the h -index of G and k . In contrast to previous work, these FPT-algorithms solve the more general problems with weights and gap-improvements; the algorithms for $\Delta(G)$ will be used as subroutines in the algorithm for the h -index of G .

We start by presenting an algorithm for instances with an h -index of at most 1 which will be used to handle border cases for both parameterizations.

► **Lemma 5.1 (*)**. GLS-WVC can be solved in $\mathcal{O}(k \cdot \log(k) + n)$ time if $h(G) \leq 1$.

Hence, from now on, we assume that $h(G)$ and $\Delta(G)$ are at least 2.

5.1 Parameterizing Unweighted Gap Local Search by Maximum Degree

The main result in this section for GLS-VC is the following.

► **Theorem 5.2 (*)**. *GLS-VC can be solved in $\mathcal{O}(k! \cdot (\Delta - 1)^{\frac{k+d}{2}} \cdot n)$ time.*

The first idea for an algorithm is to slightly adapt the known $\mathcal{O}(2^k \cdot (\Delta - 1)^{\frac{k+1}{2}} \cdot k^2 \cdot n)$ -time algorithm for LS-VC [18] to GLS-VC. This algorithm, however, relies on the fact that for LS-VC, it is sufficient to consider only connected swaps. For d -improving swaps, however, this is not the case: an improvement of at least 2 may be only achievable by swapping vertices that may have an arbitrarily large distance in the graph. Thus, the gap version of the problem becomes considerably harder.

To avoid considering all possible vertex sets of size at most k , we present two branching rules. The first one applies if there is a vertex v in S where $N(v) \subseteq S$ and branches in all possible ways to swap either v or two non-adjacent vertices of $N(v)$. If this rule cannot be applied, then each vertex in S has at least one neighbor in $V \setminus S$ and, thus, there is no valid improving swap of size one.

► **Proposition 5.3 (*)**. *Let $I = (G = (V, E), S, k, d)$ be a yes-instance of GLS-VC and let v be a vertex of S with $N(v) \subseteq S$. There is a solution W for I such that either $v \in W$ or $|W \cap N(v)| \geq 2$.*

Hence, we obtain the following branching rule. Here, we are interested in swapping two independent neighbors of v at a time to obtain a better branching vector than the one, we could obtain by swapping only a single neighbor of v at a time.

► **Branching Rule 5.1**. *Let $I = (G = (V, E), S, k, d)$ be an instance of GLS-VC and let v be a vertex of S with $N(v) \subseteq S$. For each swap $W \in \left(\binom{N(v)}{2} \setminus E\right) \cup \{\{v\}\}$, branch into the case of swapping W .*

As mentioned above, if the branching rule cannot be applied anymore, then each valid improving swap contains at least two vertices. Before applying the second branching rule, we perform the following preprocessing. First, we compute for each $j \in [2, d]$ some minimum valid connected j -improving k -swap W_j for S in G if there is any. Consider some valid minimum solution W for I and let C be a connected component in $G[W]$ with the minimal improvement. Let $\ell := \alpha(C)$ and let $W' = (W \setminus C) \cup W_\ell$. Since W_ℓ contains at most $|C|$ vertices, we have $|W'| \leq k$. The resulting swap W' is a solution for I if $W \cap N[W_\ell] = \emptyset$.

The idea of the branching rule is now the following: either the d -improving swap contains W_ℓ , some neighbor of W_ℓ , or no connected component that is exactly ℓ -improving. First, we present an algorithm to efficiently find the swaps W_j for all $j \in [1, d]$ using the algorithm of Katzmann and Komusiewicz [18] as a subroutine and afterwards, we formally prove the correctness of this branching.

► **Proposition 5.4 (*)**. *Let $I = (G = (V, E), S, k, d)$ be an instance of GLS-VC. For all $j \in [1, d]$, one can find some minimum valid connected j -improving k -swap W_j for S in G with $|W_j \setminus S| \leq (k - d)/2$ or correctly output that no such swap exists in total in $\mathcal{O}(2^k \cdot (\Delta - 1)^{(k+d)/2} \cdot k^3 \cdot n)$ time.*

► **Proposition 5.5**. *Let $I = (G = (V, E), S, k, d)$ be a yes-instance of GLS-VC and, for each $j \in [1, \lfloor \frac{d}{2} \rfloor]$, let W_j denote a minimum valid connected j -improving $(k - d + j)$ -swap for I . There is a solution W for I such that (i) W is connected or (ii) there is some $j \in [1, \lfloor \frac{d}{2} \rfloor]$ such that $W_j \subseteq W$ or $W \cap N(W_j) \neq \emptyset$.*

Proof. Let W be a minimum solution for I . Hence, the improvement of W is exactly d . Suppose that W is not connected and that for each $j \in [1, \lfloor \frac{d}{2} \rfloor]$, $W_j \not\subseteq W$ and $W \cap N(W_j) = \emptyset$, as otherwise the statement already holds. Let C be a connected component in $G[W]$ that minimizes $\alpha(C)$, that is, the connected swap of W with the smallest improvement. Let $\ell := \alpha(C)$. Since W is not connected and has improvement exactly d , $\ell \leq \lfloor \frac{d}{2} \rfloor$. Note that $\ell \geq 1$ as, otherwise, W is not minimum. Moreover, note that $W' := W \setminus C$ is a valid $(d - \ell)$ -improving $(k - |C|)$ -swap for I and $|C| < k$. Since W has size at most k and is d -improving, $|W \setminus S| \leq (k - d)/2$, which implies that $|C| \leq k - d + \ell$. Recall that W_ℓ is some minimum valid connected ℓ -improving $(k - d + \ell)$ -swap for I . Hence, $|C| \geq |W_\ell|$. Recall that by assumption $W_\ell \not\subseteq W$ and $N(W_\ell) \cap W = \emptyset$. Since W is minimum, this implies that $W_\ell \cap W = \emptyset$, as otherwise $W_\ell \cap W$ is a connected component in $G[W]$ such that $0 < \alpha(W_\ell \cap W) < \alpha(C)$. We set $W^* := W' \cup W_\ell$. Note that W^* is a d -improving k -swap for S in G . It remains to show that W^* is valid. Since W' and W_ℓ are both valid, it follows that W^* is valid if $W' \cap N(W_\ell \cap S) \cap S = \emptyset$. By assumption, this is the case. \blacktriangleleft

Hence, we derive the following branching rule.

► **Branching Rule 5.2.** *Let $I = (G = (V, E), S, k, d)$ be an instance of GLS-VC such that there is no connected solution for I . Moreover, for each $j \in [1, \lfloor \frac{d}{2} \rfloor]$, let W_j denote a minimum valid connected j -improving $(k - d + j)$ -swap for S in G . For each swap $W \in \{\{w\} \mid w \in N(W_j), j \in [1, \lfloor \frac{d}{2} \rfloor]\} \cup \{W_j \mid j \in [1, \lfloor \frac{d}{2} \rfloor]\}$, branch into the case of swapping W .*

With these branching rules, we are now able to prove Theorem 5.2. To obtain the stated running time of Theorem 5.2, we apply a branching algorithm using three steps in each node of the branching tree. First, check in $\mathcal{O}(n + m)$ time if there is a vertex $v \in S$ with $N(v) \subseteq S$. If this is the case, apply Branching Rule 5.1. Due to Proposition 5.3, this is correct. If there is no vertex $v \in S$ with $N(v) \subseteq S$, find some minimum valid connected j -improving k -swaps W_j for I where $|W_j \setminus S| \leq (k - d)/2$ (if such a swap exists), for each $j \in [1, d]$. Due to Proposition 5.4, this can be done in $\mathcal{O}(2^k \cdot (\Delta - 1)^{(k+d)/2} \cdot k^3 \cdot n)$ time. If W_d exists, answer yes. Otherwise, apply Branching Rule 5.2.

Note that this is (besides the change from the 2^k factor to a $k!$ factor in the running time) a direct generalization of the previous best algorithm for LS-VC which runs in $\mathcal{O}(2^k \cdot (\Delta - 1)^{k/2} \cdot k \cdot n)$ time [18] to GLS-VC.

5.2 Parameterizing Weighted Gap Local Search by Maximum Degree

► **Proposition 5.6 (*).** *Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS-WVC. One can enumerate all valid connected k -swaps for I in $\mathcal{O}(2^k (\Delta - 1)^k \cdot k^3 \cdot n)$ time.*

Due to Observation 3.4 and Proposition 5.6 we obtain the following.

► **Corollary 5.7.** *LS-WVC can be solved in $\mathcal{O}(2^k \cdot (\Delta - 1)^k \cdot k^3 \cdot n)$ time.*

To solve GLS-WVC, again, we encounter the problem that the sought solution is not necessarily connected. Hence, for GLS-WVC we show a related algorithm to the one we presented for GLS-VC using only one branching rule which is, more or less, an adaptation of Branching Rule 5.2 to the weighted version. Consider a solution W for I . This time, we want to find some valid improving j -swap W_j for S in G for each $j \in [1, k]$ and branch into the cases of either swapping W_j or swapping some neighbor of W_j . Unfortunately, a

result similar to Proposition 5.3 cannot be obtained¹. Hence, in the worst case, each of these branching cases reduces the parameter k only by one which would lead to a running time factor of $(\Delta - 1)^{2 \cdot k}$ instead of $(\Delta - 1)^k$. Our goal is, thus, to reduce the number of cases in which the parameter is only reduced by one. To this end, we analyze the swap W_1 separately. Let $S_1 := \{v \in S \mid N(v) \subseteq S\}$ denote the set of vertices of improving 1-swaps for S in G and let v^* be the unique vertex of W_1 . Since v^* is some vertex in S_1 of highest weight, if $W \cap N[v^*] = W \cap N[W_1] = \emptyset$, then we can replace some distinct vertex w^* of S_1 contained in W by v^* and also obtain a solution for I . Hence, we can then reduce our branching cases for $j \geq 2$ to the ones in which we consider either swapping W_j or some neighbor of W_j which is not contained in S_1 . Since the remaining considered swaps for $j \geq 2$ have size at least 2, only $|N[W_1]| \leq \Delta + 1$ cases remain in which the parameter is only reduced by one. Hence, these ideas lead to the following branching rule.

► **Branching Rule 5.3.** *Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS-WVC such that I has no connected solution. For each $j \in [1, \lfloor \frac{k}{2} \rfloor]$, let W_j denote some valid connected j -swap for I with maximal improvement. Branch into the case of swapping W for each swap $W \in \{\{w\} \mid w \in N(W_1)\} \cup \{W_j \mid j \in [1, \frac{k}{2}]\} \cup \{\{w\} \mid w \in N(W_j) \setminus S_1, j \in [2, \frac{k}{2}]\}$.*

With this branching rule, we are now able to show the following.

► **Theorem 5.8 (*)**. *GLS-WVC can be solved in $\mathcal{O}(k! \cdot (\Delta - 1)^k \cdot n)$ time.*

To obtain this running time, we do the following in each node of the branching tree. First, find for each $j \in [1, k]$ some valid connected j -swap W_j for I that maximizes $\alpha(W_j)$. Due to Proposition 5.6, this can be done in $\mathcal{O}(2^k \cdot (\Delta - 1)^k \cdot k^3 \cdot n)$ time. Now, if $\alpha(W_k) \geq d$, then I is a yes-instance of GLS-WVC. Otherwise, there is no connected solution for I , since W_k has the maximal improvement of all valid connected k -swaps. Compute the set $S_1 := \{v \in S \mid N(v) \subseteq S\}$ of possible improving swaps of size 1 and apply Branching Rule 5.3.

Finally, we show that we can replace $\Delta(G)$ in the above running time by the h -index of G . The idea behind this algorithm is to branch on all possibilities on how a potential improving swap may intersect the set of high-degree vertices. For each of these potential intersections, we compute the corresponding swap instance and solve it with the help of Theorem 5.8 after removing the remaining high-degree vertices. This is correct due to the following lemma.

Since a valid swap W for S in G that avoids a given set V' does not contain any vertex of $S \setminus V'$ adjacent to some vertex of $V' \setminus S$, we define an *exclusion instance* I' for V' and I as the instance of GLS-WVC, where all vertices of $V' \cup N(V' \setminus S)$ are removed from I . Formally, let $I = (G, \omega, S, k, d)$ be an instance of GLS-WVC, then the exclusion instance I' of GLS-WVC for V' and I is defined as $I' := (G', \omega, S', k, d)$, where $G' := G - (V' \cup N(V' \setminus S))$ and $S' := S \cap V(G')$. Due to the above, we obtain the following.

► **Lemma 5.9 (*)**. *Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS-WVC and let $V' \subseteq V$. There is a solution W for I with $W \cap V' = \emptyset$ if and only if the exclusion instance I' of V' and I is a yes-instance of GLS-WVC.*

► **Theorem 5.10 (*)**. *GLS-WVC can be solved in $\mathcal{O}(k! \cdot (h - 1)^k \cdot n)$ time.*

¹ Consider the path (u, v, w) , with $\omega(v) = 3$, and $\omega(u) = \omega(w) = 1$. Let $S = \{u, v\}$, $k = d = 2$. The only 2-improving 2-swap is $\{v, w\}$. Note that this swap avoids the only valid improving 1-swap $\{u\}$ and contains only one neighbor of u .

6 Using Modular Decompositions

Next, we provide FPT-algorithms that use modular decompositions which, roughly speaking, provide a hierarchical view of the different neighborhoods in a graph G .

We now provide a dynamic programming algorithm over the modular decomposition of G . The nodes of the decomposition are processed in a bottom-up manner. The idea is to consider for a node x the possibilities of how a swap may interact with the vertex sets that are represented by the vertices y of $\beta(x)$. We use the fact that any valid swap of G must also correspond in the natural way to a valid swap of $\beta(x)$. More precisely, if some vertex in the set represented by y goes to the independent set, then the vertex cover must include the vertex set represented by z for all neighbors z of y in $\beta(x)$.

► **Theorem 6.1 (*)**. *GLS-WVC can be solved in $\mathcal{O}(\text{mw}(G)^k \cdot k \cdot (\text{mw}(G) + k) \cdot n + m)$ time.*

Proof. Let $I = (G = (V, E), \omega, S, k, d)$ be an instance of GLS-WVC. First, we compute a modular decomposition $(\mathcal{T} = (\mathcal{V}, \mathcal{A}, x^*), \beta)$ of minimal width in $\mathcal{O}(n + m)$ time [22]. Note that \mathcal{T} has $\mathcal{O}(n)$ quotient graphs. Next, we describe a dynamic program on the modular decomposition (\mathcal{T}, β) to solve GLS-WVC.

For each node $x \in \mathcal{V}$ in the modular decomposition, we have a dynamic programming table D_x . The table D_x has entries of type $D_x[k']$ for $k' \in [0, k]$. Each entry $D_x[k']$ stores the maximal improvement $\alpha_S(W)$ of a valid k' -swap $W \subseteq V_x$ for $S \cap V_x$ in G_x .

Next, we describe how to fill the dynamic programming tables. Let ℓ be a leaf node of \mathcal{T} and let v be the unique vertex of $V(\beta(\ell)) = V_\ell$. We fill the table D_ℓ by setting

$$D_\ell[k'] := \begin{cases} 0 & v \notin S \vee k' = 0 \\ \omega(v) & v \in S \wedge k' > 0 \end{cases}$$

for each $k' \in [0, k]$.

To compute the entries for all remaining nodes x of \mathcal{T} , we use an auxiliary table Q_{S_x} . Let S_x be an independent set in $\beta(x)$ and let $S_x(i)$ denote the i th vertex of S_x according to some arbitrary but fixed ordering with $i \in [1, |S_x|]$. Moreover, let $V_x^{\geq i} = \bigcup_{j=i}^{|S_x|} V_{S_x(j)}$. The dynamic programming table $Q_{S_x}[i, k']$ has entries for $i \in [1, |S_x| + 1]$ and $k' \in [0, k]$ and stores the maximal improvement of a valid k' -swap W for $S \cap V_x^{\geq i}$ in $G[V_x^{\geq i}]$, such that there is at least one vertex in $S \cap W \cap V_{S_x(j)}$ for each $j \in [i, |S_x|]$. We set

$$Q_{S_x}[i, k'] := \begin{cases} -\infty & |S_x| - i + 1 > k', \\ 0 & i = |S_x| + 1, \text{ and} \\ \max_{1 \leq k'' \leq k'} D_{S_x(i)}[k''] + Q_{S_x}[i + 1, k' - k''] & \text{otherwise.} \end{cases}$$

The entries for D_x can then be computed as follows:

$$D_x[k'] := \max_{\substack{S_x \subseteq V(\beta(x)) \\ |W^*| \leq k' \\ S_x \text{ is independent}}} Q_{S_x}[1, k' - |W^*|] - \omega(W^*)$$

where $W^* := \bigcup_{y \in N_x(S_x)} (V_y \setminus S)$.

The maximal improvement of any valid k -swap for S in G can be found in $D_{x^*}[k]$, where x^* is the root of the modular decomposition.

Next, we analyze the running time. For each non-leaf node x , and each independent set S_x of size at most k in $\beta(x)$, there are $\mathcal{O}(k^2)$ table entries in Q_{S_x} and each of these entries can be computed in $\mathcal{O}(k)$ time. For a set of size x , let $\binom{x}{\leq k}$ denote the number of different

subsets of size at most k . Since each quotient graph has $\mathcal{O}(\binom{\text{mw}(G)}{\leq k})$ many independent sets of size at most k , all entries of all tables Q_{S_x} can be computed in $\mathcal{O}(\binom{\text{mw}(G)}{\leq k} \cdot k^3 \cdot n)$ time, since the modular decomposition has $\mathcal{O}(n)$ quotient graphs. For each node x , there are $\mathcal{O}(k)$ table entries in D_x . We will show that we can compute each of them in $\mathcal{O}(k^2 \cdot (\text{mw}(G) + k))$ time. To this end, we precompute for each node x the size $|V_x \setminus S|$ and the weight $\omega(V_x \setminus S)$ to compute $|W^*|$ and $\omega(W^*)$ in $\mathcal{O}(k)$ time afterwards. Since for all non-leaf nodes x , $V_x \setminus S = \bigcup_{y \in V(\beta(x))} (V_y \setminus S)$, we can compute $|V_x \setminus S|$ as $\sum_{y \in V(\beta(x))} |V_y \setminus S|$ and $\omega(V_x \setminus S)$ as $\sum_{y \in V(\beta(x))} \omega(V_y \setminus S)$. This can be done in $\mathcal{O}(\text{mw}(G) \cdot n)$ time since the modular decomposition has $\mathcal{O}(n)$ quotient graphs. Hence, for an independent set S_x of size at most k , we can compute $|W^*|$ and $\omega(W^*)$ in $\mathcal{O}(k)$ time. Since we can enumerate all subsets S_x of size at most k of $V(\beta(x))$ in $\mathcal{O}(\binom{\text{mw}(G)}{\leq k})$ time and check in $\mathcal{O}(\text{mw}(G) \cdot k)$ time if S_x is independent in $\beta(x)$, we can compute $D_x[k']$ in $\mathcal{O}(\binom{\text{mw}(G)}{\leq k} \cdot k^2 \cdot (k + \text{mw}(G)))$ time. Consequently, we can compute all entries of the dynamic programming tables in $\mathcal{O}(\binom{\text{mw}(G)}{\leq k} \cdot k^3 \cdot (k + \text{mw}(G)) \cdot n + m)$ time, which is $\mathcal{O}(\text{mw}(G)^k \cdot k \cdot (\text{mw}(G) + k) \cdot n + m)$ time (the proof of this fact is deferred to Appendix B).

Since the value of $Q_{S_x}[1, k']$ is only evaluated one time during the whole computation of this dynamic programming algorithm, we can remove the table Q_{S_x} after evaluating $Q_{S_x}[1, k']$ for each k' . Consequently, this algorithm also only uses polynomial space. ◀

With a slight modification, we can improve the running time for GLS-VC.

► **Corollary 6.2.** *GLS-VC can be solved in $\mathcal{O}(\text{mw}(G)^{\frac{k+d}{2}} \cdot k \cdot (\text{mw}(G) + k) \cdot n + m)$ time.*

We also obtain an FPT-algorithm for GLS-VC for a new parameter that is upper-bounded by the maximum degree $\Delta(G)$ and by the modular-width $\text{mw}(G)$. We call this parameter the *maximum modular degree* and it is defined by taking the maximum degree over all quotient graphs of a modular decomposition of minimum width.

► **Definition 6.3.** *Let $(\mathcal{T} = (\mathcal{V}, \mathcal{A}, x^*), \beta)$ be a modular decomposition of a graph G . Then the maximum modular degree of (\mathcal{T}, β) is $\Delta_{\text{md}}(\mathcal{T}, \beta) := \max_{x \in \mathcal{V}} \Delta(\beta(x))$. Moreover, the maximum modular degree $\Delta_{\text{md}}(G)$ of G is the maximum modular degree of a modular decomposition (\mathcal{T}', β') of G that minimizes $\Delta_{\text{md}}(\mathcal{T}', \beta')$.*

Since $\text{mw}(G)$ is the largest vertex count of any quotient graph, we have $\Delta_{\text{md}}(G) < \text{mw}(G)$. Moreover, the graph $\beta(x)$ is isomorphic to an induced subgraph of G for all x , and thus $\Delta_{\text{md}}(G) \leq \Delta(G)$.

► **Proposition 6.4 (*)**. *Let $(\mathcal{T} = (\mathcal{V}, \mathcal{A}, x^*), \beta)$ be a modular decomposition of a graph G of minimum width where the quotient graph $\beta(x)$ is prime for each $x \in \mathcal{V}$. Then, $\Delta_{\text{md}}(\mathcal{T}, \beta) = \Delta_{\text{md}}(G)$.*

► **Theorem 6.5 (*)**. *GLS-VC can be solved in $(\Delta_{\text{md}}(G)(k^2 + 2))^k \cdot n^{\mathcal{O}(1)}$ time.*

In the algorithm for $\Delta_{\text{md}}(G)$, to avoid considering all k -swaps of a quotient graph $\beta(x)$, we instead adapt the algorithm of Section 5.2 to find suitable swaps of $\beta(x)$. The main difficulty is that we need to consider all possibilities of how many vertices have been swapped in the subtree of $\beta(x)$.

7 Conclusion

We introduced the notion of FPT running times that grow mildly with respect to some parameter ℓ and strongly with respect to another parameter k . Such running times are desirable in the setting where the parameter k is much smaller than ℓ . Parameterized local search is one scenario in which this assumption is certainly true, when k is the operational parameter that bounds the size of the local search neighborhood. We showed that such running times are achievable for one of the most important graph problems in parameterized local search, LS-VERTEX COVER, and different well-known structural parameters taking the place of ℓ .

There are numerous possibilities for future research. First, it seems interesting to study further parameterized local search problems with the aim of achieving FPT-algorithms whose running times grow strongly only with respect to the operational parameter k . This could also be relevant in other scenarios with operational parameters, for example in turbo-charging of greedy algorithms [2, 8, 12]. Second, it is open to improve our running time bounds for LS-VERTEX COVER since our conditional lower bounds are not completely tight. For example, for LS-VERTEX COVER parameterized by k and the h -index it is open whether a running time of $\mathcal{O}(h^{k/2} \cdot n)$ is possible. Third, it would be interesting to explore gap versions of further local search problems, both from a theoretical and a practical perspective. Furthermore, in our study, we did not explicitly consider permissive local search, where one may report better solutions outside of the local neighborhood if they exist [13]. Our positive and negative results also work in this setting, but it would be interesting to identify structural parameters ℓ where permissive local search has an FPT-algorithm with running time $\ell^{g(k)} \cdot n^{\mathcal{O}(1)}$ and strict local search does not.

Finally, it is open to explore the concrete practical potential of our results for VERTEX COVER: Can our theoretical results lead to good implementations of parameterized local search for WEIGHTED VERTEX COVER? Moreover, can the performance of the parameterized local search algorithm for unweighted VERTEX COVER with parameter (Δ, k) [18] be improved by some of the techniques presented in this work?

References

- 1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant's parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2018. doi:10.1137/16M1061771.
- 2 Faisal N. Abu-Khzam, Shaowei Cai, Judith Egan, Peter Shaw, and Kai Wang. Turbo-charging dominating set with an FPT subroutine: Further improvements and experimental analysis. In *Proceedings of the 14th Annual Conference on Theory and Applications of Models of Computation (TAMC '17)*, volume 10185 of *Lecture Notes in Computer Science*, pages 59–70, 2017.
- 3 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA '21)*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.
- 4 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.
- 5 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.
- 6 Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. NuMVC: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research*, 46:687–716, 2013.

- 7 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 8 Alexander Dobler, Manuel Sorge, and Anaïs Villedieu. Turbocharging heuristics for weak coloring numbers. In *Proceedings of the 30th Annual European Symposium on Algorithms (ESA '22)*, volume 244 of *LIPICs*, pages 44:1–44:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 9 David Eppstein and Emma S. Spiro. The h -index of a graph and its application to dynamic subgraph statistics. *Journal of Graph Algorithms and Applications*, 16(2):543–567, 2012.
- 10 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, and Yngve Villanger. Local search: Is brute-force avoidable? *Journal of Computer and System Sciences*, 78(3):707–719, 2012.
- 11 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Transactions on Algorithms*, 14(3):34:1–34:45, 2018. doi:10.1145/3186898.
- 12 Serge Gaspers, Joachim Gudmundsson, Mitchell Jones, Julián Mestre, and Stefan Rümmele. Turbocharging treewidth heuristics. *Algorithmica*, 81(2):439–475, 2019. doi:10.1007/s00453-018-0499-1.
- 13 Serge Gaspers, Eun Jung Kim, Sebastian Ordyniak, Saket Saurabh, and Stefan Szeider. Don't be strict in local search! In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI '12)*. AAAI Press, 2012.
- 14 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.
- 15 Jiong Guo, Danny Hermelin, and Christian Komusiewicz. Local search for string problems: Brute-force is essentially optimal. *Theoretical Computer Science*, 525:30–41, 2014.
- 16 Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004.
- 17 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988.
- 18 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.
- 19 Christian Komusiewicz and Nils Morawietz. Finding 3-swap-optimal independent sets and dominating sets is hard. In *Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science (MFCS '22)*, volume 241 of *LIPICs*, pages 66:1–66:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 20 Ruizhi Li, Shuli Hu, Shaowei Cai, Jian Gao, Yiyuan Wang, and Minghao Yin. NuMWVC: A novel local search for minimum weighted vertex cover problem. *Journal of the Operational Research Society*, 71(9):1498–1509, 2020.
- 21 Dániel Marx. Searching the k -change neighborhood for TSP is $W[1]$ -hard. *Operations Research Letters*, 36(1):31–36, 2008.
- 22 Ross M. McConnell and Jeremy P. Spinrad. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '94)*, pages 536–545. ACM/SIAM, 1994.
- 23 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.
- 24 Stefan Szeider. The parameterized complexity of k -flip local search for SAT and MAX SAT. *Discrete Optimization*, 8(1):139–145, 2011.
- 25 Gerhard J. Woeginger. Space and time complexity of exact algorithms: Some open problems (invited talk). In *Proceedings of the First International Workshop on Parameterized and Exact Computation (IWPEC '04)*, volume 3162 of *Lecture Notes in Computer Science*, pages 281–290. Springer, 2004.

A

 Tree Decompositions and Treewidth

A *tree decomposition* of a graph $G = (V, E)$ is a pair (\mathcal{T}, β) consisting of a rooted tree $\mathcal{T} = (\mathcal{V}, \mathcal{A}, x^*)$ with root $x^* \in \mathcal{V}$ and a function $\beta : \mathcal{V} \rightarrow 2^V$ such that

1. for every vertex $v \in V$, there is at least one $x \in \mathcal{V}$ with $v \in \beta(x)$,
2. for each edge $\{u, v\} \in E$, there is at least one $x \in \mathcal{V}$ such that $u \in \beta(x)$ and $v \in \beta(x)$, and
3. for each vertex $v \in V$, the subgraph $\mathcal{T}[\mathcal{V}_v]$ is connected, where $\mathcal{V}_v := \{x \in \mathcal{V} \mid v \in \beta(x)\}$.

We call $\beta(x)$ the *bag* of x . The *width* of a tree decomposition is the size of the largest bag minus one and the *treewidth* of a graph G , denoted by $\text{tw}(G)$, is the minimal width of any tree decomposition of G .

We consider tree decompositions with specific properties. A node $x \in \mathcal{V}$ is called:

1. a *leaf node* if x has no child nodes in \mathcal{T} ,
2. a *forget node* if x has exactly one child node y in \mathcal{T} and $\beta(y) = \beta(x) \cup \{v\}$ for some $v \in V \setminus \beta(x)$,
3. an *introduce node* if x has exactly one child node y in \mathcal{T} and $\beta(y) = \beta(x) \setminus \{v\}$ for some $v \in V \setminus \beta(y)$, or
4. a *join node* if x has exactly two child nodes y and z in \mathcal{T} and $\beta(x) = \beta(y) = \beta(z)$.

A tree decomposition $(\mathcal{T} = (\mathcal{V}, \mathcal{A}, x^*), \beta)$ is called *nice* if the bag of the root and the bags of all leaf nodes are empty sets and if every node $x \in \mathcal{V}$ is either a leaf node, a forget node, an introduce node, or a join node.

For a node $x \in \mathcal{V}$, we denote with V_x the union of all bags $\beta(y)$, where y is reachable from x in \mathcal{T} . Moreover, we set $G_x := G[V_x]$ and $E_x := E_G(V_x)$.

B

 Bounding the Number of Small Subsets

To obtain small polynomial factors in the running times of our algorithms, we show the following.

► **Lemma B.1** (*). *Let $k \geq 1$ be an integer and let X be an arbitrary set of size $x \geq 3$. Moreover, let $\binom{x}{\leq k}$ denote the number of different subsets of X of size at most k . Then, $\binom{x}{\leq k} \leq 256 \cdot (x-1)^k / k^2$.*

Proof. Note that $\binom{x}{\leq k} = \sum_{r=0}^k \binom{x}{r} \leq 2^x$ and $\binom{x}{\leq k} \leq x^k$. If $k \leq 4$, then $x^k \leq 16 \cdot (x-1)^k$ and since $k^2 \leq 16$, $\binom{x}{\leq k} \leq 256 \cdot (x-1)^k / k^2$. If $k \geq \max\{4, x/2\}$, then $4^k \geq 2^x$ and $2^k \geq k^2$. Hence, if $x \geq 9$, then by the fact that $k \geq x/2$, $\binom{x}{\leq k} \leq 2^x \leq 4^k \leq 8^k / 2^k \leq 8^k / k^2 \leq (x-1)^k / k^2$. If $4 \leq x \leq 8$, then $\binom{x}{\leq k} \leq 2^x \leq 256 \cdot (x-1)^k / k^2$ since $(x-1)^k > k^2$ for all $k \geq 2$, and for $x = 3$, $\binom{x}{\leq k} \leq 2^x = 8 \leq 256 \cdot 2^k / k^2$ for all $k \geq 2$. If $4 < k < x/2$, then $2 \cdot \binom{x}{k} \geq \sum_{r=0}^k \binom{x}{r} = \binom{x}{\leq k}$. Hence $\binom{x}{\leq k} \leq 2 \cdot \binom{x}{k} \leq 2 \cdot x! / (x-k)! \cdot 1/k! \leq 2 \cdot x \cdot (x-1)! / (x-k)! \cdot 1/k^2 \leq 2 \cdot 2(x-1) \cdot (x-1)^{k-1} \cdot 1/k^2 = 4(x-1)^k / k^2 < 256 \cdot (x-1)^k / k^2$. ◀

Parameterized Complexity of a Parallel Machine Scheduling Problem

Maher Malleem  

Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Claire Hanen  

Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Université Paris Nanterre, UPL, 92000 Nanterre, France

Alix Munier-Kordon  

Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Abstract

In this paper we consider the parameterized complexity of two versions of a parallel machine scheduling problem with precedence delays, unit processing times and time windows. In the first version – with exact delays – we assume that the delay between two jobs must be exactly respected, whereas in the second version – with minimum delays – the delay between two jobs is a lower bound on the time between them. Two parameters are considered for this analysis: the pathwidth of the interval graph induced by the time windows and the maximum precedence delay value. We prove that our problems are para-NP-complete with respect to any of the two parameters and fixed-parameter tractable parameterized by the pair of parameters.

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

Keywords and phrases parameterized complexity, scheduling, precedence delays, pathwidth, chains, parallel processors

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.21

1 Introduction

While scheduling jobs with resources and precedence constraints has a wide range of industrial applications, these problems have been proven to be NP-hard in the strong sense even for very simple settings. For example minimizing the makespan of a schedule of jobs on two parallel processors assuming chain-like precedence constraints is already NP-hard [8]. Scheduling problems are usually denoted by the three field denotation $\alpha|\beta|\gamma$ of Graham [11]. Field α describes the machine setting, field β describes the job relations and properties, and field γ defines the optimization criterion - or is a star \star for a decision problem. For example the NP-hard problem described above is denoted by $P2|chains|C_{\max}$.

In this paper we tackle two decision scheduling problems on M parallel processors. We consider a set of n jobs \mathcal{T} . Each job i has a unit processing time and a time interval $[r_i, d_i]$ given for its computation. r_i is the release date of job i , and d_i its deadline. Jobs are linked by precedence constraints defined by an acyclic precedence graph G with non-negative precedence delays $\ell_{i,j}$ on each arc (i, j) of G . Two variants of the precedence delays are considered: exact precedence delays and minimum precedence delays, which will be denoted by $\ell_{i,j}^{ex}$ and $\ell_{i,j}^{min}$ in the standard notations.

A feasible schedule σ defines for each job $i \in \mathcal{T}$ a starting time $\sigma(i) \in [r_i, d_i]$ so that no more than M jobs are scheduled at the same time, and so that for each arc (i, j) of G :

- in case of exact delays: $\sigma(i) + 1 + \ell_{i,j}^{ex} = \sigma(j)$,
- in case of minimum delays: $\sigma(i) + 1 + \ell_{i,j}^{min} \leq \sigma(j)$.



© Maher Malleem, Claire Hanen, and Alix Munier-Kordon;
licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 21; pp. 21:1–21:21

Leibniz International Proceedings in Informatics



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

In particular we will explore the parameterized complexity of these problems with chain-like precedence constraints, denoted by $P|chains(\ell_{i,j}^{ex}), p_j = 1, r_j, d_j|*$ and $P|chains(\ell_{i,j}^{min}), p_j = 1, r_j, d_j|*$.

When restricting all processing times to be equal to one, NP-hardness results typically depend on the precedence relations used in the problem. Ullman [19] first showed that $P|prec, p_j = 1|C_{max}$ is strongly NP-hard. This was later improved by Lenstra et al [14] with a precedence graph of bounded height, then by Garey et al [10] in the case of an opposing forest. When we have release dates Brucker et al [5] showed that $P|intree, p_j = 1, r_j|C_{max}$ is strongly NP-hard with an intree as the precedence graph.

When further restricting the problem to chains with precedence delays, Yu et al [22] showed that $1|chains(\ell_{i,j}^{min}), p_j = 1|C_{max}$ and $1|chains(\ell_{i,j}^{ex}), p_j = 1|C_{max}$ are strongly NP-hard, even when all chains are of length two (i.e. if we only have coupled tasks). The problem $1|chains(\ell_{i,j}^{ex}), p_j = 1|*$ was also proven NP-hard earlier by Orman in [17]. Note that apart from the delays themselves, there is nothing more to restrict in order to go around NP-hardness. Thus within the scope of classical complexity theory it becomes difficult to find the frontier between polynomial-time solvability and NP-hardness when delays are considered.

Parameterized complexity theory gives numerous tools for a refined analysis of such hard scheduling problems. Given a parameter k and denoting n the input size, a problem is called *fixed-parameter tractable* (FPT) with respect to parameter k if it can be solved in time $poly(n) \times f(k)$ with f an arbitrary function [6]. Not only it tells us more about a problem than if we only showed NP-hardness, it allows us to further classify the NP-hard problems depending on whether they are FPT parameterized by k or not.

When the studied problem is believed to not be FPT, the para-NP class is used as a parameterized version of NP: a problem is in para-NP with respect to parameter k if there is a non-deterministic algorithm that solves it in time $poly(n) \times f(k)$ with f an arbitrary function. In order to prove that a problem is para-NP-hard with respect to k , it is enough to prove that the un-parameterized problem is NP-hard for some fixed value of the parameter [9]. The W-hierarchy defined in [7] is an additional widely used tool in parameterized complexity. It is a sequence of intermediate complexity classes between FPT and para-NP, which allows us to further investigate the time complexity of a parameterized problem.

Until now quite few parameterized complexity results have been proved for scheduling problems. Among the first ones related to our problems, Bodlaender proved in [3] that minimizing the makespan of unit processing times jobs on parallel machines is W[2]-Hard considering the number of machines as parameter. When precedence constraints are involved, several authors studied the parameter defined by the width w of the precedence graph. Van Bevern et al. [20] proved that the problem $P2|prec, p_j \in \{1, 2\}|C_{max}$ is W[2]-hard with respect to the width. In the same paper they also proved that if we add the maximum allowed lag of a job with respect to its earliest start time (ignoring resource constraints) to the width as parameter then the problem becomes FPT, even for more complex resource constraints (RCPSP). In [15] it is noted that the parameterized complexity of the problem with three processors, precedence constraints and unit execution times with respect to width w is still open.

Now considering precedence delays, we can first mention the work of Bessy et al [2] on coupled tasks with due dates, a compatibility graph and a common deadline on a single machine. They consider the number of jobs that end before the due date as their parameter. They establish W[1]-hardness for this problem and propose a FPT algorithm when the maximum duration of a job (i.e. the processing time of the two tasks plus their delay) is bounded. Bodlaender et al in [4] studied the two problems we address in this

paper but assuming that each chain C has a time window $[r_C, d_C)$ instead of time windows for individual jobs like in our case. They considered two parameters: the first one is the number of chains and the second one is the thickness, i.e. the maximum number of overlapping chain time windows. On one hand they proved that for exact or minimum delays $1|chains(\ell_{i,j}), p_j = 1, r_C, d_C|*$ is W[1]-hard with any of the two parameters, while the parallel machine variant $P|chains(\ell_{i,j}), p_j = 1, r_C, d_C|*$ is W[2]-hard. On the other hand they proved that these problems are in XP (i.e. deterministic $n^{f(k)}$ time) if the delays are unary encoded.

Considering scheduling problems with job time windows, the pathwidth μ has been considered recently as a parameter by several authors [1, 16, 21]. Parameter μ is the pathwidth of the graph induced by jobs time intervals. It can be easily computed since $\mu + 1$ is the maximum number of overlapping job time windows at any given time. In particular Munier proved in [16] that scheduling unit processing times jobs with time windows and precedence constraints is FPT with parameter μ . Note that van Bevern et al. [20] showed that $P2|prec, p_j \in \{1, 2\}|C_{max}$ is W[2]-hard parameterized by w , while this problem is likely to be FPT parameterized by μ according to Hanen and Munier [12]. With μ allowing such problems to be FPT when other parameters do not, it makes it all the more difficult to find hardness results relative to μ .

The two studied parameters in our paper are pathwidth μ and the maximum precedence delay $\ell_{max} = \max_{(i,j) \text{ arc of } G} \ell_{i,j}$. We first show in Section 2 that both decision problems $P|chains(\ell_{i,j}^{ex}), p_j = 1, r_j, d_j|*$ and $P|chains(\ell_{i,j}^{min}), p_j = 1, r_j, d_j|*$ are para-NP-complete parameterized by either one alone. In the case of pathwidth μ as the parameter we even prove that the single machine variant of both problems are para-NP-complete. Then in section 3 we prove that these problems are FPT parameterized by the couple (μ, ℓ_{max}) .

2 Hardness reductions

In this section we prove para-NP-completeness of our two parallel machine scheduling problems parameterized by pathwidth μ or maximum delay ℓ_{max} alone. We even establish para-NP-completeness of the single-machine variant in the case of parameter μ .

We use the following result from Flum and Grohe's book [9]:

► **Lemma 1** (Flum, Grohe 1998). *If a (nontrivial) problem \mathcal{P} is NP-complete with a fixed value of some parameter k , then the parameterized problem (\mathcal{P}, k) is para-NP-complete.*

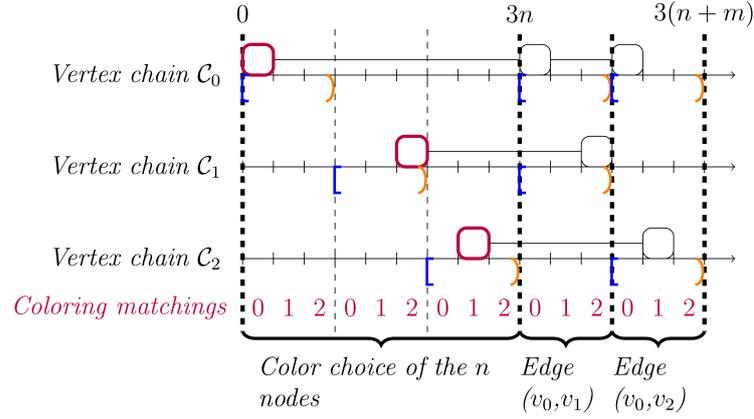
Thus the following scheduling problems will be shown to be NP-complete:

1. $1|chains(\ell_{i,j}^{ex}), p_j = 1, r_j, d_j|*$ with $\mu = 1$,
2. $1|chains(\ell_{i,j}^{min}), p_j = 1, r_j, d_j|*$ with $\mu = 2$,
3. $P|chains(\ell_{i,j}^{ex}), p_j = 1, r_j, d_j|*$ with $\ell_{max} = 1$,
4. $P|chains(\ell_{i,j}^{min}), p_j = 1, r_j, d_j|*$ with $\ell_{max} = 1$.

These problems are all trivially in NP by guessing the starting time of each job then checking if this leads to a feasible schedule. For the hardness proofs all reductions will start from the (strongly) NP-hard 3-COLORING graph problem [13]. Let $G = (V, E)$ be the input graph. Let v_0, \dots, v_{n-1} be the vertices in V and e_0, \dots, e_{m-1} be the edges in E . Let $n = |V|$ and $m = |E|$. The colors will be named 0, 1 and 2.

2.1 NP-hardness of $1|chains(\ell_{i,j}^{ex}), p_j = 1, r_j, d_j|*$ with $\mu = 1$

We build an instance of $1|chains(\ell_{i,j}^{ex}), p_j = 1, r_j, d_j|*$ with $\mu = 1$. An example is given in Figure 1. We have n vertex chains \mathcal{C}_i with $deg(v_i) + 1$ jobs in chain \mathcal{C}_i , $0 \leq i \leq n - 1$ with $deg(v_i)$ the degree of node v_i in G . We define vertex chain \mathcal{C}_i the following way:



■ **Figure 1** An instance of $1|chains(\ell_{i,j}^{e_x}, p_j = 1, r_j, d_j)|\star$ representing a graph coloring. We have $G = (V, E)$ with $V = \{v_0, v_1, v_2\}$ and $E = (\{v_0, v_1\}, \{v_0, v_2\})$. This schedule corresponds to the coloring $(0, 2, 1)$.

▶ **Definition 2** (Vertex chain C_i). We segment time into $m + 1$ segments: a color choice segment $[0, 3n)$ and m edge check selection segments of length 3 along $[3n, 3(n + m))$. We describe the chain from left to right:

(1) Color choice segment $[0, 3n)$

- The first job of chain C_i has time window $[3i, 3(i + 1))$.
 - a. If v_i appears in no edge of G : end the chain.
 - b. Else: set $3(n - i) - 1$ as the current exact delay after this job.

(2) Edge check segment $[3(n + j), 3(n + j + 1))$, $0 \leq j \leq m - 1$

For j in $[0, m - 1]$:

Let edge $e_j = \{v_{i_1}, v_{i_2}\}$, $i_1 < i_2$.

- a. Vertex chain C_i with $i \notin \{i_1, i_2\}$
 - Add 3 to the current exact delay after the currently latest job of chain C_i .
- b. Vertex chain C_i with $i = i_1$ or $i = i_2$
 - Set a job with time window $[3(n + j), 3(n + j + 1))$
 - i. If e_j is the last edge where v_i appears: end the chain.
 - ii. Else: set 2 as the current exact delay after this job.

▶ **Remark 3.** The created instance has pathwidth 1. In the color choice segment: for $i \in [0, n - 1]$ there is exactly one job to be scheduled in time window $[3i, 3(i + 1))$: the first job of vertex chain C_i . In the edge check segments: for $j \in [0, m - 1]$ if edge $e_j = \{v_{i_1}, v_{i_2}\}$ then there are exactly two jobs to be scheduled in time window $[3(n + j), 3(n + j + 1))$: one from vertex chain C_{i_1} and one from vertex chain C_{i_2} . Thus there are indeed at most two overlapping time windows at any given time.

Let $i \in [0, n - 1]$. Vertex chain C_i has three possible starting times in $[3i, 3(i + 1))$ which corresponds to the three color choices of node v_i . Then this color choice is propagated to every edge check segment $[3(n + j), 3(n + j + 1))$ where node v_i is a part of edge e_j , $j \in [0, m - 1]$. The following lemma ensures that the color choices are faithfully propagated.

▶ **Lemma 4.** Let $0 \leq i \leq n - 1$. In any feasible schedule, if vertex chain C_i starts at time $3i + k$ with $k \in \{0, 1, 2\}$, then all jobs J in this chain are scheduled at time $r(J) + k$, where $r(J)$ is the release date of job J .

Proof. Suppose we have a feasible schedule where vertex chain \mathcal{C}_i starts at time $3i + k$ with $k \in \{0, 1, 2\}$.

- If vertex v_i is part of no edge in the graph:
Then by Definition 2 vertex chain \mathcal{C}_i only has one job. It is scheduled at time $3i + k$, which is indeed the release date of this job plus k .
- If vertex v_i is part of at least one edge in the graph:
Let $j_0 < j_1 < \dots < j_{deg(v_i)-1}$ be the indices of the edges e_j such that $v_i \in e_j$. We prove by induction on $l \in [0, deg(v_i) - 1]$ that the job of vertex chain \mathcal{C}_i which has time window $[3(n + j_l), 3(n + j_l + 1))$ is scheduled at time $3(n + j_l) + k$.
 - Consider the job of vertex chain \mathcal{C}_i which has time window $[3(n + j_0), 3(n + j_0 + 1))$. By Definition 2 this job is the successor of the first job in the chain and the exact delay between them is $3(n - i) - 1 + 3j_0$. Thus if the first job of the chain is scheduled at time $3i + k$, then this following job is scheduled at time $(3i + k) + 1 + (3(n - i) - 1 + 3j_0) = 3(n + j_0) + k$, which is indeed the release date of this job plus k .
 - Let $l \in [1, deg(v_i) - 1]$. Suppose the job of vertex chain \mathcal{C}_i which has time window $[3(n + j_{l-1}), 3(n + j_{l-1} + 1))$ is scheduled at time $3(n + j_{l-1}) + k$. Then by Definition 2 there is an exact delay $2 + 3(j_l - j_{l-1} - 1)$ before the next job of the chain. Thus the next job of the chain is scheduled at time $(3(n + j_{l-1}) + k) + 1 + (2 + 3(j_l - j_{l-1} - 1)) = 3(n + j_l) + k$, which is indeed the release date of this job plus k .

This proves the lemma for all the jobs of vertex chain \mathcal{C}_i in an edge check segment. ◀

Then for each edge $e_j = \{v_{i_1}, v_{i_2}\}, i_1 < i_2$, the color choices of v_{i_1} and v_{i_2} are confronted in edge check segment $[3(n + j), 3(n + j + 1))$. If both nodes chose the same color then both jobs in this edge check segment would be scheduled at the same time, which would invalidate our schedule in this single-machine instance. Conversely if we start from a valid coloring, then there will never be two jobs scheduled at the same time in an edge check segment. This is the key ingredient behind the reduction.

► **Proposition 5.** *G is 3-colorable if and only if there exists a feasible schedule for this instance of EXACT DELAYS.*

Proof. (\implies) Suppose we have $(c_0, \dots, c_{n-1}) \in \{0, 1, 2\}^n$ a 3-coloring of G where vertex v_i has color c_i . We propose the schedule where for all $0 \leq i \leq n - 1$, chain \mathcal{C}_i starts at time $3i + c_i$. Then in every edge $e_j \in E$ where vertex v_i appears, we schedule the job of chain \mathcal{C}_i which is in edge check segment $[3(n + j), 3(n + j + 1))$ at time $3(n + j) + c_i$.

We show that the jobs in different chains do not interfere with each other. Since the time windows do not overlap in the color choice segment, only the edge check segments remain to be checked. Let $e_j = \{v_{i_1}, v_{i_2}\}$ be an edge in E . By definition of the vertex chains, only chains \mathcal{C}_{i_1} and \mathcal{C}_{i_2} have a job to be scheduled in time window $[3(n + j), 3(n + j + 1))$. In our schedule the job of chain \mathcal{C}_{i_1} is scheduled at time $3(n + j) + c_{i_1}$ and the job of chain \mathcal{C}_{i_2} at time $3(n + j) + c_{i_2}$. Since (c_0, \dots, c_{n-1}) is a 3-coloring and $\{v_{i_1}, v_{i_2}\} \in E$, we have $c_{i_1} \neq c_{i_2}$. Thus both jobs are scheduled at different times and the jobs in edge check segment $[3(n + j), 3(n + j + 1))$ do not interfere with each other. Thus the proposed schedule is feasible.

(\impliedby) Suppose we have a feasible schedule. For all $0 \leq i \leq n - 1$, let $s_i \in \{0, 1, 2\}$ be such that $3i + s_i$ is the starting time of chain \mathcal{C}_i . We show that (s_0, \dots, s_{n-1}) is a 3-coloring of G . By contradiction suppose there is an edge $e_j = \{v_{i_1}, v_{i_2}\} \in E$ such that $s_{i_1} = s_{i_2}$. Then by Lemma 4 the jobs of chains i_1 and i_2 that must be scheduled in edge check segment $[3(n + j), 3(n + j + 1))$ are scheduled at the same time $3(n + j) + s_{i_1}$. Thus the schedule is not feasible, which leads to a contradiction. Thus (s_0, \dots, s_{n-1}) is indeed a 3-coloring of G . ◀

This proves that $1|chains(\ell_{i,j}^{ex}), p_j = 1, r_j, d_j|*$ with $\mu = 1$ is NP-hard, which concludes the para-NP-completeness proof of the corresponding parameterized problem.

► **Theorem 6.** $1|chains(\ell_{i,j}^{ex}), p_j = 1, r_j, d_j|*$ is para-NP-complete parameterized by pathwidth μ .

2.2 NP-hardness of $1|chains(\ell_{i,j}^{min}), p_j = 1, r_j, d_j|*$ with $\mu = 2$

We build an instance of $1|chains(\ell_{i,j}^{min}), p_j = 1, r_j, d_j|*$ with $\mu = 1$. We begin in a similar way: for each node we have a vertex chain C'_i with three possible starting times, each corresponding to a color choice, then we want to propagate this color choice. However now that the delays are not exact anymore, the color choice cannot be propagated properly as previously. More constraints are needed in order to deal with the extra flexibility coming from the minimum delays.

One way is to add a closing segment $[3(n+m), 3(2n+m))$ at the end and two gadget chains $C'_{i,1}, C'_{i,2}$ per node, each composed of two jobs. As shown in Figure 2 the gadget chains will fill the two gaps at the start and at the end of each vertex chain.

► **Definition 7** (Vertex chain C'_i). *We segment time into $m+2$ segments: a color selection segment $[0, 3n)$, m edge check selection segments along $[3n, 3(n+m))$ and a closing segment $[3(n+m), 3(2n+m))$. We describe the chain from left to right:*

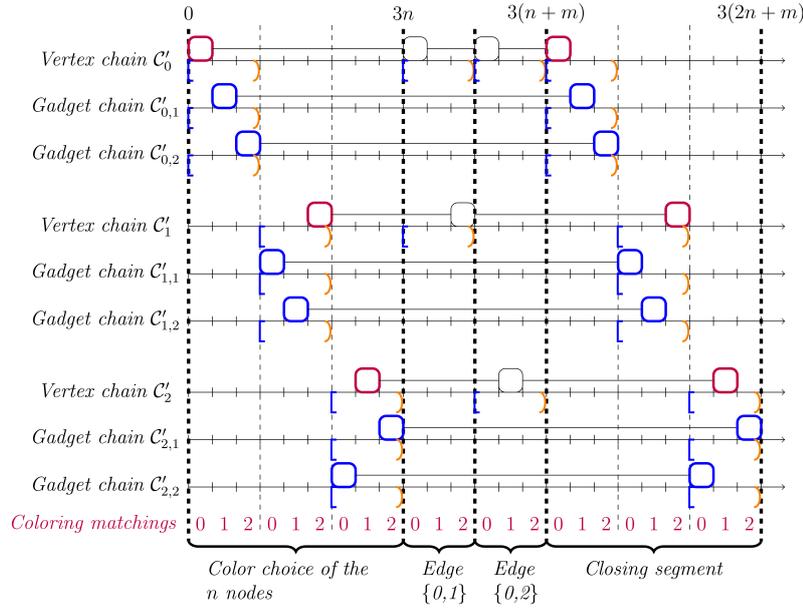
- (1) *Color selection segment $[0, 3n)$*
 - *The first job of chain C'_i has time window $[3i, 3(i+1))$.*
 - *Set $3(n-i) - 1$ as the current minimum delay after this job.*
- (2) *Edge check segment $[3(n+j), 3(n+j+1))$, $0 \leq j \leq m-1$*
For j in $[0, m-1]$:
Let edge $e_j = \{v_{i_1}, v_{i_2}\}$, $i_1 < i_2$.
 - a. *Vertex chain C'_i with $i \notin \{i_1, i_2\}$*
 - *Add 3 to the current minimum delay after the currently latest job of chain C'_i*
 - b. *Vertex chain C'_i with $i = i_1$ or $i = i_2$*
 - *Set a job with time window $[3(n+j), 3(n+j+1))$*
 - *Set 2 as the current minimum delay after this job*
- (3) *Closing segment $[3(n+m), 3(2n+m))$*
 - *Add $3i$ to the current minimum delay of the currently latest job of chain C'_i .*
 - *Set a job with time window $[3(n+i+m), 3(n+i+1+m))$ as the last job of vertex chain C'_i .*

► **Definition 8** (Gadget chains $C'_{i,1}, C'_{i,2}$). *For both gadget chains $C'_{i,1}, C'_{i,2}$ relative to vertex v_i , the first job must be scheduled in time window $[3i, 3(i+1))$, the second one in time window $[3(n+m+i), 3(n+m+i+1))$, and there is a minimum delay $3(n+m) - 1$ between them.*

► **Remark 9.** The created instance has indeed pathwidth 2: gadget chains add two more time windows at the beginning and the end of each vertex chain, so there are at most three time windows overlapping at any time.

A full example is given in Figure 2. For our proof the goal is to show that adding these gadget chains is enough to get an analogue result to Lemma 4. Note that if we only use the definition of the chains like in the reduction of Section 2.1, we only get this weaker result:

► **Lemma 10.** *Let $0 \leq i \leq n-1$. In any feasible schedule, if a chain starts at time $3i+k$ with $k \in \{0, 1, 2\}$, then all jobs J in this chain are scheduled at time $r(J) + k$ or later, where $r(J)$ is the release date of job J .*



■ **Figure 2** An instance of $1|chains(\ell_{i,j}^{min}), p_j = 1, r_j, d_j|*$ representing a graph coloring. We have $G = (V, E)$ with $V = \{v_0, v_1, v_2\}$ and $E = (\{v_0, v_1\}, \{v_0, v_2\})$. This schedule corresponds to the coloring $(0, 2, 1)$.

Proof. For gadget chains $C'_{i,1}, C'_{i,2}$ this comes from the minimum delay $3(n+m) - 1$ between their two jobs. For vertex chain C'_i : suppose we have a feasible schedule where vertex chain C'_i starts at time $3i + k$ with $k \in \{0, 1, 2\}$.

- If vertex v_i is part of no edge in the graph:
Then by Definition 7 vertex chain C_i only has two jobs and there is a minimum delay $3(n-i) - 1 + 3m + 3i = 3(n+m) - 1$ between them. Thus if the first job is scheduled at time $3i + k$, then the job in the closing segment is scheduled at time $(3i + k) + 1 + (3(n+m) - 1) = 3(n+m+i) + k$, which is indeed the release date of this job plus k .
- If vertex v_i is part of at least one edge in the graph:
Let $j_0 < j_1 < \dots < j_{deg(v_i)-1}$ be the indices of the edges e_j such that $v_i \in e_j$. We prove the lemma for all the jobs of vertex chain C_i in an edge check segment by induction on $l \in [0, deg(v_i) - 1]$ the same way as in Lemma 4. Then only the job of the chain in the closing segment is left. By Definition 7 there is a minimum delay $2 + 3(m - 1 - j_{deg(v_i)-1}) + 3i$ between this job and the one in edge check segment $[3(n + j_{deg(v_i)-1}), 3(n + j_{deg(v_i)-1} + 1))$. Since we know from the induction that the latter job is scheduled at time $3(n + j_{deg(v_i)-1}) + k$ or later, this means that the job of vertex chain C_i in the closing segment is scheduled at time $(3(n + j_{deg(v_i)-1}) + k) + 1 + (2 + 3(m - 1 - j_{deg(v_i)-1}) + 3i) = 3(n + m + i) + k$ or later, which is indeed the release date of this job plus k . ◀

By taking into account the constraints added by the gadget chains at the beginning and at the end of each vertex chain, we are able to prove the needed key property.

► **Lemma 11.** *In any feasible schedule, if a vertex chain C_i starts at time $3i + k$ with $k \in \{0, 1, 2\}$, then all jobs J in vertex chain C_i which are in an edge check segment have to be scheduled at time $r(J) + k$, where $r(J)$ is the release date of job J .*

Proof. Consider a feasible schedule. Let $0 \leq i \leq n - 1$. Three jobs are scheduled in time window $[3i, 3(i + 1))$: the first job of vertex chain \mathcal{C}_i and the first job of the two gadget chains relative to it. Let $3i + k$ (resp. $3i + k'_1, 3i + k'_2$) be the starting time of the first job of vertex chain \mathcal{C}_i (resp. gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$). We have k, k'_1 and k'_2 in $\{0, 1, 2\}$ and since we have a feasible schedule the three values are different from each other. Thus one chain starts at time $3i + 2$. By Lemma 10 and the time window $[3(n + m + i), 3(n + m + i + 1))$ of the last job, this means that this last job must be exactly scheduled at time $3(n + m + i) + 2$. Now consider the chain which starts at time $3i + 1$. By Lemma 10 its last job must be scheduled at time $3(n + m + i) + 1$ or $3(n + m + i) + 2$. However the later time position is already taken by the chain starting at time $3i + 2$, which means that this last job must be exactly scheduled at time $3(n + m + i) + 1$. Finally by the same reasoning we get that the chain starting at time $3i$ must have its last job scheduled at time $3(n + m + i)$.

This means that whatever the starting time $3i + k$ is for vertex chain \mathcal{C}_i , its last job must be scheduled at time $3(n + m + i) + k$. So all the delays in the chain must be equal to their minimum and thus by Lemma 10 all the jobs J in the vertex chain must be scheduled at time $r(J) + k$. ◀

Now in any feasible schedule we proved that we have the same guarantee on the position of the edge check jobs as we had with Lemma 4 in the exact delay case. Thus we are able to propagate the color choices accurately and complete the reduction the same way.

► **Proposition 12.** *G is 3-colorable if and only if there exists a feasible schedule for this instance of MIN DELAYS.*

Proof. (\implies) Suppose we have $(c_0, \dots, c_{n-1}) \in \{0, 1, 2\}^n$ a 3-coloring of G where vertex v_i has color c_i . We propose a schedule where for all $0 \leq i \leq n - 1$, vertex chain \mathcal{C}'_i starts at time $3i + c_i$ and gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$ start in the two remaining time positions $3i + k_1, 3i + k_2$ in $[3i, 3(i + 1))$ (with $k_1 \neq k_2$). Plus we require all delays to match their minimum value.

Then, according to Definition 7 and going from left to right as we did in the proof of Lemma 10, we know that in every edge $e_j \in E$ where node v_i appears, the job of vertex chain \mathcal{C}'_i which is in edge check segment $[3(n + j), 3(n + j + 1))$ is scheduled at time $3(n + j) + c_i$, and the last job of \mathcal{C}'_i is scheduled at time $3(n + m + i) + c_i$. Plus from Definition 8 we know that the last job of gadget chain $\mathcal{C}'_{i,1}$ (resp. $\mathcal{C}'_{i,2}$) is scheduled $3(n + m + i) + k_1$ (resp. $3(n + m + i) + k_2$).

We show that the jobs in different chains do not interfere with each other. For the color choice segment we know that vertex chain \mathcal{C}'_i and gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$ start respectively at times $3i + c_i, 3i + k_1$, and $3i + k_2$ with c_i, k_1 and k_2 in $\{0, 1, 2\}$ and different from each other. For the closing segment we determined that vertex chain \mathcal{C}'_i and gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$ end respectively at times $3(n + m + i) + c_i, 3(n + m + i) + k_1$, and $3(n + m + i) + k_2$, again with c_i, k_1 and k_2 in $\{0, 1, 2\}$ and different from each other. Thus only the edge check segments remain to be checked. Let $e_j = \{v_{i_1}, v_{i_2}\}$ be an edge in E . By definition of the vertex chains, only vertex chains \mathcal{C}'_{i_1} and \mathcal{C}'_{i_2} have a job to be scheduled in time window $[3(n + j), 3(n + j + 1))$. In our schedule the job of chain \mathcal{C}'_{i_1} is scheduled at time $3(n + j) + c_{i_1}$ and the job of chain \mathcal{C}'_{i_2} at time $3(n + j) + c_{i_2}$. Since (c_0, \dots, c_{n-1}) is a 3-coloring and $\{v_{i_1}, v_{i_2}\} \in E$, we have $c_{i_1} \neq c_{i_2}$. Thus both jobs are scheduled at different times and the jobs in edge check segment $[3(n + j), 3(n + j + 1))$ do not interfere with each other. Thus the proposed schedule is feasible.

(\impliedby) Suppose we have a feasible schedule. We reuse the same coloring as in the proof of Proposition 5: for all $0 \leq i \leq n - 1$, let $s_i \in \{0, 1, 2\}$ be such that $3i + s_i$ is the starting time of chain \mathcal{C}'_i . We show that (s_0, \dots, s_{n-1}) is a 3-coloring of G . Considering any edge

$e_j = \{v_{i_1}, v_{i_2}\} \in E$, Lemma 11 ensures that the job of vertex chain \mathcal{C}_{i_1} (resp. \mathcal{C}_{i_2}) in edge check segment $[3(n+j), 3(n+j+1))$ is scheduled at time $3(n+j) + s_{i_1}$ (resp. $3(n+j) + s_{i_2}$), with s_{i_1} (resp. s_{i_2}) the starting time of vertex chain \mathcal{C}_{i_1} (resp. \mathcal{C}_{i_2}). Since this is a feasible schedule we have: $3(n+j) + s_{i_1} \neq 3(n+j) + s_{i_2}$, which means: $s_{i_1} \neq s_{i_2}$. Thus the two nodes of edge e_j have indeed different colors. \blacktriangleleft

This proves that $1|chains(\ell_{i,j}^{min}), p_j = 1, r_j, d_j| \star$ with $\mu = 2$ is NP-hard, which concludes the para-NP-completeness proof of the corresponding parameterized problem.

► **Theorem 13.** $1|chains(\ell_{i,j}^{min}), p_j = 1, r_j, d_j| \star$ is para-NP-complete parameterized by path-width μ .

2.3 NP-hardness of $P|chains(\ell_{i,j}^{ex}), p_j = 1, r_j, d_j| \star$ with $\ell_{max} = 1$

We build an instance of $P|chains(\ell_{i,j}^{ex}), p_j = 1, r_j, d_j| \star$ with $\ell_{max} = 1$. We still have n vertex chains, one per node v_i in graph G , and we want to represent and check a coloring of G in a similar way to Section 2.1: choose the color of the nodes with the starting time of the vertex chains, then propagate these choices and check that two nodes of an edge do not have the same color. However with the change of parameter we must worry about the maximum delay value instead of the overlapping of time windows. Here we want to propagate the color choices while keeping the delays small.

We propose to add extra intermediate jobs that we call *propagators* at every other time position. This way the color choices can be propagated along the odd time positions with exact delays of length 1 while the even time positions are kept for edge checking. We set $M = n$ as the number of machines in order to make room for these propagators. We define vertex chain \mathcal{C}_i the following way:

► **Definition 14** (Vertex chain \mathcal{C}_i with $\ell_{max} = 1$). We define \mathcal{C}_i as a chain of $3(n+m-i) + deg(v_i)$ jobs. These jobs will fulfill two roles:

- Propagators $O_{j,k}^i$: $O_{j,0}^i$ will give the color choice of node v_i . The other $3(n+m-i) - 1$ jobs $O_{j,k}^i$ ($i \leq j \leq n+m-1$, $0 \leq k \leq 2$) will propagate this color choice along the whole chain while keeping the maximum delay value at 1. Job $O_{j,k}^i$ will have time window $[6j+2k, 6(j+1)+2k)$.
- Edge jobs J_j^i : the $deg(v_i)$ jobs J_{n+j}^i will represent the color choice of node v_i in every edge e_j where node v_i is in ($0 \leq j \leq m-1$). Job J_{n+j}^i will have time window $[6(n+j), 6(n+j+1))$.

We segment time into $m+1$ segments: a color choice segment $[0, 6n)$ and m edge check segments along $[6n, 6(n+m))$. We describe the chain from left to right:

(1) Color choice segment $[0, 6n)$

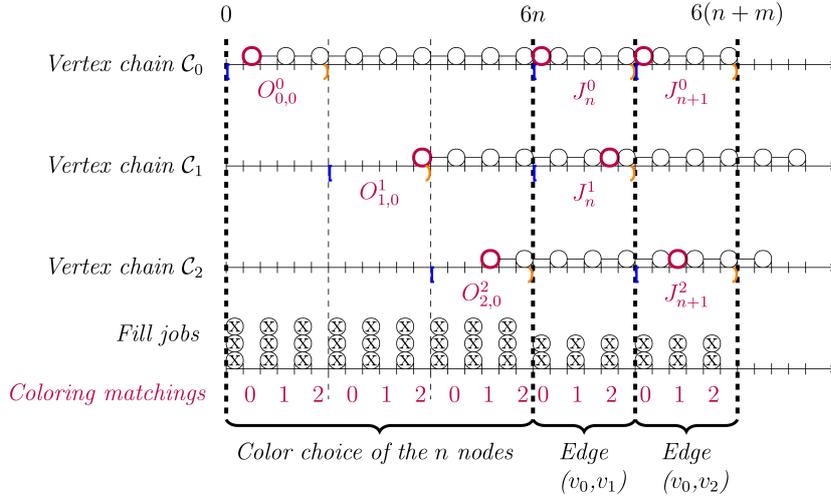
- Set the first job $O_{i,0}^i$ of \mathcal{C}_i in time window $[6i, 6(i+1))$.
- Add a unit-time exact delay then a job, and do this $3(n-i) - 1$ times in a row. These jobs are named $O_{i,1}^i, O_{i,2}^i, O_{i+1,0}^i, \dots, O_{n,2}^i$.

(2) Edge check segment $[6(n+j), 6(n+j+1))$, $0 \leq j \leq m-1$

Let edge $e_j = \{v_{i_1}, v_{i_2}\}$, $i_1 < i_2$. This segment will check if the vertices v_{i_1} and v_{i_2} have different colors.

- a. Vertex chain \mathcal{C}_i with $i \notin \{i_1, i_2\}$: add a unit-time exact delay then job $O_{n+j,0}^i$ then a unit-time exact delay then job $O_{n+j,1}^i$ then a unit-time exact delay then job $O_{n+j,2}^i$.
- b. Vertex chain \mathcal{C}_i with $i = i_1$ or $i = i_2$: add job J_{n+j}^i then an exact delay of length zero then job $O_{n+j,0}^i$ then a unit-time exact delay then job $O_{n+j,1}^i$ then a unit-time exact delay then job $O_{n+j,2}^i$.

21:10 Parameterized Complexity of a Parallel Machine Scheduling Problem



■ **Figure 3** An instance of $P|chains(\ell_{i,j}^{ex}), p_j = 1, r_j, d_j|*$ representing a graph coloring. We have $G = (V, E)$ with $V = \{v_0, v_1, v_2\}$ and $E = (\{v_0, v_1\}, \{v_0, v_2\})$. There are $M = n = 3$ machines and this schedule corresponds to the coloring $(0, 2, 1)$.

Note that time intervals of length 6 are used instead of length 3. This way three odd starting times are available for each vertex chain. However *fill jobs* must be added at every even time position of the color choice segment: M of them so that only these odd starting times are actually allowed. Plus now that we are in a parallel-machine framework instead of a single-machine one, more *fill jobs* are needed at the even time positions of the edge check segments: $M - 1$ of them so that one edge job at any of these positions is allowed but two would invalidate the schedule.

► **Definition 15** (Fill jobs). *Fill jobs are chains of one job with a time window of length 1. M fill jobs are set at every even time position in color choice segment $[0, 6n)$ and $M - 1$ fill jobs are set at every even time position in time segment $[6n, 6(n + m))$.*

An example is given in Figure 3. With the addition of fill jobs the reduction can now be proved correct in a similar way to Section 2.1: determine the exact positions of all jobs in a vertex chain given its starting time, then show that a schedule is feasible if and only if whenever two vertex chains have an edge job in the same edge check segment they must start at different times.

► **Proposition 16.** *G is 3-colorable if and only if there exists a feasible schedule for this instance of $P|chains(\ell_{i,j}^{ex}), p_j = 1, r_j, d_j|*$.*

First we determine the positions of all jobs in a vertex chain given its starting time. In particular we confirm that in any feasible schedule propagators are always scheduled at odd time positions and edge jobs at even time positions:

► **Lemma 17.** *Let $0 \leq i \leq n - 1$. In any feasible schedule, if vertex chain C_i starts at time $6i + 2l + 1$ with $l \in \{0, 1, 2\}$, then all the jobs $O_{j,k}^i$ (resp. J_j^i) in this chain are scheduled at time $r(O_{j,k}^i) + 2l + 1$ (resp. $r(J_j^i) + 2l$), where $r(J)$ is the release date of job J .*

Proof. Suppose we have a feasible schedule where vertex chain \mathcal{C}_i starts at time $6i + 2l + 1$ with $l \in \{0, 1, 2\}$.

- Propagators $O_{j,k}^i$: by Definition 14 there is always either a unit-time exact delay or a job J_j^i between two consecutive jobs $O_{j,k}^i$ in vertex chain \mathcal{C}_i . Thus if the first job $O_{i,0}^i$ is scheduled at time $6i + 2l + 1 = r(O_{i,0}^i) + 2l + 1$, then we know that the next propagator $O_{i,1}^i$ is scheduled at time $(6i + 2l + 1) + 2 = r(O_{i,1}^i) + 2l + 1$, and so on. By induction on the couple (j, k) with $i \leq j \leq n + m - 1$ and $0 \leq k \leq 2$, we get that all the jobs $O_{j,k}^i$ in vertex chain \mathcal{C}_i are scheduled at time $[6i + 2l + 1] + 2 \times (3(j - i) + k) = 6j + 2k + 2l + 1 = r(O_{j,k}^i) + 2l + 1$.
- Edge jobs J_j^i : let $j_0 < j_1 < \dots < j_{deg(v_i)-1}$ be the indices of the edges e_j such that $v_i \in e_j$ (if there are any). Let $p \in [0..deg(v_i) - 1]$. According to Definition 14 job $O_{j_p,0}^i$ has the same time window $[6(n + j_p), 6(n + j_p + 1))$ as job $J_{j_p}^i$ and it is scheduled right before it. Therefore according to our previous point about propagators $O_{j,k}^i$, job $J_{j_p}^i$ is scheduled at time $[r(O_{j_p,0}^i) + 2l + 1] - 1 = r(J_{j_p}^i) + 2l$. ◀

Now we are able to prove Proposition 16:

Proof. (\implies) Suppose we have $(c_0, \dots, c_{n-1}) \in \{0, 1, 2\}^n$ a 3-coloring of G where vertex v_i has color c_i . We propose the schedule σ where for all $0 \leq i \leq n - 1$, chain \mathcal{C}_i starts at time $6i + 2c_i + 1$. Then by Definition 14 and abiding by exact delays, in every edge $e_j \in E$ where vertex v_i is in, job J_j^i is scheduled at time $r(J_j^i) + 2c_i = 6(n + j) + 2c_i$.

We show that there are never more than $M = n$ jobs scheduled at any time position. Since there is no fill job at an odd time position and two jobs of the same chain cannot be scheduled at the time, there are at most n jobs scheduled at every odd time. Thus only the even time positions remain to be checked. All the chains \mathcal{C}_i start at an odd time $6i + 2c_i + 1$, so by Definition 14 and abiding by exact delays every job $O_{j,k}^i$ is scheduled at time $r(O_{j,k}^i) + 2c_i + 1$, which is odd since all the release dates are even according to Definition 14. Plus it means that only fill jobs are scheduled at the even time positions of color choice segment $[0, 6n)$. Thus only the even time positions of the edge check segments in $[6n, 6(n + m))$ remain to be checked. There are $M - 1$ fill jobs scheduled at all of them. Let $j \in [0..m - 1]$ and $e_j = \{v_{i_1}, v_{i_2}\}, i_1 < i_2$. In edge check segment $[6(n + j), 6(n + j + 1))$ there are exactly two non-fill jobs to be scheduled: $J_{n+j}^{i_1}$ and $J_{n+j}^{i_2}$. As mentioned in the previous paragraph they are respectively scheduled at time $6(n + j) + 2c_{i_1}$ and $6(n + j) + 2c_{i_2}$. Since $(c_0, \dots, c_{n-1}) \in \{0, 1, 2\}^n$ is a 3-coloring and $\{v_{i_1}, v_{i_2}\} = e_j \in E$, we have $c_{i_1} \neq c_{i_2}$ and thus $\sigma(J_{n+j}^{i_1}) \neq \sigma(J_{n+j}^{i_2})$. Therefore at most M jobs are scheduled at any time position.

(\impliedby) Suppose we have a feasible schedule. For all $0 \leq i \leq n - 1$, let $s_i \in \{0, 1, 2\}$ be such that $6i + 2s_i + 1$ is the starting time of chain \mathcal{C}_i (recall that it can only be an odd time because of the fill jobs defined in Definition 15). We show that (s_0, \dots, s_{n-1}) is a 3-coloring of G . By contradiction suppose there is an edge $e_j = \{v_{i_1}, v_{i_2}\} \in E$ such that $s_{i_1} = s_{i_2}$. Then according to Lemma 17 jobs $J_{n+j}^{i_1}$ and $J_{n+j}^{i_2}$ are scheduled at the same time $6(n + j) + 2s_{i_1}$. Thus taking into account the $M - 1$ fill jobs at this time position, there are $M + 1$ jobs scheduled at the same time position, which is greater than the number of machines M . Thus the schedule is not feasible, which leads to a contradiction. Thus (s_0, \dots, s_{n-1}) is indeed a 3-coloring of G . ◀

This proves that $P|chains(\ell_{i,j}^{ex}), p_j = 1, r_j, d_j|*$ with $\ell_{max} = 1$ is NP-hard, which concludes the para-NP-completeness proof of the corresponding parameterized problem.

► **Theorem 18.** $P|chains(\ell_{i,j}^{ex}), p_j = 1, r_j, d_j|*$ is para-NP-complete when parameterized by maximum delay ℓ_{max} .

2.4 NP-hardness of $P|chains(\ell_{i,j}^{min}), p_j = 1, r_j, d_j|\star$ with $\ell_{max} = 1$

We build an instance of $P|chains(\ell_{i,j}^{min}), p_j = 1, r_j, d_j|\star$ with $\ell_{max} = 1$. The general idea is to combine the two previously proposed extensions of the basic reduction from Section 2.1: gadget chains from Section 2.2 that dealt with the extra flexibility coming from using minimum delays instead of exact ones, and propagators from Section 2.3 that helped to keep the maximum delay value equal to one. However getting the final product proved to be significantly more technical than with the previous reductions, as propagators and gadget chains could interfere with each other. The reduction will not be detailed in this section: the full description and proof is available in Appendix A.1, as well as an example in Figure 7. Instead we give insight into the major roadblock that we faced and eventually managed to overcome: interference between propagators and gadget chains.

Again the goal was to prove that *in any feasible schedule the edge jobs accurately represent the color choice*. As we have minimum delays we thought about using gadget chains like the reduction in Section 2.2. This required to replicate a situation equivalent to the one displayed in Figure 2 while propagators from other chains were around and could potentially trade places. As we needed propagators at every other time position to keep the maximum delay value at 1, it was not possible to completely isolate each triplet of chains as we did in Section 2.2 when pathwidth μ was the parameter. So when a triplet of chains was considered by the lemma it had to be guaranteed that the propagators from other chains were fixed and thus could not trade places.

This was made possible by setting the closing time windows of the chain triplets in the reverse order of the color choice time windows. Then, as shown in Figure 7, chains $C'_0, C'_{0,1}$, and $C'_{0,2}$ start at the first part of the color choice segment and end at the last part of the closing segment. Then chains $C'_1, C'_{1,1}$, and $C'_{1,2}$ start at the second part of the color choice segment and end at the second to last part of the closing segment, and so on. This way only propagators of chains $C'_j, C'_{j,1}, C'_{j,2}$ with $j < i$ might interfere. These jobs would be fixed by induction hypothesis and as such could not trade places.

Now that such a property was proved, the correspondence between valid graph 3-colorings and feasible schedules could be established the same way as in our previous reductions.

► **Proposition 19.** *G is 3-colorable if and only if there exists a feasible schedule for this instance of $P|chains(\ell_{i,j}^{min}), p_j = 1, r_j, d_j|\star$.*

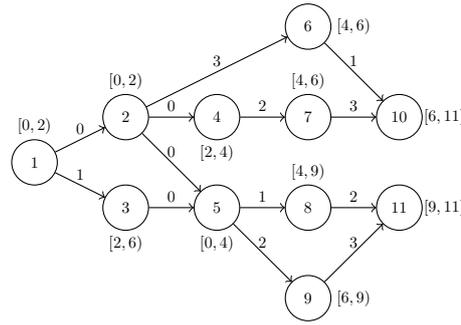
This proves that $P|chains(\ell_{i,j}^{min}), p_j = 1, r_j, d_j|\star$ with $\ell_{max} = 1$ is NP-hard, which concludes the para-NP-completeness proof of the corresponding parameterized problem.

► **Theorem 20.** *$1|chains(\ell_{i,j}^{min}), p_j = 1, r_j, d_j|\star$ is para-NP-complete when parameterized by maximum delay ℓ_{max} .*

3 A FPT algorithm with two parameters

In this section we prove that the problem with precedence delays (exact or minimum) $P|prec(\ell_{i,j}), p_i = 1, r_i, d_i|\star$ is fixed-parameter tractable with the couple of parameters (ℓ_{max}, μ) . For the sake of readability we detail the minimum delays case.

Let us consider the sorted list $x_k, k \in \{0, \dots, K\}$ of the release times and deadlines in non decreasing order. We define a sub-sequence $u_\alpha, \alpha \in \{0, \dots, \kappa - 1\}$ of this sequence so that two consecutive terms - except the last one - are separated by at least ℓ_{max} . So we set $u_0 = x_0$, then $u_{\alpha+1} = x_k$ with k the minimum value in $\{1, \dots, K\}$ such that $u_{\alpha+1} - u_\alpha \geq \ell_{max}$. Lastly we set $u_\kappa = x_K$.



■ **Figure 4** A precedence graph with minimum delays. Each precedence arc $e = (i, j)$ is labeled by the minimum delay ℓ_{ij} . Each node i is labelled by its time window $[r_i, d_i)$.

Let us consider the instance with minimum delays described in Figure 4.

For this instance we get the sequence $x_0 = 0, x_1 = 2, x_2 = 4, x_3 = 6, x_5 = 9$ and $x_6 = 11$ with $K = 6$. The associated pathwidth $\mu = 3$ is reached in the interval $[4, 6)$ crossed by intervals of jobs 3, 6, 7, 8. We also have $\ell_{max} = 3$, so we get: $u_0 = x_0 = 0, u_1 = x_2 = 4, u_2 = x_5 = 9$ and $u_3 = x_6 = 11$.

We set $X_0 = \emptyset$ and for any $\alpha \in \{1, \dots, \kappa\}$ we define $X_\alpha = \{i \in \mathcal{T}, [r_i, d_i) \cap [u_{\alpha-1}, u_\alpha) \neq \emptyset\}$ the set of jobs that could be scheduled in interval $[u_{\alpha-1}, u_\alpha)$. The idea of sequence (u_0, \dots, u_κ) is that the number of jobs in each X_α is bounded by $(\mu + 1) \times \ell_{max}$ (see Lemma 22). We also define $Z_\alpha, \alpha \in \{0, \dots, \kappa\}$ the set of jobs with a deadline not greater than u_α , i.e. $Z_\alpha = \{i \in \mathcal{T}, d_i \leq u_\alpha\}$. In our example we have: $Z_0 = \emptyset, Z_1 = \{1, 2, 4, 5\}, Z_2 = Z_1 \cup \{3, 6, 7, 8, 9\}$ and $Z_3 = \mathcal{T}$.

We define a dynamic programming scheme for our problem. The stages of the scheme are $\{0, \dots, \kappa\}$. For each stage $\alpha \in \{0, \dots, \kappa\}$ we denote N_α the set of states of stage α . A state $s \in N_\alpha$ represents the minimum information from a feasible schedule spanning in $[0, u_\alpha)$ that is necessary to extend this schedule in interval $[u_\alpha, u_\kappa)$.

Hence a state $s \in N_\alpha$ with $\alpha \in \{1, \dots, \kappa - 1\}$ is a tuple $s = (\beta, Y)$, where:

- $Y \subseteq X_\alpha - Z_\alpha$ is a subset of jobs such that $Y \cup Z_\alpha$ represents the set of jobs scheduled in $[0, u_\alpha)$.
- β is a complete schedule (i.e a set of jobs with their starting time) of the last ℓ_{max} time units before time u_α - i.e in interval $[u_\alpha - \ell_{max}, u_\alpha)$. We denote $J(\beta)$ the set of jobs scheduled in β . β is called a *border schedule*. Only such a schedule can influence the earliest starting times of the jobs not scheduled yet.

For $\alpha = \kappa$ we set $N_\kappa = \{s_\kappa\}$ with $s_\kappa = (\bullet, \emptyset)$ where \bullet is an empty schedule. Moreover $X_\kappa - Z_\kappa = \emptyset$ and so N_κ can be reduced to only one element. Similarly, we set $N_0 = \{s_0\}$ with $s_0 = (\bullet, \emptyset)$ since $X_0 = \emptyset$ and no job may be executed in interval $[u_0, u_0) = \emptyset$.

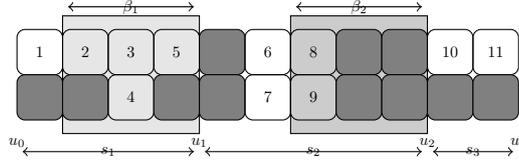
As an example let us consider a feasible schedule σ pictured by Figure 5 for $m = 2$ identical machines associated to the instance given by Figure 4. The associated states are: $s_0 = (\bullet, \emptyset), s_1 = (\beta_1, \{3\}), s_2 = (\beta_2, \emptyset)$ and $s_3 = (\bullet, \emptyset)$.

Now assume that $s = (\beta, Y) \in N_\alpha$ with $\alpha \in \{0, \dots, \kappa\}$. The boolean function $\text{ExistSched}(s)$ is set to *true* if and only if there exists a (partial) feasible schedule of jobs from $Y \cup Z_\alpha$ in time interval $[u_0, u_\alpha)$ that ends with schedule β .

We can now establish the recurrence equation for this function:

1. $\text{ExistSched}(s_0) = \text{true}$; indeed, $s_0 = (\bullet, \emptyset)$ and $Z_0 = \emptyset$, thus no job has to be scheduled.
2. Let us now consider $\alpha \in \{1, \dots, \kappa\}$. If $\text{ExistSched}(s) = \text{true}$ then there exists a feasible schedule in $[0, u_\alpha)$ which can be decomposed into a feasible schedule in $[0, u_{\alpha-1})$ associated with a state $s' \in N_{\alpha-1}$ and a schedule in the interval $[u_{\alpha-1}, u_\alpha)$ consistent with s and s' . The existence of such a schedule is denoted by the function $\text{Sched}(s, s')$.

21:14 Parameterized Complexity of a Parallel Machine Scheduling Problem



■ **Figure 5** A feasible schedule σ associated with the example given in Figure 4 for $m = 2$ machines.

We now bound the complexity of computing $\text{Sched}(s, s')$ from a tuple of states $(s', s) \in N_{\alpha-1} \times N_{\alpha}$.

Let $s = (\beta, Y)$ and $s' = (\beta', Y')$. Then boolean $\text{Sched}(s', s)$ is true if and only if there exists a schedule of $Y \cup Z_{\alpha} - Y' - Z_{\alpha-1}$ in the interval $[u_{\alpha-1}, u_{\alpha}]$ that is consistent with the border schedule β' and ends with the border schedule β .

► **Lemma 21.** *For any $\alpha \in \{1, \dots, \kappa\}$ and $(s', s) \in N_{\alpha-1} \times N_{\alpha}$, the time complexity of $\text{Sched}(s', s)$ is $\mathcal{O}(\mu^2 \times \ell_{\max}^2 \times ((\mu + 1) \times \ell_{\max})!)$.*

To prove this lemma, two more technical lemmas are needed. These lemmas bound the total number of schedulable jobs in time interval $[u_{\alpha-1}, u_{\alpha}]$ for $\alpha \in \{1, \dots, \kappa\}$:

► **Lemma 22.** $\forall \alpha \in \{0, \dots, \kappa\}, |X_{\alpha}| \leq (\mu + 1) \times \ell_{\max}$.

Proof. We simply observe that if $u_{\alpha-1} = x_k$ and $u_{\alpha} = x_{k'}$ then by construction $k' - k \leq \ell_{\max}$. Thus the inequality holds by the definition of μ . ◀

► **Lemma 23.** *For any $\alpha \in \{1, \dots, \kappa - 1\}$, $|N_{\alpha}| \leq 2^{(\mu+1) \times \ell_{\max}} \times (\ell_{\max} + 1)^{(\mu+1) \times \ell_{\max}}$.*

Proof. The total number of schedules from a set $V = J(\beta)$ is bounded by $(\ell_{\max} + 1)^{|V|}$. Thus, by Lemma 22, it is bounded by $(\ell_{\max} + 1)^{(\mu+1) \times \ell_{\max}}$. And because the number of sets $V \subseteq X_{\alpha}$ is bounded by $2^{|X_{\alpha}|} \leq 2^{(\mu+1) \times \ell_{\max}}$, the lemma holds. ◀

Now we are able to prove Lemma 21:

Proof. The problem is to schedule jobs from $S = Y \cup Z_{\alpha} - (Y' \cup Z_{\alpha-1})$ in the interval $[u_{\alpha-1}, u_{\alpha}]$ so that the schedule is consistent with the two border schedules β' and β . This can be done in several steps:

1. adjusting the release times of jobs of S with respect to the border schedule β' : if j is a successor of $i \in J(\beta')$ then $r_j = \max(r_j, \beta'(i) + 1 + \ell_{i,j})$, and propagate to precedence constraints in S .
2. adjusting the deadlines of jobs of S with respect to the border schedule β : if i is a predecessor of $j \in J(\beta)$ then $d_i = \min(d_i, \beta(j) - \ell_{i,j})$, and propagate to precedence constraints in S .
3. if a contradiction is detected at this step (a job j for which $r_j \geq d_j$), $\text{Sched}(s', s) = \text{false}$.
4. Otherwise we can enumerate all active schedules (i.e. schedules in which no job can be scheduled earlier provided the other jobs are not delayed) of $S - J(\beta)$ and verify that one of them spans in $[u_{\alpha-1}, u_{\alpha} - \ell_{\max})$.

The time complexity of the two first steps is $\mathcal{O}(|S|^2)$. For the last step it is known that any active schedule can be generated by list scheduling using a permutation of jobs [18]. Thus the enumeration of active schedules can be done by a brute force algorithm that enumerates all permutations of jobs and then performs a list scheduling algorithm to check whether the schedule spans in the interval $[u_{\alpha-1}, u_{\alpha} - \ell_{\max})$.

At most m jobs are executed at each instant, and the number of iterations is bounded by $|S|$. For each iteration, we must check that all the precedence constraints (with exact or minimum delays) are fulfilled, and thus one execution of this priority list has a complexity bounded by $\mathcal{O}(|S|^2)$.

The total number of permutations is $|S - J(\beta)|!$. Thus the overall complexity is bounded by $\mathcal{O}(|S|^2 \times |S - J(\beta)|!)$. And since $S \subseteq X_\alpha$, by Lemma 22 we get that $|S| \leq (\mu + 1) \times \ell_{max}$ and the lemma holds. \blacktriangleleft

Finally we formalize the recurrence equation that yields a FPT algorithm when we have minimum delays: if $s \in N_\alpha$,

$$\text{ExistSched}(s) = \bigvee_{s' \in N_{\alpha-1}} \text{Sched}(s', s) \wedge \text{ExistSched}(s') \quad (1)$$

► Theorem 24. *The answer to an instance \mathcal{I} of $P|prec(\ell_{ij}), p_j = 1, r_j, d_j|*$ (with minimum or exact delays) is “yes” if and only if $\text{ExistSched}(s_\kappa)$ is true. Moreover the time complexity of the computation of $\text{ExistSched}(s_\kappa)$ is $\mathcal{O}(n \times (2\ell_{max} + 2)^{2(\mu+1) \times \ell_{max}} \times \mu^2 \times \ell_{max}^2 \times ((\mu+1) \times \ell_{max})!)$.*

Proof. If $\text{ExistSched}(s_\kappa) = \text{true}$, then a sequence of states $s_0, s_1, \dots, s_\kappa$ with $s_\alpha \in N_\alpha$ for $\alpha \in \{0, \dots, \kappa\}$ and $\text{Sched}(s_{\alpha-1}, s_\alpha) = \text{true}$ for all $\alpha \in \{1, \dots, \kappa\}$ can be built. And conversely such a sequence induces a feasible schedule.

Now, the number of calls of the function Sched necessary to compute the recurrence equation (1) is proportional to $\sum_{\alpha=1}^{\kappa} |N_{\alpha-1}| \times |N_\alpha|$. By Lemma 23, this value is bounded by $\kappa \times 2^{2(\mu+1) \times \ell_{max}} \times (\ell_{max} + 1)^{2(\mu+1) \times \ell_{max}}$. Since $\kappa \leq 2n$, by Lemma 21 we get the theorem.

Notice that for exact delays, the computation of $\text{Sched}(s_{\alpha-1}, s_\alpha)$ is slightly different since the border schedule of $s_{\alpha-1}$ induces the schedule of all successors of these jobs. Similarly the border of s_α induces starting times for predecessors of these jobs, so the first step is to verify the consistency of the job starting times induced by the two border schedules. This can be done in $\mathcal{O}((\mu \times \ell_{max})^2)$. The remaining enumeration concerns the schedule of jobs without predecessors that start after $u_{\alpha-1}$. In the worst case the number of these jobs is still $\mathcal{O}(\mu \times \ell_{max})$ so that the complexity is the same as in the min delays case. \blacktriangleleft

4 Conclusion

In this paper we analyzed the parameterized complexity of two scheduling problems with precedence delays, unit processing times and job time windows with respect to two parameters: the pathwidth μ and the maximum precedence delay ℓ_{max} . To the best of our knowledge this is the first hardness result with pathwidth μ as a parameter and unit processing times, and also the first time that ℓ_{max} is considered as a parameter. Our work raises an open problem with parameter ℓ_{max} , namely the single-machine problem $1|chains(\ell_{i,j}), p_j = 1, r_i, d_i|*$ for which the hardness reductions we developed do not apply. Further work is also underway to extend our FPT algorithm to more general problems, for instance with any processing times jobs or more complex resource constraints.

References

- 1 Robert Baart, Mathijs de Weerd, and Lei He. Single-machine scheduling with release times, deadlines, setup times, and rejection. *European Journal of Operational Research*, 291(2):629–639, 2021.
- 2 S. Bessy and R. Giroudeau. Parameterized complexity of a coupled-task scheduling problem. *Journal of Scheduling*, 22(3):305–313, June 2019. doi:10.1007/s10951-018-0581-1.

21:16 Parameterized Complexity of a Parallel Machine Scheduling Problem

- 3 Hans L Bodlaender and Michael R Fellows. W[2]-hardness of precedence constrained k-processor scheduling. *Operations Research Letters*, 18(2):93–97, 1995.
- 4 Hans L Bodlaender and Marieke van der Wegen. Parameterized complexity of scheduling chains of jobs with delays. In *15th International Symposium on Parameterized and Exact Computation (IPEC)*, 2020.
- 5 P. Brucker, M.R. Garey, and D.S. Johnson. Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness. *Math. Oper. Res.*, 2(3):275–284, 1977.
- 6 R. Downey and M. Fellows. *Parameterized complexity*. Springer, 1999.
- 7 Rodney G Downey, Michael R Fellows, and Kenneth W Regan. Descriptive complexity and the W hierarchy. In *Proof Complexity and Feasible Arithmetics*, pages 119–134, 1996.
- 8 Jianzhong Du, Joseph YT Leung, and Gilbert H Young. Scheduling chain-structured tasks to minimize makespan and mean flow time. *Information and Computation*, 92(2):219–236, 1991.
- 9 J. Flum and M. Grohe. *Parameterized complexity theory*. Springer, 1998.
- 10 M.R. Garey, D.S. Johnson, R.E. Tarjan, and M. Yannakakis. Scheduling opposing forests. *SIAM Journal on Algebraic Discrete Methods*, 4(1):72–93, 1983.
- 11 Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.
- 12 Claire Hanen and Alix Munier-Kordon. Fixed-Parameter tractability of scheduling dependent typed tasks subject to release times and deadlines. *Submitted*, 2021.
- 13 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
- 14 J.K. Lenstra and A.H.G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Oper. Res.*, 26(1):22–35, 1978.
- 15 Matthias Mnich and René van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, 100:254–261, December 2018. doi:10.1016/j.cor.2018.07.020.
- 16 Alix Munier Kordon. A fixed-parameter algorithm for scheduling unit dependent tasks on parallel machines with time windows. *Discrete Applied Mathematics*, 290:1–6, 2021.
- 17 Alex J Orman and Chris N Potts. On the complexity of coupled-task scheduling. *Discrete Applied Mathematics*, 72(1-2):141–154, 1997.
- 18 Linus Schrage. Solving resource-constrained network problems by implicit enumeration – nonpreemptive case. *Operations Research*, 18(2):263–278, 1970.
- 19 J. D. Ullman. NP-complete scheduling problems. *Journal of Computer and System sciences*, 1975. URL: <https://core.ac.uk/reader/82723490>.
- 20 René van Bevern, Robert Bredebeck, Laurent Bulteau, Christian Komusiewicz, Nimrod Talmon, and Gerhard J. Woeginger. Precedence-constrained scheduling problems parameterized by partial order width. In Yury Kochetov, Michael Khachay, Vladimir Beresnev, Evgeni Nurminski, and Panos Pardalos, editors, *Discrete Optimization and Operations Research*, pages 105–120, Cham, 2016. Springer International Publishing.
- 21 René van Bevern, Andrey Melnikov, Pavel V. Smirnov, and Oxana Yu. Tsidulko. On data reduction for dynamic vector bin packing. *CoRR*, abs/2205.08769, 2022. doi:10.48550/arXiv.2205.08769.
- 22 Wenci Yu, Han Hoogeveen, and Jan Karel Lenstra. Minimizing Makespan in a Two-Machine Flow Shop with Delays and Unit-Time Operations is NP-Hard. *Journal of Scheduling*, 7(5):333–348, September 2004. doi:10.1023/B:JOSH.0000036858.59787.c2.

A

 Appendix

A.1 Description of the reduction in Section 2.4 and proof of Proposition 19

► **Definition 25** (Vertex chain \mathcal{C}'_i with $\ell_{max} = 1$). We define \mathcal{C}'_i as a chain of $3(2n + m - 2i - 1) + \deg(v_i) + 1$ jobs. These jobs will fulfill two roles:

- Propagators $O_{j,k}^i$: job $O_{i,0}^i$ will give the color choice of node v_i . The other $3(2n + m - 2i)$ jobs $O_{j,k}^i$ ($i \leq j \leq 2n + m - i - 1$, $0 \leq k \leq 2$) and $O_{2n+m-i-1,0}^i$ will propagate this color choice along the whole chain while keeping the maximum delay value at 1. Job $O_{j,k}^i$ will have time window $[6j + 2k, 6j + 2k + 3)$.
- Edge jobs J_j^i : The $\deg(v_i)$ jobs J_{n+j}^i will represent the color choice of node v_i in every edge e_j where node v_i is in ($0 \leq j \leq m - 1$). Job J_{n+j}^i will have time window $[6(n + j) + 1, 6(n + j) + 4)$.

In order to define vertex chain \mathcal{C}'_i we segment time into $m + 2$ segments: a color choice segment $[0, 6n)$, m edge check segments along $[6n, 6(n + m))$ and a closing segment $[6(n + m), 6(n + 2m))$. We describe the chain from left to right:

- (1) Color choice segment $[0, 6n)$
 - Set the first job $O_{i,0}^i$ of \mathcal{C}'_i in time window $[6i, 6(i + 1))$. Then add a unit-time minimum delay.
 - Add a job then a unit-time minimum delay, and do this $3(n - i) - 1$ times in a row. These jobs are named $O_{i,1}^i, O_{i,2}^i, O_{i+1,0}^i, \dots, O_{n,2}^i$.
- (2) Edge check segment $[6(n + j), 6(n + j + 1))$, $0 \leq j \leq m - 1$
Let edge $e_j = \{v_{i_1}, v_{i_2}\}$, $i_1 < i_2$. This segment will check if the vertices v_{i_1} and v_{i_2} have different colors.
 - a. Vertex chain \mathcal{C}'_i with $i \notin \{i_1, i_2\}$
 - Add job $O_{n+j,0}^i$ then a unit-time minimum delay then job $O_{n+j,1}^i$ then a unit-time minimum delay then job $O_{n+j,2}^i$ then a unit-time minimum delay.
 - b. Vertex chain \mathcal{C}'_i with $i = i_1$ or $i = i_2$
 - Add job $O_{n+j,0}^i$ then a minimum delay of length zero then job J_{n+j}^i then a minimum delay of length zero then job $O_{n+j,1}^i$ then a unit-time minimum delay then job $O_{n+j,2}^i$ then a unit-time minimum delay.
- (3) Closing segment $[6(n + m), 6(n + 2m))$
 - Add a job then a unit-time minimum delay, and do this $3(n - i - 1)$ times in a row. These jobs are named $O_{n+m,0}^i, O_{n+m,1}^i, O_{n+m,2}^i, O_{n+m+1,0}^i, \dots, O_{2n+m-(i+2),2}^i$.
 - Add the last job $O_{2n+m-(i+1),0}^i$ of \mathcal{C}'_i with time window $[6(2n + m - (i + 1)), 6(2n + m - (i + 1)) + 3)$.

► **Definition 26** (Gadget chain $\mathcal{C}'_{i,1}$ (resp. $\mathcal{C}'_{i,2}$)).

- Set the first job $O_{i,0}^{i,1}$ (resp. $O_{i,0}^{i,2}$) in time window $[6i, 6i + 3)$.
- Add a unit-time minimum delay then a job, and do this $3(2n + m - 2i - 1) - 1$ times in a row. These jobs are named $O_{i,1}^{i,1}, O_{i,2}^{i,1}, O_{i+1,0}^{i,1}, \dots, O_{2n+m-(i+1),0}^{i,1}$ (resp. $O_{i,1}^{i,2}, O_{i,2}^{i,2}, O_{i+1,0}^{i,2}, \dots, O_{2n+m-(i+1),0}^{i,2}$).

► **Definition 27** (Fill jobs).

- (1) Color choice segment $[0, 6n)$
Let $i \in [0, n - 1]$. In time segment $[6i, 6(i + 1))$:
 - At time $6i$: set $M - 1 - 2i$ fill jobs.
 - At time $6i + 1$: set $M - 1 - i$ fill jobs.
 - At time $6i + 2$: set $M - 2 - 2i$ fill jobs.

21:18 Parameterized Complexity of a Parallel Machine Scheduling Problem

(2) *Edge check segments* $[6n, 6(n+m))$

Let $j \in [0, m-1]$. In time segment $[6(n+j), 6(n+j+1))$:

- At time $6(n+j)+1$: set $M-n-1$ fill jobs.
- At time $6(n+j)+3$: set $M-n-1$ fill jobs.

(3) *Closing segment* $[6(n+m), 6(2n+m))$

Let $i \in [0, n-1]$. In time segment $[6(n+2m-(i+1)), 6(n+2m-i))$:

- At time $6(n+2m-(i+1))$: set $M-2-2i$ fill jobs.
- At time $6(n+2m-(i+1))+1$: set $M-1-i$ fill jobs.
- At time $6(n+2m-(i+1))+2$: set $M-1-2i$ fill jobs.

► **Lemma 28.** *Let $0 \leq i \leq n-1$. In any feasible schedule, if a chain starts at time $6i+k$ with $k \in \{0, 1, 2\}$, then all jobs J in this chain are scheduled at time $r(J)+k$ or later, where $r(J)$ is the release date of job J .*

Proof. First the result is proved for vertex chains \mathcal{C}'_i . Suppose we have a feasible schedule where vertex chain \mathcal{C}'_i starts at time $6i+l$ with $l \in \{0, 1, 2\}$.

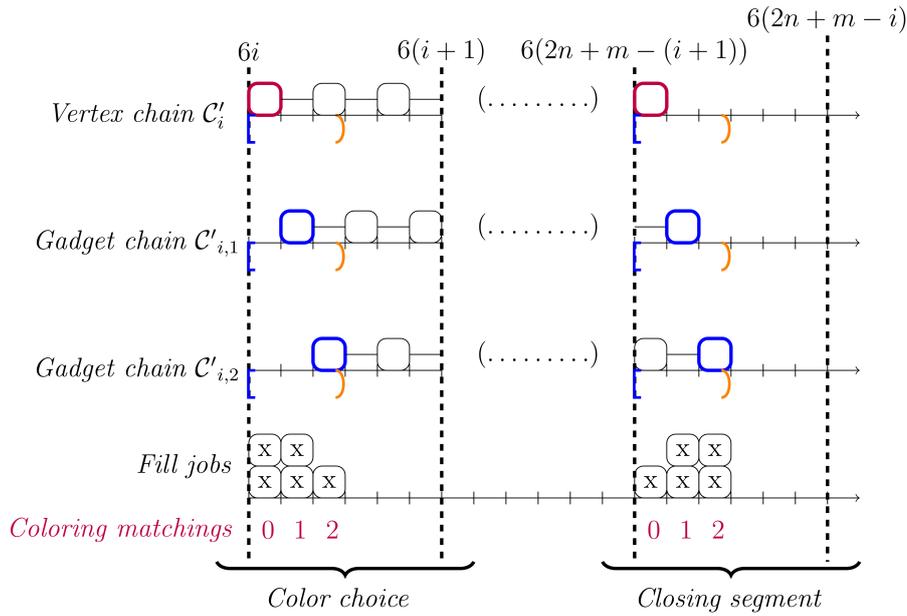
- Propagators $O_{j,k}^i$: by Definition 25 there is always either a unit-time minimum delay or a job J_j^i between two consecutive jobs $O_{j,k}^i$ in vertex chain \mathcal{C}_i . Thus if the first job $O_{i,0}^i$ is scheduled at time $6i+l = r(O_{i,0}^i) + l$ (or later), then we know that the next propagator $O_{i,1}^i$ is scheduled at time $(6i+l)+2 = r(O_{i,1}^i) + l$ or later, and so on. By induction on the couple (j, k) with $i \leq j \leq n+m-1$ and $0 \leq k \leq 2$, we get that all the jobs $O_{j,k}^i$ in vertex chain \mathcal{C}_i are scheduled at time $(6i+l) + 2 \times (3(j-i) + k) = 6j + 2k + l = r(O_{j,k}^i) + l$ or later.
- Edge jobs J_j^i : let $j_0 < j_1 < \dots < j_{deg(v_i)-1}$ be the indices of the edges e_j such that $v_i \in e_j$ (if there are any). Let $p \in [0, deg(v_i)-1]$. According to Definition 25 job $J_{j_p}^i$ is scheduled right before job $O_{j_p,0}^i$ with a minimum delay of length zero between them. Therefore according to our previous point about jobs $O_{j,k}^i$, job $J_{j_p}^i$ is scheduled at time $(r(O_{j_p,0}^i) + l) + 1 = r(J_{j_p}^i) + l$ or later.

Gadget chain $\mathcal{C}'_{i,1}$ (resp. $\mathcal{C}'_{i,2}$) only features propagators. By Definition 26 there is always a unit-time minimum delay between two consecutive jobs $O_{j,k}^{i,1}$ (resp. $O_{j,k}^{i,2}$), so the result can be proven the same way as in the first item of the proof for vertex chains. ◀

► **Lemma 29.** *Let $0 \leq i \leq n-1$. In any feasible schedule, if a chain starts at time $6i+k$ with $k \in \{0, 1, 2\}$, then all jobs J in this chain have to be scheduled at time $r(J)+k$, where $r(J)$ is the release date of job J .*

Proof. We prove by induction on $i \in [0, n-1]$ that for all $0 \leq j \leq i$ exactly one chain starts at each time $6j$, $6j+1$, $6j+2$ and all jobs J in a chain \mathcal{C}'_j , $\mathcal{C}'_{j,1}$, $\mathcal{C}'_{j,2}$ that starts at time $6j+k$ have to be scheduled at time $r(J)+k$.

- According to Definition 27 on time windows $[0, 6)$ and $[6(2n+m-1), 6(2n+m))$, the chain triplet \mathcal{C}'_0 , $\mathcal{C}'_{0,1}$, $\mathcal{C}'_{0,2}$ is in the situation described in *Figure 6*. Thus at least one chain must start at time 2 which means by Lemma 28 that this chain has to end at time $6(2n+m-1)+2$. Then time $6(2n+m-1)+2$ is blocked, so by the same lemma another chain cannot start at time 2. Thus the two other chains have to start at the two remaining time positions 0 and 1, one per chain. By Lemma 28 the chain that starts at time 1 ends at time $6(2n+m-1)+1$ (or later but the only other time position possible $6(2n+m-1)+2$ is already blocked). So time position $6(2n+m-1)+1$ is now blocked, which forces the chain that starts at time 0 to end at time $6(2n+m-1)$.



■ **Figure 6** A toy situation with three machines and the three chains related to a node in the $P|chains(\ell_{i,j}^{min}, p_j = 1, r_j, d_j)★$ reduction. In any feasible schedule featuring these chains, for $k \in \{0, 1, 2\}$ exactly one chain starts at time $6i + k$, and then this chain has to end at time $6(n + 2m - (i + 1)) + k$.

- Let $i \in [0, n - 1)$. Assume the induction hypothesis to be true for chain triplets of index j with $0 \leq j \leq i - 1$. By Definition 25 we know that only these chain triplets have propagators that might interfere in time windows $[6i, 6(i + 1))$ and $[6(2n + m - (i + 1)), 6(2n + m - i))$. By induction hypothesis we know that these propagators are fixed, and we deduce that the number of fixed jobs (propagators from other chains plus fill jobs) at the relevant time positions is the following:
 - (1) Color choice segment, part $[6i, 6(i + 1))$:
 - At time $6i$: $M - 1 - 2i$ fill jobs and $2i$ propagators from other chains which add up to $M - 1$ jobs.
 - At time $6i + 1$: set $M - 1 - i$ fill jobs and i propagators from other chains which add up to $M - 1$ jobs.
 - At time $6i + 2$: set $M - 2 - 2i$ fill jobs and $2i$ propagators from other chains which add up to $M - 2$ jobs.
 - (2) Closing segment, part $[6(n + 2m - (i + 1)), 6(n + 2m - i))$:
 - At time $6(n + 2m - (i + 1))$: set $M - 2 - 2i$ fill jobs and $2i$ propagators from other chains which add up to $M - 2$ jobs.
 - At time $6(n + 2m - (i + 1)) + 1$: set $M - 1 - i$ fill jobs and i propagators from other chains which add up to $M - 1$ jobs.
 - At time $6(n + 2m - (i + 1)) + 2$: set $M - 1 - 2i$ fill jobs and $2i$ propagators from other chains which add up to $M - 1$ jobs.

Thus we are again in the situation described in Figure 6 and we can prove the result for the triplet $C'_i, C'_{i,1}, C'_{i,2}$ the same way as in the initialization.

This concludes the proof of the lemma for all chains. ◀

Now we are able to prove Proposition 19:

21:20 Parameterized Complexity of a Parallel Machine Scheduling Problem

Proof. (\implies) Suppose we have $(c_0, \dots, c_{n-1}) \in \{0, 1, 2\}^n$ a 3-coloring of G where vertex v_i has color c_i . We propose a schedule σ where for all $0 \leq i \leq n-1$, chain \mathcal{C}'_i starts at time $6i + c_i$ and gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$ start in the two remaining time positions $6i + l_1, 6i + l_2$ in $[3i, 3(i+1))$ (with $l_1 \neq l_2$). Plus we require all delays to match their minimum value.

Then, according to Definition 25, Definition 26 and going from left to right as we did in the proof of Lemma 28, we know that propagators $O_{j,k}^i, O_{j,k}^{i,1}$ and $O_{j,k}^{i,2}$ are respectively scheduled at times $6j + 2k + c_i, 6j + 2k + l_1$ and $6j + 2k + l_2$. In the same way we know that in every edge $e_j \in E$ where node v_i appears, edge job J_{n+j}^i of vertex chain \mathcal{C}'_i is scheduled at time $6(n+j) + 1 + c_i$. Thus for all jobs J in our proposed schedule, if its chain starts at time $6i + l$ with $l \in \{0, 1, 2\}$ then it is scheduled at time $r(J) + l$.

We show that there are never more than $M = 2n + 1$ jobs scheduled at any time position. According to the previous paragraph we can infer that for every chain triplet two propagators are scheduled at every even time position and one propagator at every odd time position in time segment $[6i, 6(2n + m - (i+1)) + 3)$. Recall that only the chain triplets of index $j \leq i$ are present in the two time segments $[0, 6n), 6(n+2m - (i+1))$ related to node v_i . With Definition 27 we count the number of propagators plus the number of fill jobs at every time position and show that it is always no more than $M - 1$:

(1) Color choice segment $[0, 6n)$

Let $i \in [0, n-1]$. In time segment $[6i, 6(i+1))$:

- At time $6i$: $M - 1 - 2i$ fill jobs and $2i$ propagators which add up to $M - 1$ jobs.
- At time $6i + 1$: set $M - 1 - i$ fill jobs and i propagators which add up to $M - 1$ jobs.
- At time $6i + 2$: set $M - 2 - 2i$ fill jobs and $2i$ propagators which add up to $M - 2$ jobs.
- At times $6i + 3, 6i + 4, 6i + 5$: respectively $i, 2i, i$ propagators.

(2) Edge check segments $[6n, 6(n+m))$

Let $j \in [0, m-1]$. In time segment $[6(n+j), 6(n+j+1))$:

- At time $6(n+j) + 1$: $M - n - 1$ fill jobs and n propagators which add up to $M - 1$ jobs.
- At time $6(n+j) + 3$: $M - n - 1$ fill jobs and n propagators which add up to $M - 1$ jobs.
- At times $6(n+j), 6(n+j) + 2, 6(n+j) + 4, 6(n+j) + 5$: respectively $2n, 2n, 2n, i$ propagators.

(3) Closing segment $[6(n+m), 6(2n+m))$

Let $i \in [0, n-1]$. In time segment $[6(n+2m - (i+1)), 6(n+2m - i))$:

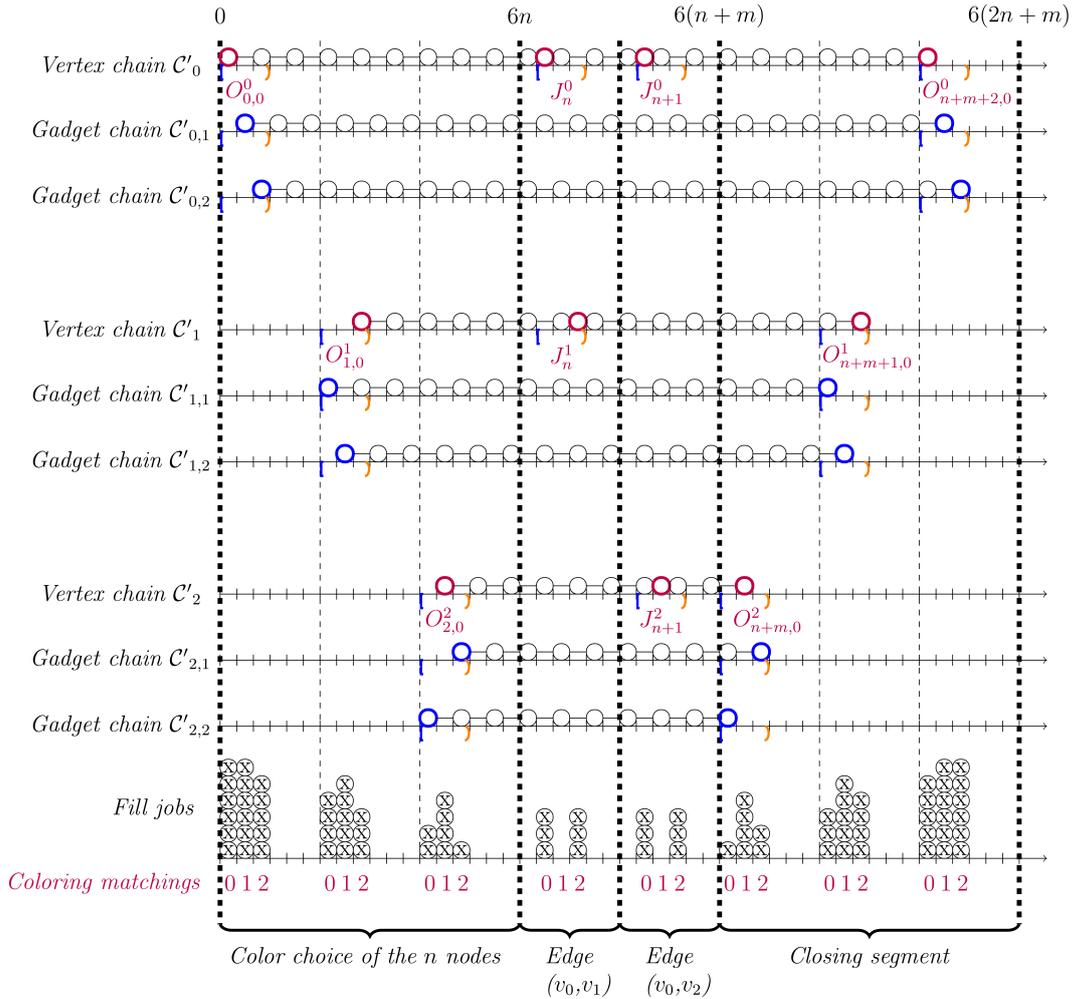
- At time $6(n+2m - (i+1))$: set $M - 2 - 2i$ fill jobs and $2i$ propagators which add up to $M - 2$ jobs.
- At time $6(n+2m - (i+1)) + 1$: set $M - 1 - i$ fill jobs and i propagators which add up to $M - 1$ jobs.
- At time $6(n+2m - (i+1)) + 2$: set $M - 1 - 2i$ fill jobs and $2i$ propagators which add up to $M - 1$ jobs.
- At times $6(n+2m - (i+1)) + 3, 6(n+2m - (i+1)) + 4, 6(n+2m - (i+1)) + 5$: respectively $i, 2i, i$, propagators.

Thus only the two edge jobs $J_{n+j}^{i_1}, J_{n+j}^{i_2}$ from an edge $e_j = \{v_{i_1}, v_{i_2}\}$ could invalidate the schedule if both jobs were scheduled at the same time. This would mean that $6(n+j) + 1 + c_{i_1} = 6(n+j) + 1 + c_{i_2}$ and thus $c_{i_1} = c_{i_2}$, which is impossible since we started from a valid 3-coloring. Therefore at most $2n + 1 = M$ jobs are scheduled at any time position.

(\impliedby) Suppose we have a feasible schedule. For all $0 \leq i \leq n-1$, let $s_i \in \{0, 1, 2\}$ be such that $6i + s_i$ is the starting time of chain \mathcal{C}'_i (recall that it can only be an odd time because of the fill jobs defined in Definition 15). We show that (s_0, \dots, s_{n-1}) is a 3-coloring

of G . By contradiction suppose there is an edge $e_j = \{v_{i_1}, v_{i_2}\} \in E$ such that $s_{i_1} = s_{i_2}$. Then according to Lemma 29 jobs $J_{n+j}^{i_1}$ and $J_{n+j}^{i_2}$ are scheduled at the same time $6(n+j) + s_{i_1}$. However according to Definition 27 and Lemma 29 fill jobs and propagators add up to $M - 1$ in all three positions $6(n+j) + 1, 6(n+j) + 2, 6(n+j) + 3$.

Thus adding both edge jobs there are $M + 1$ jobs scheduled at one of these three time positions, which would make the schedule not feasible. This leads to a contradiction. Thus (s_0, \dots, s_{n-1}) is indeed a 3-coloring of G . ◀



■ **Figure 7** An instance of $P|chains(\ell_{i,j}^{min}), p_j = 1, r_j, d_j|*$ with $M = 2n + 1 = 7$ machines representing a graph coloring. We have $G = (V, E)$ with $V = \{v_0, v_1, v_2\}$ and $E = (\{v_0, v_1\}, \{v_0, v_2\})$. This schedule corresponds to the coloring $(0, 2, 1)$.

Anti-Factor Is FPT Parameterized by Treewidth and List Size (But Counting Is Hard)

Dániel Marx  

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Govind S. Sankar  

Duke University, Durham, NC, USA

Philipp Schepper  

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract

In the general ANTI-FACTOR problem, a graph G and, for every vertex v of G , a set $X_v \subseteq \mathbb{N}$ of forbidden degrees is given. The task is to find a set S of edges such that the degree of v in S is *not* in the set X_v . Standard techniques (dynamic programming plus fast convolution) can be used to show that if M is the largest forbidden degree, then the problem can be solved in time $(M + 2)^{\text{tw}} \cdot n^{\mathcal{O}(1)}$ if a tree decomposition of width tw is given. However, significantly faster algorithms are possible if the sets X_v are sparse: our main algorithmic result shows that if every vertex has at most x forbidden degrees (we call this special case ANTI-FACTOR $_x$), then the problem can be solved in time $(x + 1)^{\mathcal{O}(\text{tw})} \cdot n^{\mathcal{O}(1)}$. That is, ANTI-FACTOR $_x$ is fixed-parameter tractable parameterized by treewidth tw and the maximum number x of excluded degrees.

Our algorithm uses the technique of representative sets, which can be generalized to the optimization version, but (as expected) not to the counting version of the problem. In fact, we show that #ANTI-FACTOR $_1$ is already #W[1]-hard parameterized by the width of the given decomposition. Moreover, we show that, unlike for the decision version, the standard dynamic programming algorithm is essentially optimal for the counting version. Formally, for a fixed nonempty set X , we denote by X -ANTI-FACTOR the special case where every vertex v has the same set $X_v = X$ of forbidden degrees. We show the following lower bound for every fixed set X : if there is an $\epsilon > 0$ such that # X -ANTI-FACTOR can be solved in time $(\max X + 2 - \epsilon)^{\text{tw}} \cdot n^{\mathcal{O}(1)}$ given a tree decomposition of width tw , then the Counting Strong Exponential-Time Hypothesis (#SETH) fails.

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

Keywords and phrases Anti-Factor, General Factor, Treewidth, Representative Sets, SETH

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.22

Related Version *Full Version*: <https://arxiv.org/abs/2110.09369> [29]

Funding Research supported by the European Research Council (ERC) consolidator grant No. 725978 SYSTEMATICGRAPH.

Philipp Schepper: Part of Saarbrücken Graduate School of Computer Science, Germany.

1 Introduction

Matching problems and their generalizations form a well studied class of problems in combinatorial optimization and computer science [26]. A *perfect matching* is a set S of edges such that every vertex has degree exactly 1 in S ; finding a perfect matching is known to be polynomial-time solvable [16, 21, 32]. In the f -FACTOR problem, an integer $f(v)$ is given for each vertex v and the task is to find a set of edges where every vertex v has degree exactly $f(v)$. A simple transformation reduces f -FACTOR to finding a perfect matching. Conversely, in f -ANTI-FACTOR the task is to find a set S of edges where the degree of v is *not* $f(v)$ [34].



© Dániel Marx, Govind S. Sankar, and Philipp Schepper;
licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 22; pp. 22:1–22:23

Leibniz International Proceedings in Informatics



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

The problems above can be unified under the GENERAL FACTOR (GENFAC) problem [10, 27, 30], where one is given a graph G and an associated set of integers B_v for every vertex v of G . The objective is to find a subgraph such that every vertex v has its degree in B_v . Cornuéjols [10] showed that the complexity of GENFAC depends on the maximum gap of the sets B_v . The maximum gap of a set B (denoted by $\max\text{-gap}(B)$) is defined as the largest contiguous sequence of integers not in B but whose boundaries are in B . Cornuéjols [10] showed that if $\max\text{-gap}(B_v) \leq 1$, then GENFAC is polynomial-time solvable. In a sense, we can say that this case is the only one that is polynomial-time solvable. Formally, for a fixed, finite set B of integers, B -FACTOR is the special case of GENFAC where every vertex has the same set $B_v = B$ of allowed degrees. It follows from a result of Dalmau and Ford [13] that if B is a fixed finite set such that $\max\text{-gap}(B) > 1$, then B -FACTOR is NP-hard.

Given the hardness of B -FACTOR in general, Marx et al. [30] studied the complexity of the problem on bounded treewidth graphs. Recall the long history of study on treewidth, which is a measure for how “tree-like” a graph is, [3, 4, 6]. For a wide range of hard problems, algorithms with running time of the form $f(k) \cdot n^{\mathcal{O}(1)}$ exist if the input graph comes with a tree decomposition of width k . In many cases even the best possible form of $f(k)$ in the running time is known (under suitable complexity assumptions, such as the Strong Exponential Time Hypothesis (SETH) [23]). Marx et al. [30] use a combination of standard dynamic programming techniques with fast subset convolution (cf. [36]) to give optimal (under SETH) $(\max B + 1)^{\text{tw}} n^{\mathcal{O}(1)}$ time algorithms for the decision, optimization, and counting versions.

- **Theorem 1.1** (Theorems 1.3–1.6 in [30]). *Fix a finite, non-empty set $B \subseteq \mathbb{N}$.*
 - *We can count in time $(\max B + 1)^{\text{tw}} n^{\mathcal{O}(1)}$ the solutions of a certain size for a B -FACTOR instance if we are given a tree decomposition of width tw .*
 - *For any $\epsilon > 0$, there is no $(\max B + 1 - \epsilon)^{\text{pw}} n^{\mathcal{O}(1)}$ algorithm for the following problems, even if we are given a path decomposition of width pw , unless SETH (resp. #SETH) fails:*
 - *B -FACTOR and MIN- B -FACTOR if $0 \notin B$ and $\max\text{-gap}(B) > 1$,*
 - *MAX- B -FACTOR if $\max\text{-gap}(B) > 1$,*
 - *# B -FACTOR if $B \neq \{0\}$.*

We study the complementary problem of X -ANTIFACTOR for finite sets X of excluded degrees.

► **Definition 1.2** (X -ANTIFACTOR). *Let $x \in \mathbb{N}$ be fixed. ANTIFACTOR_x is the decision problem of finding for an undirected graph G where all vertices v are assigned a finite set $X_v \subseteq \mathbb{N}$ with $|X_v| \leq x$, a set $S \subseteq E(G)$ such that for all $v \in V$ we have $\deg_S(v) \notin X_v$.*

For a fixed $X \subseteq \mathbb{N}$ with $|X| = x$, we define X -ANTIFACTOR as the restriction of ANTIFACTOR_x to those graphs where all vertices are labeled with the same set X .

► **Note.** \overline{X} -FACTOR, the special case of GENFAC where every vertex has set \overline{X} , precisely corresponds to X -ANTIFACTOR where we set $\overline{X} := \mathbb{N} \setminus X$.

The decision and minimization versions are trivially solvable if $0 \notin X$ as the empty set is a valid solution. Further, if X does not contain two consecutive numbers, then \overline{X} has no gap of size at least two. In this case, by results from Cornuéjols [10] and Dudyycz and Paluch [15], the decision, maximization and minimization version of \overline{X} -FACTOR are poly-time solvable.

Our Results. One could expect that similar results can be obtained for X -ANTIFACTOR as for B -FACTOR, but this is very far from the truth and the exact complexity of X -ANTIFACTOR is much less clear. In the B -FACTOR problem, a partial solution (a set of edges that we intend to further extend to a solution) can have degree at most $\max B$ at each vertex, which

is the main reason one needs $(\max B + 1)^{\text{tw}} n^{\mathcal{O}(1)}$ running time. For X -ANTIFACTOR, a vertex can also have degree larger than $\max X$ in a (partial) solution, but all degrees larger than $\max X$ are equivalent in some sense. Therefore, the natural running time we expect is $(\max X + 2)^{\text{tw}} n^{\mathcal{O}(1)}$. We show that this running time can be achieved, but requires some modification of the convolution to handle the state “degree more than $\max X$.”

► **Theorem 1.3.** *Let $X \subseteq \mathbb{N}$ be finite and fixed. Given an X -ANTIFACTOR instance and its tree decomposition of width tw . Then we can count the number of solutions of size exactly s in time $(\max X + 2)^{\text{tw}} n^{\mathcal{O}(1)}$ for all s simultaneously.*

However, there are many cases where algorithms significantly faster than $(\max X + 2)^{\text{tw}} n^{\mathcal{O}(1)}$ are possible. At first, this may seem unlikely: at each node of the tree decomposition, the partial solutions can have up to $(\max X + 2)^{\text{tw}+1}$ different equivalence classes¹ and it may seem necessary to find a partial solution for each of these classes. Nevertheless, we show that the technique of *representative sets* can be used to achieve a running time lower than the number of potential equivalence classes. Representative sets were defined by Monien [33] for use in an FPT algorithm for k -PATH, and subsequently found use in many different contexts, including faster dynamic programming algorithms on tree decompositions [1, 5, 7, 17, 18, 19, 25, 31, 35]. The main idea is that we do not need to find a partial solution for each equivalence class, but it is sufficient to find a representative set of partial solutions such that if there is a partial solution that is compatible with some extension, then there is a partial solution in our set that is also compatible with this extension. Our main algorithmic result shows that if X is sparse, then this representative set can be much smaller than $(\max X + 2)^{\text{tw}+1}$, yielding improved algorithms. In particular, ANTIFACTOR_x is FPT parameterized by tw and x .

► **Theorem 1.4.** *One can decide in time $(x + 1)^{\mathcal{O}(\text{tw})} n^{\mathcal{O}(1)}$ whether there is a solution of a certain size for ANTIFACTOR_x assuming a tree decomposition of width tw is given.*

We note that Theorem 1.4 clearly distinguishes X -ANTIFACTOR from B -FACTOR. By the known lower bounds from Marx et al. [30] (cf. Theorem 1.1), a similar result for B -FACTOR is not possible. In light of Theorem 1.4, it is also far from obvious to determine, the exact complexity of X -ANTIFACTOR for a fixed set X . The combinatorial properties of the set X influence the complexity of the problem in a subtle way and new algorithmic techniques seem to be needed to fully exploit this. Currently, we do not have a tight bound similar to Theorem 1.1 for every fixed X . Instead we propose a candidate for the combinatorial property that influences the complexity: We define a bipartite compatibility graph for every set X and conjecture that the maximum size of a so-called *half-induced matching* is the key property to obtain a faster algorithm via representative sets. See Conjecture 4.5 for a formal statement.

We use such half-induced matchings of large size to show a lower bound for ANTIFACTOR_x that, assuming SETH, complements the algorithm in Theorem 1.4 up to constant factors in the exponent (see Theorem 5.5). Moreover, if there is a half-induced matching of size h , then, assuming SETH, we show that there is no $(h - \epsilon)^{\text{tw}} n^{\mathcal{O}(1)}$ algorithm for X -ANTIFACTOR for any $\epsilon > 0$ (Theorem 5.4). Although, in this case the representative set cannot be smaller than $(h - \epsilon)^{\text{tw}+1}$ for any $\epsilon > 0$ (Lemma 4.4) we do not have matching upper bounds at this point. There are two main reasons why it is difficult to obtain tight upper bounds:

¹ Recall that in a graph with treewidth tw , the largest bag has size $\text{tw} + 1$.

- **Representative set bounds.** In Theorem 1.4, the upper bound on the size of representative sets are based on earlier algebraic techniques [18, 19, 25, 35]. It is not clear how they can be extended to the combinatorial notion of half-induced matchings.
- **Join nodes.** Even if we have tight bounds on the size of representative sets there is an additional issue that can increase the running time. At join nodes of the tree decomposition, we need to compute from two representative sets a third one. Doing this operation in a naive way results in a running time that is at least the *square* of the bound on the size of the representative set. If we want to have a running time that matches the size of the representative set, we need a more clever way of handling join nodes.

Representative sets of the form we study here could be relevant for other problems and tight bounds for such representative sets could be of fundamental importance. In particular, the notion of half-induced matchings could be a key property in other contexts as well.

Counting Problems. We also investigate the $\#\text{ANTIFACTOR}$ problem, where we need to count the total number of solutions satisfying the degree constraints. The idea of representative sets is fundamentally incompatible with exact counting: if we need to count every solution, then we cannot ignore certain partial solutions even if they can be always replaced by others. Therefore, the algorithm of Theorem 1.4 cannot be extended to the counting version.² In fact, we show that already $\#\text{ANTIFACTOR}_1$ is unlikely to be FPT by showing the following stronger statement for path decompositions.

► **Theorem 1.5.** *There is a fixed constant c such that $\#\text{ANTIFACTOR}_1$ cannot be solved in time $\mathcal{O}(n^{\text{pw}-c})$ on graphs with n vertices given a path decomposition of width pw , unless $\#\text{SETH}$ is false. Furthermore, $\#\text{ANTIFACTOR}_1$ is $\#\text{W}[1]$ -hard parameterized by pathwidth.*

Recall that $\#\text{SETH}$ (cf. [11, 14]) is actually a weaker assumption than SETH . Hence, the first result is stronger than a version based on SETH . Moreover, the algorithm from Theorem 1.3 is essentially optimal for $\#X\text{-ANTIFACTOR}$.

► **Theorem 1.6.** *Let $X \subseteq \mathbb{N}$ be a non-empty, finite and fixed set. For any constant $\epsilon > 0$, there is no algorithm that can solve $\#X\text{-ANTIFACTOR}$ in time $(\max X + 2 - \epsilon)^{\text{pw}} n^{\mathcal{O}(1)}$ given a graph along with a path decomposition of width pw , unless $\#\text{SETH}$ fails.*

Organization. Section 2 presents the algorithm of Theorem 1.4 and Section 3 shows how to compute representative sets. Section 4 introduces half-induced matchings and discusses some combinatorial properties related to representative sets. Sections 5 and 6 present the lower bounds for the decision and counting versions, respectively.

2 Algorithms

In this section, we use without loss of generality “nice” tree decompositions that have *introduce edge nodes* (see, e.g., [12] for formal definitions). When given a node t of a tree decomposition, we denote by B_t the bag of t , by V_t the vertices introduced at the subtree rooted at t , and by E_t the edges introduced in the subtree rooted at t .

We leave the proof of Theorem 1.3 to the full version of this paper, as many of the details are similar to that in [30].

² Counting the solutions approximately is a problem of independent interest.

2.1 Parameterizing by the Number of Excluded Degrees

In this section we prove Theorem 1.4 which shows that ANTIFACTOR_x is FPT parameterized by treewidth and the *size* x of the set. We first show a naive algorithm, i.e. the standard dynamic programming approach, solving the problem. In a second step we improve this algorithm by using representative sets. That is, we do not store all solutions but only so much information such that we can correctly solve the decision and optimization version.

2.1.1 Naive Algorithm

Let $X_v \subseteq \mathbb{N}$ be the set assigned to vertex v with $|X_v| \leq x$. Let n be the number of vertices of G and m the number of edges. Let $U = [0, n]$ be the universe of the values in the following.

The idea is to fill a table $\text{ParSol}[\cdot, \cdot]$ with partial solutions. That is, for all nodes t of the tree decomposition with bag B_t of size k and all $s \in [0, m]$, we have $\text{ParSol}[t, s] \subseteq U^{B_t}$ and $a \in \text{ParSol}[t, s]$ if and only if there is a set $S \subseteq E_t$ with $|S| = s$ such that $\deg_S(v) \notin X_v$ for all $v \in V_t \setminus B_t$ and $\deg_S(v) = a[v]$ for all $v \in B_t$.

Dynamic Program. Initialize the table ParSol with \emptyset for every entry. We fill the table iteratively for all nodes t of the tree decomposition and all $s \in [0, m]$ in the following way, depending on s and the type of t .

Leaf Node. As $B_t = \emptyset$, we set $\text{ParSol}[t, 0] := \{\emptyset\}$.

Introduce Vertex Node. Assume v is introduced at t , i.e. $B_t = B_{t'} \cup \{v\}$. We define

$$\text{ParSol}[t, s] := \{a_{v \rightarrow 0} \mid a \in \text{ParSol}[t', s]\}.$$

Introduce Edge Node. Assume the edge $e = uv$ is introduced at the node t . We combine the cases where e is not selected for the solution and where e is selected. Thus, we define:

$$\text{ParSol}[t, s] := \text{ParSol}[t', s] \cup \{a_{u \rightarrow a(u)+1, v \rightarrow a(v)+1} \mid a \in \text{ParSol}[t', s-1]\}.$$

Forget Node. Assume vertex v is forgotten at t , i.e. $B_t = B_{t'} \setminus \{v\}$. We define

$$\text{ParSol}[t, s] := \{a_{|B_t} \mid a \in \text{ParSol}[t', s] : a[v] \notin X_v\}.$$

Join Node Assume t_1 and t_2 are the two children of t with $B_t = B_{t_1} = B_{t_2}$. Then we define

$$\text{ParSol}[t, s] := \{a_1 + a_2 \mid a_1 \in \text{ParSol}[t_1, s_1], a_2 \in \text{ParSol}[t_2, s_2], s_1 + s_2 = s\}.$$

Let r be the root of the tree decomposition with $B_r = \emptyset$. For a given $s \in [0, m]$, the algorithm finally checks if $\text{ParSol}[r, s] \neq \emptyset$, i.e. $\text{ParSol}[r, s]$ contains the empty vector. Otherwise no solution exists. The correctness of this algorithm follows directly from its definition. Note that the computation might take time $\Omega(n^{\text{tw}+1})$ since the largest bag has size $\text{tw} + 1$.

2.1.2 Improving the Naive Algorithm

The final algorithm is based on the naive algorithm but makes use of so-called representative sets to keep the size of the set stored for each node of the tree decomposition small.

We first define the notion of representative set to state the final algorithm. In Section 3 we show how to actually compute the representative sets.

► **Definition 2.1** (*H-Compatibility*). Let $H = (U \dot{\cup} V, E)$ be an undirected (potentially infinite) bipartite graph. We say that $a \in U$ is *H-compatible* with $b \in V$, denoted by $a \sim_H b$, if $(a, b) \in E$.³

Based on this compatibility notation, we define the *H-representation* of a set.

► **Definition 2.2** (*H-Representation*). Let $H = (U \dot{\cup} V, E)$ be an undirected (potentially infinite) bipartite graph. For any $\mathcal{S} \subseteq U$, we say that $\mathcal{S}' \subseteq \mathcal{S}$ *H-represents* \mathcal{S} , denoted by $\mathcal{S}' \subseteq_{H\text{-rep}} \mathcal{S}$ if for every $b \in V$: $\exists a \in \mathcal{S} : a \sim_H b \iff \exists a' \in \mathcal{S}' : a' \sim_H b$.

For the algorithm we make use of this *H-compatibility* and *H-representation* where we use the following graphs.

► **Definition 2.3** (*Compatibility Graph*). For a set $B = \{v_1, \dots, v_k\}$ of k vertices with sets X_1, \dots, X_k of excluded degrees, we define the *compatibility graph* \mathcal{C}_B as follows:

- $V(\mathcal{C}_B) = U^k \dot{\cup} V^k$ where the elements in U, V are copies of numbers, i.e. $U, V = \mathbb{N}$.
- $E(\mathcal{C}_B) = \{((i_1, \dots, i_k), (j_1, \dots, j_k)) \mid \forall \ell \in [k], i_\ell + j_\ell \notin X_\ell\}$.

For a node t with bag B_t of the tree decomposition we denote by \mathcal{C}_t the graph \mathcal{C}_{B_t} .

The intuition is that the vertices in U^k represent the degrees of the constructed partial solution. The vertices in V^k correspond to the degrees of some (disjoint) partial solution one might see in the future. The edges then “check” whether both solutions can be combined, i.e. the degree of each vertex is valid with respect to the union of the solutions.

Final Algorithm. The improved algorithm applies the same operations as the naive algorithm to fill a table c . Then the algorithm computes a \mathcal{C}_t -representative set for the table entries and just stores these values in c . Only these values are used in the next steps to compute the other table entries. The correctness follows by induction on the tree decomposition.

▷ **Claim 2.4.** For all t, s : $c[t, s] \subseteq_{\mathcal{C}_t\text{-rep}} \text{ParSol}[t, s]$.

► **Lemma 2.5.** Assume there is an algorithm that can, for given $B = \{v_1, \dots, v_k\}$ with $|X_v| \leq x$ for all $v \in B$, compute for a set $\mathcal{S} \subseteq [0, n]^k$ a new set $\mathcal{S}' \subseteq_{\mathcal{C}_B\text{-rep}} \mathcal{S}$ of size $\text{Size}(k)$ in time $\text{Time}(k, |\mathcal{S}|)$, where Time and Size are allowed to depend on \mathcal{C}_B and x .

Then we can decide for a given ANTIFACTOR_x instance, whether there is a solution of size exactly s in time $\text{Time}(\text{tw}+1, (m+1) \cdot \text{Size}(\text{tw}+1)^2) n^{\mathcal{O}(1)}$ and $\text{Time}(\text{pw}+1, 2 \cdot \text{Size}(\text{pw}+1)) n^{\mathcal{O}(1)}$ given a tree and a path decomposition of width tw and pw , respectively.

Proof. We can assume that Time and Size are non-decreasing functions and inductively that the size of the given table entries is bounded by $\text{Size}(\text{tw} + 1)$. The running time follows immediately by bounding the size of $c[t, s]$ and then computing its representative set. The correctness follows directly from Claim 2.4. ◀

3 Computing Representative Sets

As mentioned in the previous section, one can think of \mathcal{C}_t -compatibility as checking whether the given partial solution of degree a fits together with some partial solution of degree c arriving in the future. This is done via the bipartition of the compatibility graph and the (non-)existence of the edges, i.e. checking if $a + c$ is not in X . To compute the representative set we avoid this two step procedure by defining the more standard k - q -compatibility.

³ Though the graph is undirected, we use tuples to denote the edges. By this the first value denotes the vertex from U and the second value the vertex from V .

► **Definition 3.1** (*k-q-Compatibility*). Let k, q be positive integers. For an $a \in \mathbb{N}^k$ and a $b \in \binom{\mathbb{N}}{q}^k$, we say a is k - q -compatible with b , denoted by $a \sim_q^k b$, if and only if for all $i \in [k]$ it holds that $a[i] \notin b[i]$.

For our purposes we can relate the two compatibility definitions as follows: In \mathcal{C}_t -compatibility one computes $a + c$ and checks if $a + c \notin X$. Instead k - q -compatibility checks if $a \notin X - c$. While both checks are equivalent at this point, the new compatibility version considers *all* possible sets of size at most $q = |X|$ and not just $X - c$ for all c . Hence, k - q -compatibility is independent from the sets X_v which are assigned to the vertices v of the graph.

We extend the notion of compatibility in the standard way to k - q -representation.

► **Definition 3.2** (*k-q-Representation*). Let k, q be positive integers. Given a set $\mathcal{S} \subseteq \mathbb{N}^k$, and a set $\mathcal{S}' \subseteq \mathcal{S}$. We say \mathcal{S}' k - q -represents \mathcal{S} , denoted by $\mathcal{S}' \subseteq_{q\text{-rep}}^k \mathcal{S}$, if and only if for all $b \in \binom{\mathbb{N}}{q}^k$: $\exists a \in \mathcal{S} : a \sim_q^k b \iff \exists a' \in \mathcal{S}' : a' \sim_q^k b$.

For both notations we omit the value k from the notation if $k = 1$. It remains to check that k - q -compatibility generalizes \mathcal{C}_t -compatibility. This follows by folding and unfolding the definitions of the two types of compatibility.

► **Lemma 3.3.** Let B be a set of k vertices where each $v \in B$ is assigned a set X_v such that $|X_v| \leq x$. Then the following holds for all $\mathcal{S}, \mathcal{S}' \subseteq \mathbb{N}^k$: If $\mathcal{S}' \subseteq_{x\text{-rep}}^k \mathcal{S}$, then $\mathcal{S}' \subseteq_{\mathcal{C}_B\text{-rep}} \mathcal{S}$.

Matroids. For the computation of the representative sets we make use of uniform matroids. They allow us to formally state the operations we are using.

► **Definition 3.4** (*Uniform Matroid*). Let U be some universe with n elements and $r \in \mathbb{N}$. Then $\mathcal{U}_{r,n} = (U, \binom{U}{\leq r})$ is the uniform matroid of rank r , that is the matroid over the ground set U and the independent sets are all subsets of U of size at most r .

Later the rank of these uniform matroids corresponds to the number of excluded degrees (plus one). Since the matroid contains all subset of size at most the rank, we automatically consider all possibilities for upcoming solutions.

There are results proving the existence of small representative sets for matroids [18, 19, 25]. Since these results are usually for general matroids, they also apply to uniform matroids which we use here. However, as we are not considering a single matroid but the product of several matroids, the previous results can only be applied partially to our setting. Moreover, one can suspect that these results can be improved by exploiting properties of the uniform matroids. In the following we show one approach to compute the representative sets. A second method, not using matrix multiplication, is given in the full version of the paper [29]. That algorithm yields a slightly faster algorithm when parameterizing by pathwidth.

The Algorithm. We base our method on a previous result for computing representative sets. Despite the fact that the following lemma is a special case of Lemma 3.4 in [25], our proof uses a completely different technique as we exploit that the given matroids are uniform.

Let ω be the matrix multiplication coefficient in the following, i.e. $\omega < 2.37286$ [2].

► **Lemma 3.5.** Let $\mathcal{M}_1, \dots, \mathcal{M}_k$ be k uniform matroids, each of rank r , with integer universes U_1, \dots, U_k . Given a set $\mathcal{S} \subseteq U_1 \times \dots \times U_k$ we can find a set $\mathcal{S}' \subseteq_{r-1\text{-rep}}^k \mathcal{S}$ of size r^k in time $\mathcal{O}(|\mathcal{S}| \cdot r^{k(\omega-1)})$.

Proof Idea. We follow the ideas behind the proof of Theorem 12.15 in [12] where a variant of this lemma is shown for $k = 1$ with a general matroid.

Let M_1, \dots, M_k be $r \times |U|$ matrices representing the matroids $\mathcal{M}_1, \dots, \mathcal{M}_k$, which are known to exist. Enumerate all $I \in [r]^k$ in an arbitrary order I_1, \dots, I_{r^k} and compute for all $A \in \mathcal{S}$ the vector v_A , where for all $j \in [r^k]$ we set $v_A[j] = \prod_{i=1}^k M_i[I_j[i], A[i]]$. Construct a $r^k \times |\mathcal{S}|$ matrix Q with the vectors v_A as columns and find a column basis B_Q of Q . Output the set $\mathcal{S}' = \{A \mid v_A \in B_Q\}$ as solution. Since B_Q is a basis, it contains at most r^k elements.

Computing all v_A takes time $\mathcal{O}(|\mathcal{S}| \cdot r^k \cdot k)$ in total. As the computation of the basis takes time $\mathcal{O}(|\mathcal{S}| \cdot r^{k(\omega-1)})$, the complete procedure requires time $\mathcal{O}(|\mathcal{S}| \cdot r^{k(\omega-1)} \cdot k)$.

The basic idea of the correctness proof is to characterize the compatibility by a product of determinants of certain submatrices of each M_i . Then rewrite this by the Laplacian expansion using the I_j and exploit that B_Q is a basis. The formal proof is given in the full version as it is rather technical and does not give any insight into the problem. ◀

To finish the algorithm for ANTIFACTOR_x from Theorem 1.4, it remains, by Lemma 3.3, to compute a k - x -representative set as $|X_v| \leq x$. To achieve this we define k uniform matroids with universe $\{0, \dots, n\}$ and rank $x + 1$. Then, plugging in the values from Lemma 3.5 into Lemma 2.5, directly gives the following result. Note that we can assume $x \leq n$.

► **Corollary 3.6.** *Given a tree and a path decomposition, ANTIFACTOR_x can be solved in time $(x + 1)^{(\omega+1) \cdot \text{tw}} n^{\mathcal{O}(1)}$ and $(x + 1)^{\omega \cdot \text{pw}} n^{\mathcal{O}(1)}$, respectively.*

4 Half-Induced Matchings

In this section we introduce *half-induced matchings* and show relations to compatibility graphs and representative sets. We use these properties later in the lower bounds for the decision and optimization version of X - ANTIFACTOR and ANTIFACTOR_x . The proofs of all results in this section are given in Appendix A.

► **Definition 4.1 (Half-induced Matching).** *Let $G = (U \dot{\cup} V, E)$ be a bipartite graph. G has a half-induced matching of size ℓ if there are pairwise different $a_1, \dots, a_\ell \in U$ and pairwise different $b_1, \dots, b_\ell \in V$ such that (1) $(a_i, b_i) \in E$ for all i but (2) $(a_i, b_j) \notin E$ for all $j > i$.*

By an abuse of notation, \mathcal{C}_X denotes the compatibility graph for a vertex with set X of forbidden degrees. We show that arithmetic progressions in the set of excluded degrees are sufficient to obtain large half-induced matchings in the corresponding compatibility graph.

► **Lemma 4.2.** *If X contains an arithmetic progression of length ℓ , but not one of length $\ell + 1$, then \mathcal{C}_X has a half-induced matching of size $\ell + 1$.*

Conversely to the previous lemma, we also prove that arithmetic progressions are necessary to obtain large half-induced matchings.

► **Lemma 4.3.** *Let $X \subseteq \mathbb{N}$ with $|X| = \ell \geq 2$. Suppose \mathcal{C}_X contains a half-induced matching of size $\ell + 1$. Then X is an arithmetic progression.*

For a graph \mathcal{C} and an integer $k > 1$, we extend $\sim_{\mathcal{C}}$ and $\subseteq_{\mathcal{C}\text{-rep}}$ to k dimensions, denoted by $\sim_{\mathcal{C}}^k$ and $\subseteq_{\mathcal{C}\text{-rep}}^k$, such that the \mathcal{C} -compatibility must hold for each dimension.

► **Lemma 4.4.** *Let $X \subseteq \mathbb{N}$, and $\epsilon > 0$ and $\ell \geq 2$ be constants. Then there exists a constant k depending only on ϵ and ℓ such that the following holds. Suppose the compatibility graph \mathcal{C}_X contains a half-induced matching of size ℓ . Then there is a set $\mathcal{S} \subseteq \mathbb{N}^k$ such that every representative set $\mathcal{S}' \subseteq_{\mathcal{C}_X\text{-rep}}^k \mathcal{S}$ has size $|\mathcal{S}'| \geq (\ell - \epsilon)^k$.*

The proof is given in Appendix A but we briefly discuss its implications. The running time of the algorithm for X -ANTIFACTOR from Theorem 1.4 depends on the size of the representative sets computed. This lemma implies that any such algorithm using representative sets in a similar way takes time at least $(\ell - \epsilon)^{\text{pw}}$. This can be seen as an *unconditional* version of the lower bound for the decision version shown in Theorem 5.4.

We conjecture that the converse of the above lemma is also true. For example, for $X = \{10, 100, 1000, \dots\}$ the largest half-induced matching in \mathcal{C}_X is of size three, a constant, (even though X itself is infinite). Intuitively, the size of the representative set itself must be small because knowing any *two* forbidden degrees of a vertex in the future solution is enough for us to deduce the degree of the vertex in the partial solution.

► **Conjecture 4.5.** *Let $X \subseteq \mathbb{N}$ and $\ell \geq 2$ be a constant. Then there exists a constant k depending only on ℓ such that the following holds. Suppose the largest half-induced matching in \mathcal{C}_X has size ℓ . Then every $\mathcal{S} \subseteq \mathbb{N}^k$ has a representative set $\mathcal{S}' \subseteq_{\mathcal{C}_X\text{-rep}}^k \mathcal{S}$ with $|\mathcal{S}'| \leq \ell^{k+o(k)}$.*

Recall, the runtime of the algorithm in Theorem 1.3 depends on $\max X$ but the lower bound in Theorem 5.4 on the size ℓ of the half-induced matching. With the conjecture it seems reasonable to get algorithms for the decision and optimization version based on representative sets with a running time depending on ℓ . This would complement the lower bound. Note, for the counting version the algorithm is essentially optimal (Theorem 6.3).

5 Lower Bounds for the Decision Version

In this section we prove the lower bounds for the decision version of X -ANTIFACTOR and ANTIFACTOR_x . Instead of showing the lower bound directly, we first define the following intermediate problem and show the hardness of this problem.

► **Definition 5.1** (X -ANTIFACTOR $^{\mathcal{R}}$). *Let $X \subseteq \mathbb{N}$ be fixed and finite. Let $G = (V_S \cup V_C, E)$ be a vertex labeled graph such that*

- *all vertices in V_S , called simple vertices, are labeled with set X ,*
- *all vertices $v \in V_C$, called complex vertices, are labeled with a relation R_v that is given as a truth table such that $R_v \subseteq 2^{I(v)}$ where $I(v)$ is the set of edges incident to v in G .*

A set $\hat{E} \subseteq E$ is a solution for G if (1) for $v \in V_S$: $\deg_{\hat{E}}(v) \notin X$ and (2) for $v \in V_C$: $I(v) \cap \hat{E} \in R_v$.

X -ANTIFACTOR WITH RELATIONS (X -ANTIFACTOR $^{\mathcal{R}}$) is the problem of deciding if such an instance G has a solution.

We show our lower bounds based on this problem definition.

► **Lemma 5.2** (Lower Bound for X -ANTIFACTOR $^{\mathcal{R}}$). *Let $X \subseteq \mathbb{N}$ be a fixed set which contains a half-induced matching of size $h \geq 2$.*

Let $f_X : \mathbb{N} \rightarrow \mathbb{R}^+$ be an arbitrary function that may depend on the set X .

For every constant $\epsilon > 0$, there is no algorithm that can solve X -ANTIFACTOR $^{\mathcal{R}}$ in time $(h - \epsilon)^{\text{pw} + f_X(\Delta^)} n^{\mathcal{O}(1)}$, where $\Delta^* = \max_{\text{bag } B} \sum_{v \in B \cap V_C} \deg(v)$, even if we are given a path decomposition of width pw , unless SETH fails.*

In a second step we remove the relations and replace them by appropriate gadgets. To be able to reuse the reduction later we introduce a slightly more general version of the problem. For two finite sets $X, Y \subseteq \mathbb{N}$, we define (X, Y) -ANTIFACTOR as the generalization of X -ANTIFACTOR where we allow the sets X and Y to be assigned to the vertices. We show hardness when $0 \in X$ and when $\max\text{-gap}(\overline{X}) > 1$. The former is to ensure there are no

trivial solutions and the latter ensures that the problem is not polynomial-time solvable [10]. Recall that $\max\text{-gap}(\overline{X})$ is the size of the largest contiguous sequence of integers not in \overline{X} but whose boundaries are in \overline{X} .

► **Lemma 5.3.** *Fix a finite set $X \subseteq \mathbb{N}$ such that $0 \in X$ and $\max\text{-gap}(\overline{X}) > 1$. Let $Y \subseteq \mathbb{N}$ be arbitrary. There is a many-one reduction from $Y\text{-ANTIFACTOR}^{\mathcal{R}}$ to $(X, Y)\text{-ANTIFACTOR}$ such that pathwidth increases by at most $f(\Delta^*)$ and size by a factor of $f(\Delta^*)$, where $\Delta^* = \max_{B \in \mathcal{B}} \sum_{v \in B \cap V_C} \deg(v)$.*

The proof essentially follows a similar approach as the one for the hardness of $B\text{-FACTOR}$ given in [30]. However, we deal with the cofinite set \overline{X} , and thus the constructions do not carry over directly. We do a careful check of their constructions and give necessary modifications in the full version.

By combining the lower bound for the intermediate problem with this reduction, we can show the lower bounds for $X\text{-ANTIFACTOR}$ and ANTIFACTOR_x .

► **Theorem 5.4 (Lower Bound for Decision Version I).** *Fix a finite set $X \subseteq \mathbb{N}$ such that*

- $0 \in X$ and $\max\text{-gap}(\overline{X}) > 1$,
- and X contains a half-induced matching of size h .

For every constant $\epsilon > 0$, there is no algorithm that can solve $X\text{-ANTIFACTOR}$ in time $(h - \epsilon)^{\text{pw}} n^{\mathcal{O}(1)}$ even if we are given a path decomposition of width pw , unless SETH fails.

Proof (Sketch). For a given $X\text{-ANTIFACTOR}^{\mathcal{R}}$ instance we apply Lemma 5.3 with $X = Y$ to obtain the $X\text{-ANTIFACTOR}$. When applying the fast algorithm to it, one can easily check that this would contradict SETH by Lemma 5.2. ◀

The following theorem extends the previous result to the more general ANTIFACTOR_x and shows a more informative lower bound.

► **Theorem 5.5 (Lower Bound for Decision Version II).** *For all $x \geq 3, \epsilon > 0$, ANTIFACTOR_x cannot be solved in time $(x + 1 - \epsilon)^{\text{pw}} n^{\mathcal{O}(1)}$ on graphs given with a path decomposition of width pw , unless SETH fails.*

Proof. We set $Y := \{2, 4, \dots, 2x\}$ and $X := \{0, 2, 3\}$. By Lemma 4.2, Y contains a half-induced matching of size $x + 1$. Moreover, \overline{X} contains a gap of size two.

We use Lemma 5.3 to transform a $Y\text{-ANTIFACTOR}^{\mathcal{R}}$ instance into an $(X, Y)\text{-ANTIFACTOR}$ instance. From $|X|, |Y| \leq x$ and by the properties of X and Y , the claim follows directly. ◀

5.1 Replacing Finite Sets by Cofinite Sets

In this section we prove Lemma 5.2, i.e. the lower bound for $X\text{-ANTIFACTOR}^{\mathcal{R}}$, based on a lower bound from [30] for the following intermediate problem.

► **Definition 5.6 ($B\text{-FACTOR}^{\mathcal{R}}$ (Simplified Definition 4.1 in [30])).** *Let $B \subseteq \mathbb{N}$ be fixed of finite size. $B\text{-FACTOR WITH RELATIONS}$ ($B\text{-FACTOR}^{\mathcal{R}}$) is the variation of $X\text{-ANTIFACTOR}^{\mathcal{R}}$, cf. Definition 5.1, where the simple vertices are not labeled with set X but with set B .*

A set $\widehat{E} \subseteq E(G)$ is a solution for G if $\deg_{\widehat{E}}(v) \in B$ for all simple vertices v and the relations of the complex vertices are satisfied.

We use the following lower bound and the restrictions to the graph as a starting point for our construction.



(a) The simple vertex v before the modifications. (b) The gadget replacing vertex v .

■ **Figure 1** The transformation in the proof of Lemma 5.2. The red, orange, green, and blue edges represent the left-external, left-internal, right-internal, and right-external edges, respectively.

► **Lemma 5.7** (Corollaries 4.7 and 4.8 in the full version of [30]). *Let $B \subseteq \mathbb{N}$ be a fixed and finite set. Given a B -FACTOR $^{\mathcal{R}}$ instance*

- *and its path decomposition of width pw with $\Delta^* = \max_{\text{bag } B} \sum_{v \in B \cap V_C} \deg(v)$,*
- *moreover all simple vertices are only connected to 2 complex nodes by exactly $\max B$ (parallel) edges each,*
- *and we are given the promise that with respect to any solution the degree of the simple vertices is exactly $\max B$.*

Assume B -FACTOR $^{\mathcal{R}}$ can be solved in such a case in $(\max B + 1 - \epsilon)^{\text{pw} + f_B(\Delta^)} n^{\mathcal{O}(1)}$ time for some $\epsilon > 0$ and some function $f_B : \mathbb{N} \rightarrow \mathbb{R}^+$ that may depend on the set B . Then SETH fails. Moreover the result also holds for $\#B$ -FACTOR $^{\mathcal{R}}$ and $\#SETH$.*

To show a lower bound for X -ANTI-FACTOR $^{\mathcal{R}}$, it suffices to replace the simple vertices with set B by an appropriate gadgets consisting of simple vertices with set X and complex vertices.

Modification of the Graph. Let a_0, \dots, a_{h-1} and b_0, \dots, b_{h-1} be the labels of the half-induced matching of size h of X and U be the maximum over these labels. Let H be a B -FACTOR $^{\mathcal{R}}$ instance as stated in Lemma 5.7 with $\max B = h - 1$.⁴ We replace each simple vertex by the following gadget and keep the other vertices unchanged (see Figure 1).

By assumption, each simple vertex v is incident to $2(h - 1)$ edges which we can partition into two sets of size $h - 1$ depending on their endpoints. We call these groups of edges the left-external and right-external edges. We remove v and connect the left-external edges to a new complex vertex v_{left} with relation R_{left} . The right-external edges are connected similarly to another new complex vertex v_{right} with relation R_{right} . As a last step we create a new simple vertex v' with set X . We connect v' by U (parallel) edges to v_{left} , call these edges the left-internal edges, and by U parallel edges to v_{right} , call these edges the right-internal edges.

The relation R_{left} accepts if and only if, for some $i \in [0, h - 1]$, exactly i left-external and exactly a_{h-1-i} left-internal edges are selected. Similarly, R_{right} accepts if and only if, for some $j \in [0, h - 1]$, exactly b_j right-internal and exactly j right-external edges are selected.

We claim that the above replacement does not change the existence of solutions. For this we show that the number of selected left-external edges plus the number of selected right-external edges is at most $h - 1$ for each such modification. Then, by the properties in Lemma 5.7, they sum to exactly $h - 1$ selected edges.

If i left-external edges are selected, then v' is incident to a_{h-1-i} selected left-internal edges, by definition of R_{left} . As R_{right} rejects when v_{right} is incident to exactly k selected right-internal edges where $k \neq b_j$ for all j , vertex v' must be incident to b_j right-internal edges for some j . By the definition of the half-induced matching we get $a_{h-1-i} + b_j \in X$ if

⁴ It actually suffices to set $B = \{h - 1\}$.

$j > h - 1 - i$. Thus, some $b_{h-1-i-i'}$ with $h - 1 - i \geq i' \geq 0$ must be chosen. The relation R_{right} maps the $b_{h-1-i-i'}$ selected right-internal edges to $h - 1 - i - i'$ selected right-external edges. Thus, the gadget is incident to $i + (h - 1 - i - i') = h - 1 - i' \leq h - 1$ edges in total.

As v was only adjacent to complex vertices, we can merge the complex vertices v_{left} and v_{right} with the existing complex vertices and thus also the corresponding relations.

We analyze how the size and the pathwidth change. Replacing the simple vertices by the gadget does not change the pathwidth of the graph but only increases the degree of the complex vertices (due to the merging of the relations). Hence, Δ^* increases to $\Delta^* \cdot U$. As U only depends on the set X , it can be bounded by $\hat{f}(\max X)$ for some function \hat{f} .

Proof of Lemma 5.2 (Sketch). For a given B -FACTOR $^{\mathcal{R}}$ instance with $B = \{h - 1\}$, apply the above construction to obtain an X -ANTIFACTOR $^{\mathcal{R}}$ instance G . The total degree increases only by a factor depending on X , which is a constant as X is fixed. When running the claimed algorithm on G we contradict SETH by Lemma 5.7. \blacktriangleleft

6 Lower Bounds for the Counting Version

In this section we prove the two lower bounds for the counting version. While the lower bound for the decision and maximization version of X -ANTIFACTOR rely on half-induced matching, we avoid this dependence for $\#X$ -ANTIFACTOR by using interpolation techniques. This allows us to show a tight lower bound compared to the running time of the algorithm from Theorem 1.3. For the case when $X = \{0\}$, that is $\#$ EDGECOVER, we show a completely independent but also tight lower bound in the full version of this paper.

We also parameterize by the size of the set, i.e. by x . We design a new construction to prove the $\#W[1]$ -hardness of $\#$ ANTIFACTOR $_x$, even if $x = 1$, when parameterizing by treewidth. Hence, the problem is most likely not fixed-parameter tractable.

Both bounds use the same two-step approach as for the decision and optimization version; we first show the hardness of an intermediate problem which uses arbitrary relations and then remove these relations by a chain of reductions to obtain the actual lower bounds.

Parameterizing by the Maximum of the Set. We first show a lower bound for the intermediate problem $\#X$ -ANTIFACTOR $^{\mathcal{R}}$, which is the counting version of X -ANTIFACTOR $^{\mathcal{R}}$.

► **Lemma 6.1** (Lower Bound for $\#X$ -ANTIFACTOR $^{\mathcal{R}}$). *Let $X \subseteq \mathbb{N}$ be a fixed, non-empty and finite set. Let $f_X : \mathbb{N} \rightarrow \mathbb{R}^+$ be an arbitrary function that may depend on the set X .*

For every constant $\epsilon > 0$, there is no algorithm that can solve $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ in time $(\max X + 2 - \epsilon)^{\text{pw} + f_X(\Delta^)} n^{\mathcal{O}(1)}$, where $\Delta^* = \max_{\text{bag } B} \sum_{v \in B \cap V_C} \deg(v)$, even if we are given a path decomposition of width pw , unless $\#$ SETH fails.*

We make use of the following lemma to remove the relations. We extend the definition of (X, Y) -ANTIFACTOR in the natural way to the counting version $\#(X, Y)$ -ANTIFACTOR.

► **Lemma 6.2.** *Let $X \subseteq \mathbb{N}$ be a finite set such that $X \not\subseteq \{0\}$. Let $Y \subseteq \mathbb{N}$ be arbitrary (possibly be given as input).*

There is a Turing reduction from $\#Y$ -ANTIFACTOR $^{\mathcal{R}}$ to $\#(X, Y)$ -ANTIFACTOR increasing the size from n to $n \cdot f(\max X)$, decreasing Δ^ to zero, and increasing pw to $\text{pw} + \Delta^* \cdot f(\max X)$.*

As for the decision version, the proof of this lemma is based on the reductions in [30] for the counting version of B -FACTOR. The reduction makes use of the Holant framework [8, 9, 20, 22, 24, 28] to formally relate the different steps of the reduction. See Appendix B for the main ideas behind the proof.

Combining these two lemmas, we can prove the first lower bound for the counting version.

► **Theorem 6.3** (Lower Bound for Counting Version I). *Let $X \subseteq \mathbb{N}$ be a finite and fixed set such that $X \not\subseteq \{0\}$. For every constant $\epsilon > 0$, there is no algorithm that can solve $\#X$ -ANTIFACTOR in time $(\max X + 2 - \epsilon)^{\text{pw}} n^{\mathcal{O}(1)}$ even if we are given a path decomposition of width pw , unless $\#SETH$ fails.*

Proof (Sketch). After applying Lemma 6.2 where we set $X = Y$, the algorithm directly contradicts $\#SETH$ by Lemma 6.1. ◀

Parameterizing by the Size of the Set. If we do not fix the set X but only the *size* of the set, the decision and optimization version of ANTIFACTOR_x are still FPT parameterized by treewidth. For $\#ANTIFACTOR_x$ the following result conditionally rules out such algorithms.

► **Lemma 6.4.** *There exists a constant c such that there is no $\mathcal{O}(n^{p-c})$ algorithm for $\#ANTIFACTOR_1^{\mathcal{R}}$ on n -vertex graphs with $\Delta^* \in \mathcal{O}(1)$, even if only one set is used and we are given a path decomposition of width p , unless $\#SETH$ is false.*

Combined with Lemma 6.2 to remove the relations, we get the following hardness result.

► **Theorem 6.5** (Lower Bound for Counting Version II). *There exists a constant c such that there is no $\mathcal{O}(n^{p-c})$ algorithm for $\#ANTIFACTOR_1$ on n -vertex graphs, even if we are given a path decomposition of width p , unless $\#SETH$ is false.*

Proof. Let G be a given $\#ANTIFACTOR_1^{\mathcal{R}}$ instance which uses just one set $Y \subseteq \mathbb{N}$ with $|Y| = 1$. Use Lemma 6.2 with $X = \{2\}$ to transform G into a $\#ANTIFACTOR_1$ instance H . The remaining part of the proof follows in a standard manner by using Lemma 6.4. ◀

We prove Lemma 6.4 by a reduction from the $\#W[1]$ -hard problem COUNTING COLORFUL HITTING k -SETS. Hence, the following result holds by applying Lemma 6.2 as before.

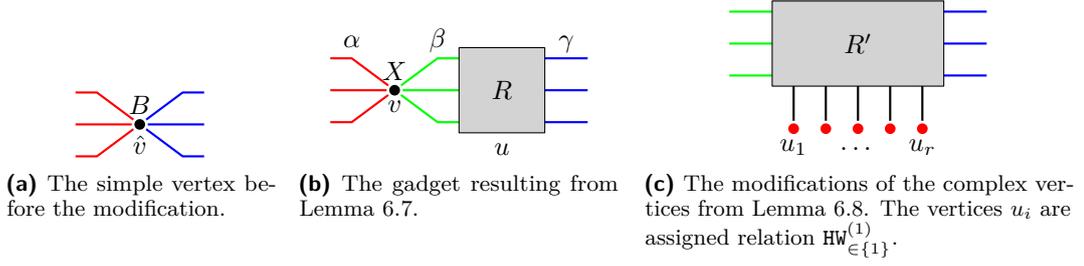
► **Theorem 6.6.** *$\#ANTIFACTOR_1$ is $\#W[1]$ -hard.*

6.1 High-level Construction for SETH Lower Bound

We show the hardness of $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ by a reduction from $\#B$ -FACTOR $^{\mathcal{R}}$, that is, we prove Lemma 6.1. As for the decision and optimization version, we mainly have to modify the simple vertices to take care of the new set. We design this reduction in three steps.

- The first step is a lower bound for the *relation-weighted* version of $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ (cf. Lemma 6.7). For this problem we assign each accepted input of a relation a weight. Then an accepted input contributes by this weight to the solution. The weight of the solution is the product of the weights of the relations. The unweighted problem can be seen as assigning weight 1 to all accepted inputs and weight 0 to all rejected inputs.
- Then in a second step we reduce this problem to the *edge-weighted* version where the edges are assigned weights by which they contribute to the solution if they are selected (cf. Lemma 6.8). Again, the weight of a solution is the product of the weights of the selected edges.
- The last step then removes these edge-weights by an appropriate Turing-reduction using the technique of interpolation (cf. Lemma 6.9).

We use the Holant framework (cf. [8, 9, 20, 22, 24, 28]) to contextualize several versions of $\#ANTIFACTOR$. In the Holant framework, we are given a signature graph $\Omega = (V, E)$, where every edge $e \in E$ has a weight w_e . Every vertex $v \in V$ is labeled with a signature



■ **Figure 2** The modifications of the vertices in the different steps of the reductions.

$f_v : \{0, 1\}^{I(v)} \rightarrow \mathbb{Q}$, where $I(v)$ is the incidence vector of edges incident to v . A *solution* is a subset of edges such that the signature for each vertex is non-zero. The weight of a solution is the product of the weights of all edges in the subset and the signatures of all the vertices. Then the Holant of Ω is defined as the sum of the weights of all solutions.

$$\text{Holant}(\Omega) = \sum_{x \in \{0,1\}^{E(\Omega)}} \prod_{e \in x} w_e \prod_{v \in V(\Omega)} f_v(x|_{I(v)}).$$

The Holant problem is easily seen to be a weighted generalization of the counting versions of GENFAC and ANTIFACTOR. For example, the problem $\#X$ -ANTIFACTOR is a Holant problem on unweighted graphs where every vertex has the following symmetric relation.

$$f(z) = \begin{cases} 1 & \text{if } \text{hw}(z) \notin X \\ 0 & \text{if } \text{hw}(z) \in X \end{cases}$$

where $\text{hw}(\cdot)$ is the Hamming weight operator. We call vertices with such functions to be $\text{HW}_{\overline{X}}$ nodes.

For relations R_1, \dots, R_k , we define $\text{Holant}(R_1, \dots, R_k)$ to be the set of Holant problems where every edge is unweighted and every vertex has signature R_j for some $j \in [k]$. By an abuse of notation also let R_j be a *family* of relations. For example, we may use $\text{Holant}(\text{HW}_{=1})$ when every vertex has relation $\text{HW}_{=1}^{(k)}$ for some k .

The relation-weighted version of $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ corresponds to a variant of the Holant problem where all edges have weight 1. Likewise the edge-weighted version of the $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ problem corresponds to a variant of the Holant problem where we require that the value of the signatures is either 0 or 1, i.e. they accept or reject.

The Holant framework was extensively used in proving the $\#$ SETH lower bounds for COUNTING PERFECT MATCHINGS [11] and COUNTING GENERAL FACTORS [30]. We make use of some of their constructions to prove the following lower bound.

► **Lemma 6.7.** *Let $X \subseteq \mathbb{N}$ be a fixed, non-empty and finite set. Let $f_X : \mathbb{N} \rightarrow \mathbb{R}^+$ be an arbitrary function that may depend on the set X .*

For every constant $\epsilon > 0$, there is no algorithm that can, even if we are given a path decomposition of width pw , solve relation-weighted $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ with $(\max X + 2)^2$ weights in time $(\max X + 2 - \epsilon)^{\text{pw} + f_X(\Delta^)} n^{\mathcal{O}(1)}$, where $\Delta^* = \max_{\text{bag } B} \sum_{v \in B \cap V_C} \deg(v)$, unless $\#$ SETH fails.*

We prove the lower bound by a reduction from $\#B$ -FACTOR $^{\mathcal{R}}$ where $B = \{\max X + 1\}$. When treating the given $\#B$ -FACTOR $^{\mathcal{R}}$ instance G as a $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ instance H , all solution of G are also a solution for H because of our choice of B . The converse is not true:

As X is finite (and thus the set of allowed degree is cofinite), the degree of the solutions for H can be different from $\max X + 1$ (possibly larger or even smaller). We construct a gadget such that the number of selected incident edges is equal to $\max X + 1$ and the solution is also valid for the B -FACTOR $^{\mathcal{R}}$ instance.

For this we add a weighted relation directly after the simple vertices. We choose the weights such that the entire gadget behaves as the original vertex with set B . For this we exploit the fact that the set X does not allow certain combinations of selected incident edges.

Proof of Lemma 6.7. We start with a $\#B$ -FACTOR $^{\mathcal{R}}$ instance H as stated in Lemma 5.7 where $B = \{\max X + 1\}$ and apply the following transformation, illustrated in Figures 2a and 2b, for each simple vertex \hat{v} .

Transformation. By assumption, the incident edges of \hat{v} can be split into *left* and *right* edges (depending on their endpoints). We remove \hat{v} and create a simple vertex v and a complex vertex u . Connect the left edges to v and the right edges to u . We connect v and u by $\max X + 1$ parallel edges which we call *middle* edges.⁵ We assign the set X to v and the relation R to u , which is defined as follows.

R requires that the selection of middle and right edges is monotone. That is the first k edges are selected and the last $\max X + 1 - k$ edges are not selected. Then for all $\beta, \gamma \in [0, \max X + 1]$, R accepts β middle and γ right edges with weight $w_{\beta, \gamma}$.

To define the weights $w_{\beta, \gamma}$, let $g(\alpha, \gamma)$ denote the signature of the whole gadget (including v and u) when α left and γ right edges are selected.⁶ Based on our construction we get:

$$g(\alpha, \gamma) = \sum_{\substack{\beta=0 \\ \beta+\alpha \notin X}}^{\max X+1} w_{\beta, \gamma}.$$

To simulate the original simple vertex \hat{v} with set $B = \{\max X + 1\}$, we need $g(i, \max X + 1 - i) = 1$ for all $i \in [0, \max X + 1]$ and 0 otherwise. These constraints and the definition of g describe a system of linear equations with $(\max X + 2)^2$ variables and equally many constraints. Assuming that there always exists a solution, we can use Gaussian elimination to compute the solution in time $\mathcal{O}(n^3)$. Since the weights are chosen such that the remaining graph does not see a difference between the vertex \hat{v} and this new gadget, we can replace all simple vertices by this procedure to get a relation-weighted $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ instance.

It remains to prove that such a solution always exists. For a fixed γ the values of $g(0, \gamma), \dots, g(\max X + 1, \gamma)$ depend only on the weights $w_{0, \gamma}, \dots, w_{\max X + 1, \gamma}$. Moreover, these weights do not appear in the sum of any $g(i, \gamma')$ where $\gamma' \neq \gamma$. Hence, we can treat each possible value of γ separately.

For a fixed γ , the sums for $g(\alpha, \gamma)$ and $g(\alpha + 1, \gamma)$ differ by (at least) one summand, i.e. $w_{\max X - \alpha, \gamma}$. When starting with the sums for $\max X + 1$ and $\max X$, we can eliminate $w_{0, \gamma}$ from the system of linear equations. Then we can repeat this process to eliminate $w_{1, \gamma}$ up to $w_{\max X + 1, \gamma}$. Hence, there is a solution for a fixed γ .

Lower Bound. Let G be the final $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ instance. We have $n_G \leq 2n_H$, $\Delta_G^* \leq 2\Delta_H^*$, and $\text{pw}_G \leq \text{pw}_H + \Delta_H^*$. Assume we run a fast algorithm for relation-weighted $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ on the new instance. Then, one can easily check that this directly contradicts $\#\text{SETH}$ by Lemma 5.7. \blacktriangleleft

⁵ Though these parallel edges disappear later, one could place EQ_2 nodes on them to obtain a simple graph.

⁶ Note that a signature is normally defined on subsets of edges. As we require the selection of the edges to be monotonous, we also use “signature” to refer to g .

The next step of our reduction removes the weights from the relations by using weighted edges.

► **Lemma 6.8.** *Let $X \subseteq \mathbb{N}$ be an arbitrary set (possibly given as input).*

We can many-one reduce relation-weighted $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ with r different weights to edge-weighted $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ such that

- *the r weights do not change,*
- *the size increases by a multiplicative factor of $\mathcal{O}(r)$,*
- *Δ^* increases to $\Delta^* + r + 1$,*
- *and pw increases to $\text{pw} + 1$.*

Proof. We apply the following procedure to each complex vertex u . See Figure 2c for the modification. For this fix some u and let R be its relation. Let $w_1, \dots, w_{r'}$ be the $r' < r$ different weights used by R . Assume w.l.o.g. that $r' = r$.

For all $i \in [r]$, we add a vertex u_i with relation $\text{HW}_{\{0,1\}}^{(1)}$ and make it adjacent to v by an edge of weight w_i .

Based on R we design a new relation R' as follows: Whenever R accepts the input x with weight w_i for some $i \in [r]$, then R' accepts x but additionally requires that the edge to u_i is selected while the edges to the other $u_{i'}$ remain unselected.

One can easily check that this modification does not change the solution. Moreover, the pathwidth increases by at most 1 and the degree of the complex nodes by at most r . ◀

In the next step of our reduction we remove the edge weights from the graph. First observe that we do not have to change edges of weight 1. Furthermore, we can simply remove all edges with weight 0.

► **Lemma 6.9.** *Let $X \subseteq \mathbb{N}$ be a finite and non-empty set (possibly given as input). There is a Turing reduction from edge-weighted $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ with r different weights to unweighted $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ running in time $n^{\mathcal{O}(r)}$. The reduction is such that*

- *the size increases by a multiplicative factor of $\mathcal{O}(\log^2(n))$,*
- *the degree of the simple vertices stays the same,*
- *Δ^* increases to $\Delta^* + \mathcal{O}(1)$,*
- *and pw increases to $\text{pw} + \mathcal{O}(1)$.*

The proof of the lemma uses the same interpolation techniques already used in [11] and later in [30] to remove the edge weights. Now we can combine the previous steps and prove the lower bound for $\#X$ -ANTIFACTOR $^{\mathcal{R}}$.

Proof of Lemma 6.1 (Sketch). For a given relation-weighted $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ instance H we apply Lemmas 6.8 and 6.9 to obtain polynomially many $\#X$ -ANTIFACTOR instances. It is easy to check that a faster algorithm for $\#X$ -ANTIFACTOR $^{\mathcal{R}}$ would contradict $\#\text{SETH}$ by Lemma 6.7. ◀

References

- 1 Akanksha Agrawal, Pallavi Jain, Lawqueen Kanesh, and Saket Saurabh. Parameterized complexity of conflict-free matchings and paths. *Algorithmica*, 82(7):1939–1965, 2020. doi:10.1007/s00453-020-00681-y.
- 2 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.

- 3 Hans L Bodlaender. Dynamic programming on graphs with bounded treewidth. In *International Colloquium on Automata, Languages, and Programming*, pages 105–118. Springer, 1988.
- 4 Hans L Bodlaender. Treewidth: Algorithmic techniques and results. In *International Symposium on Mathematical Foundations of Computer Science*, pages 19–36. Springer, 1997.
- 5 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.
- 6 Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008. doi:10.1093/comjnl/bxm037.
- 7 Édouard Bonnet, Nick Brettell, O-joung Kwon, and Dániel Marx. Generalized feedback vertex set problems on bounded-treewidth graphs: Chordality is the key to single-exponential parameterized algorithms. *Algorithmica*, 81(10):3890–3935, 2019. doi:10.1007/s00453-019-00579-4.
- 8 Jin-yi Cai, Sangxia Huang, and Pinyan Lu. From Holant to #CSP and back: Dichotomy for Holant^c problems. In Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, editors, *Algorithms and Computation – 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I*, volume 6506 of *Lecture Notes in Computer Science*, pages 253–265. Springer, 2010. doi:10.1007/978-3-642-17517-6_24.
- 9 Jin-yi Cai, Pinyan Lu, and Mingji Xia. A computational proof of complexity of some restricted counting problems. *Theor. Comput. Sci.*, 412(23):2468–2485, 2011. doi:10.1016/j.tcs.2010.10.039.
- 10 Gérard Cornuéjols. General factors of graphs. *J. Comb. Theory, Ser. B*, 45(2):185–198, 1988. doi:10.1016/0095-8956(88)90068-8.
- 11 Radu Curticapean and Dániel Marx. Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1650–1669. SIAM, 2016. doi:10.1137/1.9781611974331.ch113.
- 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 Víctor Dalmau and Daniel K. Ford. Generalized satisfiability with limited occurrences per variable: A study through delta-matroid parity. In Branislav Rován and Peter Vojtás, editors, *Mathematical Foundations of Computer Science 2003, 28th International Symposium, MFCS 2003, Bratislava, Slovakia, August 25-29, 2003, Proceedings*, volume 2747 of *Lecture Notes in Computer Science*, pages 358–367. Springer, 2003. doi:10.1007/978-3-540-45138-9_30.
- 14 Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlen. Exponential time complexity of the permanent and the Tutte polynomial. *ACM Trans. Algorithms*, 10(4):21:1–21:32, 2014. doi:10.1145/2635812.
- 15 Szymon Dudycz and Katarzyna Paluch. Optimal general matchings. In Andreas Brandstädt, Ekkehard Köhler, and Klaus Meer, editors, *Graph-Theoretic Concepts in Computer Science – 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, volume 11159 of *Lecture Notes in Computer Science*, pages 176–189. Springer, 2018. Full version: arXiv:1706.07418. doi:10.1007/978-3-030-00256-5_15.
- 16 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.
- 17 Eduard Eiben and Iyad Kanj. A colored path problem and its applications. *ACM Trans. Algorithms*, 16(4):47:1–47:48, 2020. doi:10.1145/3396573.
- 18 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016. doi:10.1145/2886094.

- 19 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Representative families of product families. *ACM Trans. Algorithms*, 13(3):36:1–36:29, 2017. doi:10.1145/3039243.
- 20 Heng Guo and Pinyan Lu. On the complexity of holant problems. In Andrei A. Krokhin and Stanislav Zivný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 159–177. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/DFU.Vol7.15301.6.
- 21 John E Hopcroft and Richard M Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.
- 22 Sangxia Huang and Pinyan Lu. A dichotomy for real weighted holant problems. *Comput. Complex.*, 25(1):255–304, 2016. doi:10.1007/s00037-015-0118-3.
- 23 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.
- 24 Michael Kowalczyk and Jin-Yi Cai. Holant problems for 3-regular graphs with complex edge functions. *Theory Comput. Syst.*, 59(1):133–158, 2016. doi:10.1007/s00224-016-9671-7.
- 25 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.
- 26 L. Lovász and M. D. Plummer. *Matching Theory*. North-Holland Publishing Co., Amsterdam, 1986. Annals of Discrete Mathematics, 29.
- 27 László Lovász. The factorization of graphs. II. *Acta Mathematica Hungarica*, 23(1-2):223–246, 1972.
- 28 Pinyan Lu. Complexity dichotomies of counting problems. *Electron. Colloquium Comput. Complex.*, 18:93, 2011. URL: <http://eccc.hpi-web.de/report/2011/093>.
- 29 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Anti-factor is FPT parameterized by treewidth and list size (but counting is hard). *CoRR*, abs/2110.09369, 2021. arXiv:2110.09369.
- 30 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Degrees and gaps: Tight complexity results of general factor problems parameterized by treewidth and cutwidth. 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 95:1–95:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. Full version: arXiv:2105.08980. doi:10.4230/LIPICs.ICALP.2021.95.
- 31 Dániel Marx and Paul Wollan. An exact characterization of tractable demand patterns for maximum disjoint path problems. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 642–661. SIAM, 2015. doi:10.1137/1.9781611973730.44.
- 32 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|} |E|)$ algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.12.
- 33 Burkhard Monien. How to find long paths efficiently. In *Analysis and design of algorithms for combinatorial problems (Udine, 1982)*, volume 109 of *North-Holland Math. Stud.*, pages 239–254. North-Holland, Amsterdam, 1985.
- 34 András Sebő. General antifactors of graphs. *J. Comb. Theory, Ser. B*, 58(2):174–184, 1993.
- 35 Hadas Shachnai and Meirav Zehavi. Representative families: A unified tradeoff-based approach. *J. Comput. Syst. Sci.*, 82(3):488–502, 2016. doi:10.1016/j.jcss.2015.11.008.
- 36 Johan M. M. van Rooij. Fast algorithms for join operations on tree decompositions. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms – Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 262–297. Springer, 2020. doi:10.1007/978-3-030-42071-0_18.

A

 Omitted Proofs from Section 4

We first prove that arithmetic progressions in the set of excluded degrees are sufficient to obtain large half-induced matchings in the corresponding compatibility graph.

Proof of Lemma 4.2. Let $a, a + d, a + 2d, \dots, a + (\ell - 1)d \in X$ be an arithmetic progression with $d \geq 1$ such that $a + \ell d \notin X$. We construct the following half-induced matching in \mathcal{C}_X where, for all $i \in [\ell + 1]$, we set $a_i := d(i - 1)$ and $b_i := a + (\ell + 1 - i)d$.

Then for all $i \in [\ell + 1]$ we have $a_i + b_i = d(i - 1) + a + (\ell + 1 - i)d = a + \ell d \notin X$ and hence $(a_i, b_i) \in E(\mathcal{C}_X)$. Similarly, for all $i \in [\ell]$ and all $i < j \in [\ell + 1]$, we have $(a_i, b_j) \notin E(\mathcal{C}_X)$ because $a_i + b_j = d(i - 1) + a + (\ell + 1 - j)d = a + (\ell + i - j)d \in X$. \blacktriangleleft

Now we show the converse of the above result. That is, arithmetic progressions are necessary to obtain large half-induced matchings.

Proof of Lemma 4.3. Let $a_1, \dots, a_{\ell+1}$ and $b_1, \dots, b_{\ell+1}$ be the vertices of the half-induced matching of size $\ell + 1$ in \mathcal{C}_X . Then we have the following constraints:

$$\begin{array}{llll} a_1 + b_2 \in X, & a_1 + b_3 \in X, & \dots, & a_1 + b_{\ell+1} \in X \\ & a_2 + b_3 \in X, & \dots, & a_2 + b_{\ell+1} \in X \end{array}$$

Let $X = \{x_1, x_2, \dots, x_\ell\}$ where $x_i \leq x_{i+1}$. From $a_1 + b_j \neq a_1 + b_{j'}$ for any $j \neq j'$, we get

$$\{a_1 + b_2, a_1 + b_3, \dots, a_1 + b_{\ell+1}\} = X.$$

Now consider the second set of constraints. Assuming $a_2 - a_1 = d > 0$, we have

$$\{a_2 + b_3, a_2 + b_4, \dots, a_2 + b_{\ell+1}\} = X \setminus \{x_i\}$$

for some i . Since $d > 0$, we get $x_\ell + d \notin X$ and thus

$$\{x_1 + d, x_2 + d, x_3 + d, \dots, x_{\ell-1} + d\} = X \setminus \{x_i\}.$$

Now further observe that x_1 cannot belong to the left-hand side because $d > 0$ and $x_1 = \min(X)$. Thus, we have that $i = 1$. Similarly, when $a_2 - a_1 = d < 0$ we can argue that $i = \ell$. Without loss of generality consider the former case. Then we have that $x_i + d = x_{i+1}$ for all $i \in [\ell]$. Hence, X is an arithmetic progression of length ℓ . \blacktriangleleft

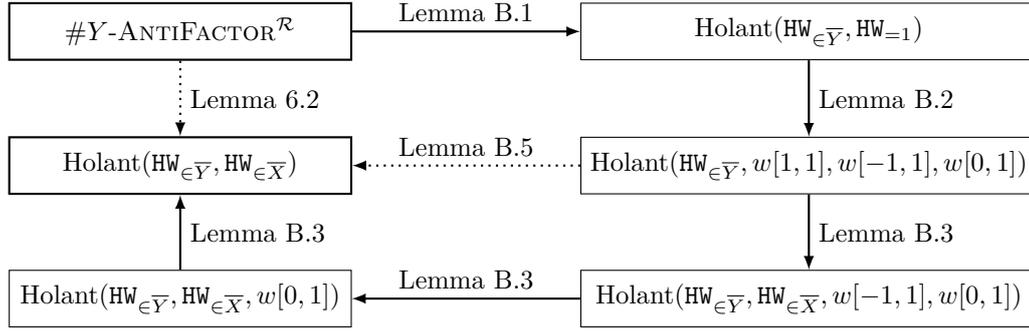
As the last part of this section, we prove Lemma 4.4 which states that a half-induced matching implies a lower bound for the size of the representative set.

Proof of Lemma 4.4. We set the value of k later. Let the half-induced matching be between $A, B \subseteq \mathbb{N}$ with $A = \{a_1, \dots, a_\ell\}$, $B = \{b_1, \dots, b_\ell\}$. Define indexing functions ind_A and ind_B such that $\text{ind}_A(a_i) = \text{ind}_B(b_i) = i$. For $s \in A^k$, define $\text{ind}_A(s) = \sum_{i \in [k]} \text{ind}_A(s[i])$. We partition A^k into sets \mathcal{S}_q with $q \in [\ell \cdot k]$ such that

$$\mathcal{S}_q = \{s \in A^k \mid \text{ind}_A(s) = q\}.$$

Hence, there exists some $q' \in [\ell \cdot k]$ such that

$$|\mathcal{S}_{q'}| \geq \frac{\ell^k}{\ell \cdot k}.$$



■ **Figure 3** Steps in the chain of reductions from $\#Y\text{-ANTIFACTOR}^{\mathcal{R}}$ to $\#(X, Y)\text{-ANTIFACTOR}$, i.e. $\text{Holant}(\text{HW}_{\in \bar{Y}}, \text{HW}_{\in \bar{X}})$. Dotted lines indicate results obtained by combined reductions.

Let $\mathcal{S} = \mathcal{S}_{q'}$ be the set for which we want to lower bound the size of its representative sets. To simplify notation, let $q = q'$. Consider some $s \in \mathcal{S}$. We claim that s is the unique compatible element for $t \in B^k$, where

$$t[i] = b_{\text{ind}_A(s[i])}.$$

It is clear that $s \sim_{\mathcal{C}_X}^k t$. Suppose there is some other $s' \in \mathcal{S}$ such that $s' \sim_{\mathcal{C}_X}^k t$. Then since $\text{ind}_A(s) = \text{ind}_A(s') = q$ and since $s \neq s'$, there is some index j such that $\text{ind}_A(s[j]) > \text{ind}_A(s'[j])$. The j th index of $s' + t$ is $a_{\text{ind}_A(s'[j])} + b_{\text{ind}_A(s[j])}$ because $s'[j] = a_{\text{ind}_A(s'[j])}$. However, observe that this sum must be in X from the fact that there is a half-induced matching between A, B in \mathcal{C}_X and $\text{ind}_A(s[j]) > \text{ind}_A(s'[j])$. This is a contradiction, implying that s is the only compatible partner of t . Thus, s is forced to belong to any representative set $\mathcal{S}' \subseteq_{\mathcal{C}_X\text{-rep}}^k \mathcal{S}$.

Since the above argument holds for all $s \in \mathcal{S}$, we conclude that the only representative set for \mathcal{S} is itself. Now we set k to be large enough such that $k \log(\ell - \epsilon) \leq k \log(\ell) - \log(\ell \cdot k)$. Then we have

$$|\mathcal{S}| \geq \frac{\ell^k}{\ell \cdot k} \geq (\ell - \epsilon)^k. \quad \blacktriangleleft$$

B Proof of Lemma 6.2: Removing Relations

In this section, we prove the reduction from $\#Y\text{-ANTIFACTOR}^{\mathcal{R}}$ to $\#(X, Y)\text{-ANTIFACTOR}$, i.e. Lemma 6.2, by a chain of reductions (cf. Figure 3). We make use of the Holant framework, which was also used in [30], to formally state the results. The first step uses Lemmas 7.5 and 7.6 from [30]. Observe for this that the lemmas work for $\#B\text{-FACTOR}$ even when B is co-finite, that is $\#\bar{B}\text{-ANTIFACTOR}$ because the simple vertices of the instance are not changed in any way.

► **Lemma B.1** (Lemma 7.5 and 7.6 in [30]). *There is a polynomial-time Turing reduction from $\#X\text{-ANTIFACTOR}^{\mathcal{R}}$ to $\text{Holant}(\text{HW}_{\in \bar{X}}, \text{HW}_{=1})$ such that the maximum degree increases to at least 6 and the pathwidth increases by at most a constant depending only on Δ^* , i.e. the maximum total degree of the complex nodes in any bag of the path decomposition.*

► **Note.** Observe that Lemma 7.5 in [30] requires that the relation is even, i.e. the Hamming weight of every accepted input is even. We can easily make every relation even by adding an additional input that is selected whenever the parity of the original input is odd. This additional input is then connected to a EQ_1 node, which can easily be realized by forcing $\max X + 1$ edges to a fresh vertex using $\text{HW}_{=1}^{(1)}$ nodes.

Before proceeding with the next steps, we define, for all $x, y \in \mathbb{Z}$, $w[x, y]$ as a new type of node which has one dangling edge e and the following signature:

$$f(e) = \begin{cases} x & \text{if } e \text{ is not selected} \\ y & \text{if } e \text{ is selected} \end{cases}.$$

Observe that $\text{HW}_{=1}^{(1)}$ is precisely $w[0, 1]$ and $\text{HW}_{\in\{0,1\}}^{(1)}$ corresponds to $w[1, 1]$. In the following constructions we additionally use a $w[-1, 1]$ node. We use the $w[x, y]$ notation in the following wherever possible.

► **Lemma B.2.** *Let $X \subseteq \mathbb{N}$ be a finite set such that $X \not\subseteq \{0\}$. Let R_1, \dots, R_d be d arbitrary relations for some $d \geq 0$. There is a polynomial-time Turing reduction from*

$$\text{Holant}(R_1, \dots, R_d, \text{HW}_{\in\overline{X}}, \text{HW}_{=1}) \text{ to } \text{Holant}(R_1, \dots, R_d, \text{HW}_{\in\overline{X}}, w[1, 1], w[-1, 1], w[0, 1])$$

such that Δ^* increases to $\Delta^* \cdot f(\max X)$ and pw increases to $\text{pw} + \Delta^* \cdot f(\max X)$.

The proof of the lemma is given in the full version [29] and uses three different gadgets depending on X to realize $\text{HW}_{=1}$. Next we show that we can realize $w[x, y]$ nodes. In particular, we can get the $w[1, 1]$, $w[-1, 1]$, and $w[0, 1]$ nodes introduced by Lemma B.2.

► **Lemma B.3.** *Let $X \subseteq \mathbb{N}$ be a fixed, finite set with $X \not\subseteq \{0\}$. Let R_1, \dots, R_d be d arbitrary relations for some $d \geq 0$. The following holds for arbitrary values x, y . There is a polynomial-time Turing reduction from $\text{Holant}(R_1, \dots, R_d, \text{HW}_{\in\overline{X}}, w[x, y])$ to $\text{Holant}(R_1, \dots, R_d, \text{HW}_{\in\overline{X}})$ such that Δ^* decreases and pw increases to $\text{pw} + \Delta^* \cdot f(\max X)$.*

Proof. We use Lemma 7.11 from [30] as our prototype. However, some arguments from their proof do not follow in our case.

Let U be the set of $w[x, y]$ nodes in the given graph G . Let A_i denotes the number of possible solutions in G where for exactly i of the $w[x, y]$ nodes the dangling edge is not selected and for the other $|U| - i$ nodes the dangling edge is selected. Then we have

$$\text{Holant}(G) = \sum_{i=0}^{|U|} A_i x^i y^{|U|-i}. \quad (1)$$

We construct graphs G_d for a new parameter d from G where we replace each $w[x, y]$ node by a gadget H_d with exactly one dangling edge. The construction of H_d is given later as it depends on X . Let $h_0(d)$ denote the number of solutions for H_d when the dangling edge is not selected and $h_1(d)$ when the dangling edge is selected. Then we get

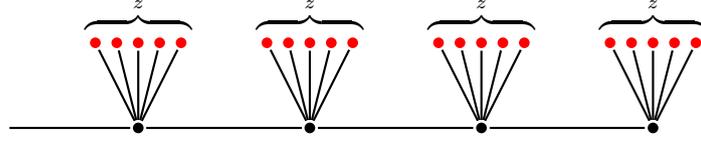
$$\text{Holant}(G_d) = \sum_{i=0}^{|U|} A_i h_0(d)^i h_1(d)^{|U|-i} = h_1(d)^{|U|} \sum_{i=0}^{|U|} A_i \left(\frac{h_0(d)}{h_1(d)} \right)^i.$$

Assume we can find at least $|U| + 1$ values for d such that for all values the ratios $h_0(d)/h_1(d)$ are pairwise different. After computing $\text{Holant}(G_d)$ for these values of d we can recover the value of each A_i . By Equation (1) we can finally output the value of $\text{Holant}(G)$.

It remains to construct the gadgets H_d and to find the values for d . We later construct the gadgets in a way such that there are constants F_1 , F_2 , and F_3 only depending on X with

$$\begin{aligned} h_0(d) &:= F_0 \cdot h_0(d-1) + F_1 \cdot h_1(d-1) & h_0(1) &:= F_0 \\ h_1(d) &:= F_1 \cdot h_0(d-1) + F_2 \cdot h_1(d-1) & h_1(1) &:= F_1. \end{aligned}$$

Given these properties of H_d , we can use the following proposition to find sufficiently many values for d . The proof is given in the full version [29].



■ **Figure 4** Gadget to realize $w[x, y]$ nodes in Case 1. Red nodes are $\text{HW}_{\in\{0,1\}}^{(1)}$ nodes.

► **Proposition B.4** (Special Case of Proposition 7.7 in [30]). *Given three constants $F_0, F_1,$ and F_2 with $F_0F_2 \neq (F_1)^2$ and $F_0, F_1 \neq 0$. Let $\{A_n\}_{n \in \mathbb{N}}, \{B_n\}_{n \in \mathbb{N}}$ be two sequences with*

$$\begin{bmatrix} A_n \\ B_n \end{bmatrix} = M \cdot \begin{bmatrix} A_{n-1} \\ B_{n-1} \end{bmatrix} = M^n \cdot U \quad \text{where} \quad M = \begin{bmatrix} F_0 & F_1 \\ F_1 & F_2 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} A_0 \\ B_0 \end{bmatrix} = \begin{bmatrix} F_0 \\ F_1 \end{bmatrix}.$$

Then $\{A_n/B_n\}_{n \in \mathbb{N}}$ is a sequence which does not contain any repetitions.

As a last step we construct the H_d gadgets.

Case 1: $0 \in X$ or $1 \notin X$. We first show how to get a $\text{HW}_{\in\{0,1\}}^{(1)}$ node.

- If $0, 1 \notin X$, then any vertex with a dangling edge acts as a $\text{HW}_{\in\{0,1\}}^{(1)}$ node.
- If $0 \in X, 1 \notin X$, then attach $\max X + 1$ pendant vertices to any vertex v . Then v acts as a $\text{HW}_{\in\{0,1\}}^{(1)}$ node.
- If $0 \in X, 1 \in X$, then take a clique of size $\min(\bar{X}) + 1$. Split the edge between two vertices into two dangling edges. This now acts as a $\text{HW}_{=2}^{(2)}$ node. Attaching $\lceil (\max X + 1)/2 \rceil$ many $\text{HW}_{=2}^{(2)}$ nodes to a new vertex with one dangling edge gives us a $\text{HW}_{\in\{0,1\}}^{(1)}$ node.

H_d consists of a path of d vertices with a dangling edge on the first vertex. For an integer $z \geq \max X + 1$ that we will choose later, attach z pendant $\text{HW}_{\in\{0,1\}}^{(1)}$ nodes to each vertex in the path. See Figure 4 for an illustration. By this definition we get:

$$F_0 = \sum_{i \geq 0: i \in \bar{X}} \binom{z}{i}, \quad F_1 = \sum_{i \geq 0: i+1 \in \bar{X}} \binom{z}{i}, \quad F_2 = \sum_{i \geq 0: i+2 \in \bar{X}} \binom{z}{i}.$$

We claim that there is a z such that assumptions from Proposition B.4 hold. If we can choose z larger than $\max X + 1$, then $F_0, F_1,$ and F_2 are never equal to 0. Now suppose that $F_0F_2 = (F_1)^2$. We will show a contradiction. We first expand the equations above. Then for every z ,

$$\begin{aligned} & \left(\sum_{i \in \bar{X}} \binom{z}{i} \right) \left(\sum_{i+2 \in \bar{X}} \binom{z}{i} \right) = \left(\sum_{i+1 \in \bar{X}} \binom{z}{i} \right)^2 \\ & \left(2^z - \sum_{i \in X} \binom{z}{i} \right) \left(2^z - \sum_{i+2 \in X} \binom{z}{i} \right) = \left(2^z - \sum_{i+1 \in X} \binom{z}{i} \right)^2 \end{aligned}$$

which implies $2^z Q_1(z) = Q_2(z)$, where

$$\begin{aligned} Q_1(z) &= \left(2 \sum_{i+1 \in X} \binom{z}{i} - \sum_{i \in X} \binom{z}{i} - \sum_{i+2 \in X} \binom{z}{i} \right) \\ Q_2(z) &= \left(\sum_{i+1 \in X} \binom{z}{i} \right)^2 - \left(\sum_{i \in X} \binom{z}{i} \right) \left(\sum_{i+2 \in X} \binom{z}{i} \right). \end{aligned}$$

For large enough z , we argue that $Q_1(z)$ is not identically zero. Observe that the second term in $Q_1(z)$ gives a non-zero $z^{\max X}$ monomial whereas the other two terms cannot give a monomial of this degree. Now, since X is a fixed, finite set, Q_1, Q_2 are polynomials with constant degree. Thus, $Q_1(z)$ is zero only for finitely many z . Hence, there are infinitely many (positive) z such that $Q_1(z)$ is non-zero. For each such z we have

$$|2^z Q_1(z)| = |Q_2(z)|.$$

This is immediately a contradiction since $2^z = \omega(z^c)$ for any constant c if z is large enough. Thus, there is some positive integral value of z such that $F_0 F_2 \neq (F_1)^2$. We use this value of z in the construction of the gadget. Note that z only depends on X and can thus be precomputed.

Case 2: $0 \notin X, 1 \in X$. In this case we do not use $\text{HW}_{\in\{0,1\}}^{(1)}$ nodes but EQ_2 nodes, i.e. $\text{HW}_{\in\{0,2\}}^{(2)}$ nodes, instead. We still attach z of these nodes by $2z$ edges to the vertices on the path. Then the proof follows similarly to the previous case. The formal proof is given in the full version [29]. ◀

► **Lemma B.5.** *Let $X \subseteq \mathbb{N}$ be a fixed, finite set with $X \not\subseteq \{0\}$. Let R_1, \dots, R_d be d arbitrary relations for some $d \geq 0$. There is a polynomial-time Turing reduction from*

$$\text{Holant}(R_1, \dots, R_d, \text{HW}_{\in\bar{X}}, w[1, 1], w[-1, 1], w[0, 1]) \text{ to } \text{Holant}(R_1, \dots, R_d, \text{HW}_{\in\bar{X}})$$

such that Δ^ decreases and pw increases to $\text{pw} + \Delta^* \cdot f(\max X)$.*

Proof. We first use Lemma B.3 to remove the $w[1, 1]$ nodes. Observe that this can alternatively be done by a simple construction using a fresh vertex with $\max X + 1$ forced edges. Then we apply the lemma two more times to remove the $w[-1, 1]$ nodes and finally the $w[0, 1]$ nodes. ◀

Now we can prove the reduction from $\#Y\text{-ANTIFACTOR}^{\mathcal{R}}$ to $\#(X, Y)\text{-ANTIFACTOR}$.

Proof of Lemma 6.2. By the reduction of Lemma B.1 we can reduce $\#Y\text{-ANTIFACTOR}^{\mathcal{R}}$ to $\text{Holant}(\text{HW}_{\in\bar{Y}}, \text{HW}_{=1})$. This can trivially be reduced to $\text{Holant}(\text{HW}_{\in\bar{Y}}, \text{HW}_{\in\bar{X}}, \text{HW}_{=1})$ as we do not have any vertices with relation $\text{HW}_{\in\bar{X}}$. Then we invoke Lemmas B.2 and B.5 such that the vertices with relation $\text{HW}_{\in\bar{Y}}$ are not changed (or used for any construction). By this we end the reduction with $\text{Holant}(\text{HW}_{\in\bar{Y}}, \text{HW}_{\in\bar{X}})$ which precisely corresponds to $\#(X, Y)\text{-ANTIFACTOR}$. ◀

Parameterized Complexity of Maximum Happy Set and Densest k -Subgraph

Yosuke Mizutani  

School of Computing, University of Utah, Salt Lake City, UT, USA

Blair D. Sullivan  

School of Computing, University of Utah, Salt Lake City, UT, USA

Abstract

We present fixed-parameter tractable (FPT) algorithms for two problems, MAXIMUM HAPPY SET (MAXHS) and DENSEST k -SUBGRAPH (DkS) – also known as MAXIMUM EDGE HAPPY SET. Given a graph G and an integer k , MAXHS asks for a set S of k vertices such that the number of *happy vertices* with respect to S is maximized, where a vertex v is happy if v and all its neighbors are in S . We show that MAXHS can be solved in time $\mathcal{O}(2^{\text{mw}} \cdot \text{mw} \cdot k^2 \cdot |V(G)|)$ and $\mathcal{O}(8^{\text{cw}} \cdot k^2 \cdot |V(G)|)$, where mw and cw denote the *modular-width* and the *clique-width* of G , respectively. This answers the open questions on fixed-parameter tractability posed in [1].

The DkS problem asks for a subgraph with k vertices maximizing the number of edges. If we define *happy edges* as the edges whose endpoints are in S , then DkS can be seen as an edge-variant of MAXHS. In this paper we show that DkS can be solved in time $f(\text{nd}) \cdot |V(G)|^{\mathcal{O}(1)}$ and $\mathcal{O}(2^{\text{cd}} \cdot k^2 \cdot |V(G)|)$, where nd and cd denote the *neighborhood diversity* and the *cluster deletion number* of G , respectively, and f is some computable function. This result implies that DkS is also fixed-parameter tractable by *twin cover number*.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis; Theory of computation \rightarrow Fixed parameter tractability

Keywords and phrases parameterized algorithms, maximum happy set, densest k -subgraph, modular-width, clique-width, neighborhood diversity, cluster deletion number, twin cover

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.23

Funding This work was supported in part by the Gordon & Betty Moore Foundation under award GBMF4560 to Blair D. Sullivan.

Acknowledgements We thank Arnab Banerjee, Oliver Flatt and Thanh Son Nguyen for their contributions to a course project that led to this research.

1 Introduction

In the study of large-scale networks, *communities* – cohesive subgraphs in a network – play an important role in understanding complex systems and appear in sociology, biology and computer science, etc. [16, 24]. For example, the concept of homophily in sociology explains the tendency for individuals to associate themselves with similar people [26]. Homophily is a fundamental law governing the structure of social networks, and finding groups of people sharing similar interests has many real-world applications [11].

People have attempted to frame this idea as a graph optimization problem, where a vertex represents a person and an edge corresponds to some relation in the social network. The notion of *happy vertices* was first introduced by Zhang and Li in terms of graph coloring [30], where each color represents an attribute of a person (possibly fixed). A vertex is *happy* if all of its neighbors share its color. The goal is to maximize the number of happy vertices by changing the color of unfixed vertices, thereby achieving the greatest social benefit.



© Yosuke Mizutani and Blair D. Sullivan;

licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 23; pp. 23:1–23:18

Leibniz International Proceedings in Informatics



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

Later, Asahiro et al. introduced MAXIMUM HAPPY SET (MAXHS) which defines that a vertex v is *happy* with respect to a *happy set* S if v and all of its neighbors are in S [1]. The MAXHS problem asks for a vertex set S of size k that maximizes the number of happy vertices. They also define its edge-variant, MAXIMUM EDGE HAPPY SET (MAXEHS) which maximizes the number of *happy edges*, an edge with both endpoints in the *happy set*. It is clear to see that MAXEHS is equivalent to choosing a vertex set S such that the number of edges in the induced subgraph on S is maximized. This problem is known as DENSEST k -SUBGRAPH (DkS) in other literature. Both MAXHS and MAXEHS are NP-hard [1, 12], and we study their parameterized complexity throughout this paper.

1.1 Parameterized Complexity and Related Work

Graph problems are often studied with a variety of structural parameters in addition to natural parameters (size k of the happy set in our case). Specifically, we investigate the parameterized complexity with respect to *modular-width* (mw), *clique-width* (cw), *neighborhood diversity* (nd), *cluster deletion number* (cd), *twin cover number* (tc), *treewidth* (tw), and *vertex cover number* (vc), all of which we define in Section 2.3. Figure 1 illustrates the hierarchy of these parameters by inclusion; hardness results are implied along the arrows, and FPT¹ algorithms are implied in the reverse direction.

Asahiro et al. showed that MAXHS is W[1]-hard with respect to k by a parameterized reduction from the q -CLIQUE problem [1]. They also presented FPT algorithms for MAXHS on parameters: clique-width plus k , neighborhood diversity, cluster deletion number (which implies FPT by twin cover number), and treewidth.

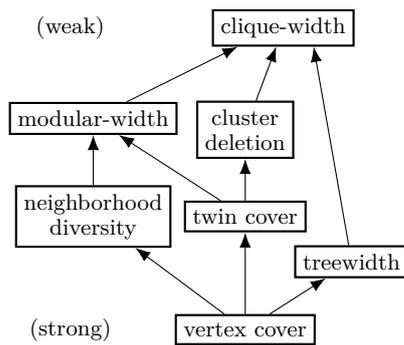
MAXEHS (DkS) has been extensively studied in different names (e.g. the k -CLUSTER problem [7], the HEAVIEST UNWEIGHTED SUBGRAPH problem [22], and k -CARDINALITY SUBGRAPH problem [5]). As for parameterized complexity, Cai showed the W[1]-hardness parameterized by k [6]. Bourgeois employed Moser’s technique in [27] to show that MAXEHS can be solved in time $2^{\text{tw}} \cdot n^{\mathcal{O}(1)}$ [3]. Broersma et al. proved that MAXEHS can be solved in time $k^{\mathcal{O}(\text{cw})} \cdot n$, but it cannot be solved in time $2^{\mathcal{O}(\text{cw} \log k)} \cdot n^{\mathcal{O}(1)}$ unless the Exponential Time Hypothesis (ETH) fails [4]. To the best of our knowledge, the parameterized complexity by modular-width, neighborhood diversity, cluster deletion number and twin cover number remained open prior to our work. Figure 2 summarizes the known and established hardness results for MAXHS and MAXEHS.

1.2 Our Contributions

In this paper, we present four novel parameterized algorithms for MAXHS and MAXEHS. First, we shall provide a dynamic-programming algorithm that solves MAXHS in time $\mathcal{O}(2^{\text{mw}} \cdot \text{mw} \cdot k^2 \cdot |V|)$, answering the question posed by the authors of [1]. Second, we show that MAXHS is FPT by clique-width, giving an $\mathcal{O}(8^{\text{cw}} \cdot k^2 \cdot |V|)$ algorithm, which removes the exponential term of k from the best known result, $\mathcal{O}(6^{\text{cw}} \cdot k^{2(\text{cw}+1)} \cdot |V|)$ [1]. While bounded modular-width implies bounded clique-width (Proposition 9), we give both algorithms because the one for modular-width has asymptotically faster running time.

Turning to MAXEHS, we prove it is FPT by neighborhood diversity, using an integer quadratic programming formulation. Lastly, we provide an FPT algorithm for MAXEHS parameterized by cluster deletion number with running time $\mathcal{O}(2^{\text{cd}} \cdot k^2 \cdot |V|)$, which also

¹ An FPT (fixed-parameter tractable) algorithm solves the problem in time $f(k) \cdot n^{\mathcal{O}(1)}$ for some computable function f .



■ **Figure 1** Hierarchy of relevant structural graph parameters. Arrows indicate generalizations.

Parameter	MAXHS	MAXEHS
Size k of happy set	W[1]-hard[1]	W[1]-hard[6]
Clique-width + k	FPT[1]	FPT[4]
Clique-width	FPT	W[1]-hard[4]
Modular-width	FPT	Open
Neighborhood diversity	FPT[1]	FPT
Cluster deletion number	FPT[1]	FPT
Twin cover number	FPT[1]	FPT
Treewidth	FPT[1]	FPT[3]
Vertex cover number	FPT[1]	FPT[3]

■ **Figure 2** Known and established hardness results under select parameters for MAXHS and MAXEHS (as known as DENSEST k -SUBGRAPH). New results from this paper in red.

implies the problem is FPT by twin cover number. These new results complete the previously-open parameterized complexities in Figure 2, except the one of DkS parameterized by modular-width.

Independent Work

Independently and simultaneously, Hanaka also showed the parameterized complexity of DkS by neighborhood diversity and cluster deletion number [21]. For neighborhood diversity, the complexity was implied by [25], as we discuss in sections 3.3 and 5.1. Further, the parameterized complexity by cluster deletion number was shown by an algorithm solving DkS in time $2^{\text{bd}} ((k^3 + \text{bd}) |V| + |E|)$, where bd denotes the block deletion number (note that it holds $\text{cd} \leq \text{bd}$).

2 Preliminaries

We use standard graph theory notation, following [10]. Given a graph $G = (V, E)$, we write $n(G) = |V|$ for the number of vertices and $m(G) = |E|$ for the number of edges. We use $N(v)$ and $N[v]$ to denote the open and closed neighborhoods of a vertex v , respectively, and for a vertex set $X \subseteq V$, $N[X]$ denotes the union of $N[x]$ for all $x \in X$. We write $\text{deg}_G(v) = \text{deg}(v)$ for the degree of a vertex v . We denote the induced subgraph of G on a set $X \subseteq V$ by $G[X]$. We say vertices u and v are *twins* if they have the same neighbors, i.e. $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. Further, they are called *true twins* if $uv \in E$.

2.1 Problem Definitions

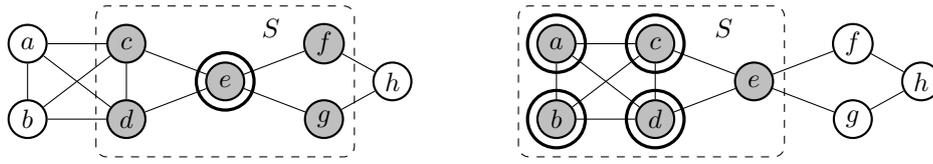
Asahiro et al. first introduced the MAXIMUM HAPPY SET problem in [1].

MAXIMUM HAPPY SET (MAXHS)

Input: A graph $G = (V, E)$ and a positive integer k .

Problem: Find a subset $S \subseteq V$ of k vertices that maximizes the number of *happy* vertices v with $N[v] \subseteq S$.

23:4 Parameterized Complexity of Maximum Happy Set and Densest k -Subgraph



■ **Figure 3** Given a graph above, if $k = 5$, choosing $S = \{c, d, e, f, g\}$ makes only one vertex (e) happy (left). On the other hand, $S = \{a, b, c, d, e\}$ is an optimal solution, making four vertices (a, b, c, d) happy (right).

Figure 3 illustrates an example instance with $k = 5$. Let us call a vertex that is not happy an *unhappy* vertex, and observe that the set of unhappy vertices is given by $N[V \setminus S]$, providing an alternative characterization of happy vertices.

► **Proposition 1.** *Given a graph $G = (V, E)$ and a happy set $S \subseteq V$, the set of happy vertices is given by $V \setminus N[V \setminus S]$.*

In addition, Asahiro et al. define an edge variant [1]:

MAXIMUM EDGE HAPPY SET (MAXEHS)

Input: A graph $G = (V, E)$ and a positive integer k .

Problem: Find a set $S \subseteq V$ of k vertices that maximizes the number of happy edges.

An edge $uv \in E$ is *happy* if and only if $\{u, v\} \subseteq S$.

It is known that MAXEHS is identical to the DENSEST k -SUBGRAPH problem (DkS), as the number of happy edges is equal to $m(G[S])$. Some literature (e.g. [14]) also phrases this problem as the dual of the SPARSEST k -SUBGRAPH problem.

2.2 Structural Parameters

We now define the structural graph parameters considered in this paper.

Treewidth. The most-studied structural parameter is treewidth, introduced by Robertson & Seymour in [28]. Treewidth measures how a graph resembles a tree and admits FPT algorithms for a number of NP-hard problems, such as WEIGHTED INDEPENDENT SET, DOMINATING SET, and STEINER TREE [9]. Treewidth is defined by the following notion of tree decomposition.

► **Definition 2** (treewidth [9]). *A tree decomposition of a graph G is a pair $(T, \{X_t\}_{t \in V(T)})$, where T is a tree and $X_t \subseteq V(G)$ is an assigned vertex set for every node t , such that the following three conditions hold:*

- $\bigcup_{t \in V(T)} X_t = V(G)$.
- For every $uv \in E(G)$, there exists a node t such that $u, v \in X_t$.
- For every $u \in V(G)$, the set $T_u = \{t \in V(T) : u \in X_t\}$ induces a connected subtree of T .

*The width of tree decomposition is defined to be $\max_{t \in V(T)} |X_t| - 1$, and the **treewidth** of a graph G , denoted by tw , is the minimum possible width of a tree decomposition of G .*

Clique-width. Clique-width is a generalization of treewidth and can capture dense, but structured graphs. Intuitively, a graph with bounded clique-width k can be built from single vertices by joining structured parts, where vertices are associated by at most k labels such that those with the same label are indistinguishable in later steps.

► **Definition 3** (clique-width [8]). For a positive integer w , a w -labeled graph is a graph whose vertices are labeled by integers in $\{1, \dots, w\}$. The **clique-width** of a graph G , denoted by cw , is the minimum w such that G can be constructed by repeated application of the following operations:

- (O1) Introduce $i(v)$: add a new vertex v with label $i \in \{1, \dots, w\}$.
- (O2) Union $G_1 \oplus G_2$: take a disjoint union of w -labeled graphs G_1 and G_2 .
- (O3) Join $\eta(i, j)$: take two labels i and j , and then add an edge between every pair of vertices labeled by i and by j .
- (O4) Relabel $\rho(i, j)$: relabel the vertices of label i to label $j \in \{1, \dots, w\}$.

This construction naturally defines a rooted binary tree, called a cw -expression tree G , where G is the root and each node corresponds to one of the above operations.

Neighborhood Diversity. Neighborhood diversity is a parameter introduced by Lampis [23], which measures the number of twin classes.

► **Definition 4** (neighborhood diversity [23]). The **neighborhood diversity** of a graph $G = (V, E)$, denoted by nd , is the minimum number w such that V can be partitioned into sets of twin vertices.

By definition, each set of twins, called a *module*, is either a clique or an independent set.

Cluster Deletion Number. Cluster (vertex) deletion number is the distance to a cluster graph, which consists of disjoint cliques.

► **Definition 5** (cluster deletion number). A vertex set X is called a *cluster deletion set* if $G[V \setminus X]$ is a cluster graph. The **cluster deletion number** of G , denoted by cd , is the size of the minimum cluster deletion set in G .

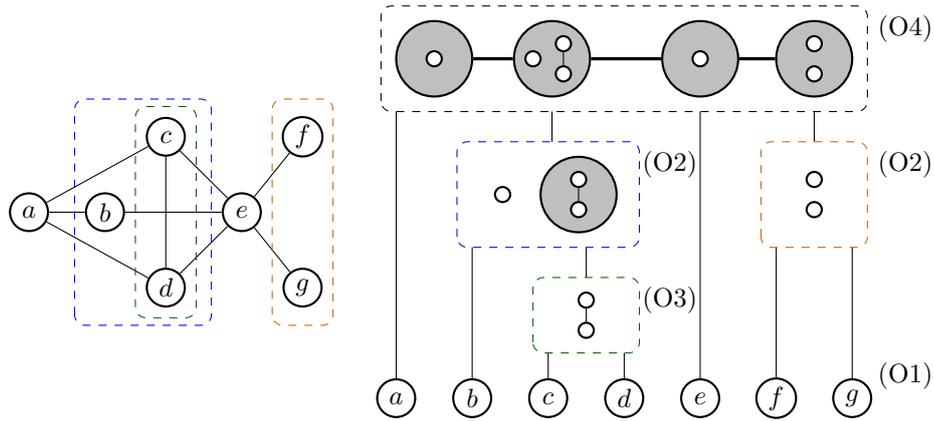
Twin Cover Number. The notion of twin cover is introduced by Ganian [19] and offers a generalization of vertex cover number.

► **Definition 6** (twin cover number [19]). A vertex set $X \subseteq V$ is a *twin cover* of $G = (V, E)$ if for every edge $uv \in E$ either (1) $u \in X$ or $v \in X$, or (2) u and v are true twins. The **twin cover number**, denoted by tc , is the size of the minimum twin cover of G .

Modular-width. Modular-width is a parameter introduced by Gajarský et al. [17] to generalize simpler notions on dense graphs while avoiding the negative results brought by moving to the full generality of clique-width (e.g. many problems FPT for treewidth becomes W[1]-hard for clique-width [13, 14, 15]). Modular-width is defined using the standard concept of modular decomposition.

► **Definition 7** (modular-width [17]). Any graph can be produced via a sequence of the following operations:

- (O1) Introduce: Create an isolated vertex.
- (O2) Union $G_1 \oplus G_2$: Create the disjoint union of two graphs G_1 and G_2 .
- (O3) Join: Given two graphs G_1 and G_2 , create the complete join G_3 of G_1 and G_2 . That is, a graph G_3 with vertices $V(G_1) \cup V(G_2)$ and edges $E(G_1) \cup E(G_2) \cup \{(v, w) : v \in G_1, w \in G_2\}$.
- (O4) Substitute: Given a graph G with vertices v_1, \dots, v_n and given graphs G_1, \dots, G_n , create the substitution of G_1, \dots, G_n in G . The substitution is a graph \mathcal{G} with vertex set $\bigcup_{1 \leq i \leq n} V(G_i)$ and edge set $\bigcup_{1 \leq i \leq n} E(G_i) \cup \{(v, w) : v \in G_i, w \in G_j, (v_i, v_j) \in E(G)\}$. Each graph G_i is substituted for a vertex v_i , and all edges between graphs corresponding to adjacent vertices in G are added.



■ **Figure 4** An example graph G (left) with modular-width 4. Modular decomposition of the same graph is shown at right. The parse-tree has G as the root, and its nodes correspond to operations (O1)-(O4). Notice that each node also represents a *module* – module members have the same neighbors outside the module.

These operations, taken together in order to construct a graph, form a parse-tree of the graph. The width of a graph is the maximum size of the vertex set of G used in operation (O4) to construct the graph. The **modular-width**, denoted by mw , is the minimum width such that G can be obtained from some sequence of operations (O1)-(O4).

Finding a parse-tree of a given graph, called a *modular decomposition*, can be done in linear-time [29]. See Figure 4 for an illustration of modular decomposition. Gajarský et al. also give FPT algorithms parameterized by modular-width for PARTITION INTO PATHS, HAMILTONIAN PATH, HAMILTONIAN CYCLE and COLORING, using bottom-up dynamic programming along the parse-tree.

Vertex Cover Number. The vertex cover number is the solution size of the classic VERTEX COVER problem.

▶ **Definition 8.** A vertex set $X \subseteq V$ is a vertex cover of $G = (V, E)$ if for every edge $uv \in E$ either $u \in X$ or $v \in X$. The **vertex cover number** of G , denoted by vc , is the size of the minimum vertex cover of G .

2.2.1 Properties of Structural Parameters

Finally, we note the relationship among the parameters defined above, which establishes the hierarchy shown in Figure 1.

▶ **Proposition 9** ([1, 17]). Let cw , tw , cd , nd , tc , vc , mw be the clique-width, tree-width, cluster deletion number, neighborhood diversity, twin cover number, vertex cover number, and modular-width of a graph G , respectively. Then the following inequalities hold²: (i) $\text{cw} \leq 2^{\text{tw}+1} + 1$; (ii) $\text{tw} \leq \text{vc}$; (iii) $\text{nd} \leq 2^{\text{vc}} + \text{vc}$; (iv) $\text{cw} \leq 2^{\text{cd}+3} - 1$; (v) $\text{cd} \leq \text{tc} \leq \text{vc}$; (vi) $\text{mw} \leq \text{nd}$; (vii) $\text{mw} \leq 2^{\text{tc}} + \text{tc}$; and (viii) $\text{cw} \leq \text{mw} + 2$.

² $\text{cw} \leq 2$ when $\text{mw} = 0$; otherwise, $\text{cw} \leq \text{mw} + 1$.

3 Background

Before describing our algorithms, we introduce some building blocks for our argument.

3.1 Entire Subgraphs

Structural parameters such as modular-width and clique-width entail the join operation in their underlying construction trees. When joining two subgraphs in MAXHS, it is important to distinguish whether all the vertices in the subgraph are included in the happy set. Formally, we introduce the notion of *entire subgraphs*.

► **Definition 10.** *Given a graph G and a happy set S , an **entire subgraph** with respect to S is a subgraph G' of G such that $V(G') \subseteq S$.*

By definition, the empty subgraph is always entire. The following lemma is directly derived from the definition of happy vertices.

► **Lemma 11.** *Let G be a complete join of subgraphs G_1 and G_2 . For any happy set $S \subseteq V(G)$, $V(G_1)$ contains a happy vertex only if G_2 is entire with respect to S .*

Proof. If G_2 is not entire, there must exist $v \in V(G_2)$ such that $v \notin S$. Recall Proposition 1, and we have $N[V(G) \setminus S] \supseteq N(v) \supseteq V(G_1)$, which implies that any vertex in $V(G_1)$ cannot be happy. ◀

3.2 Knapsack Variant with Non-linear Values

The classic KNAPSACK problem has a number of variants, including 0-1 KNAPSACK [20] and QUADRATIC KNAPSACK [18]. In this paper we consider another variant, where the objective function is the sum of non-linear functions, but the function range is limited to integers. Specifically, each item has unit weight, but its value may vary depending on the number of copies of each type of item. We also require the weight sum to be exact and call this problem f -KNAPSACK, where f stands for function.

f -KNAPSACK

Input: Given a set of n items numbered from 1 to n , a weight capacity $W \in \mathbb{Z}_0^+$ and a value function $f_i : D_i \rightarrow \mathbb{Z}_0^+$, defined on a non-negative integral domain D_i for each item i .

Problem: For every $1 \leq i \leq n$, find the number $x_i \in D_i$ of instances of item i to include in the knapsack, maximizing $\sum_{i=1}^n f_i(x_i)$, subject to $\sum_{i=1}^n x_i = W$.

We show that this problem is solvable in polynomial-time.

► **Lemma 12.** *f -KNAPSACK can be solved in time $\mathcal{O}(nW^2)$.*

Proof. First, define the value $\phi[t, w]$ to be the maximum possible sum $\sum_{i=1}^t f_i(x_i)$, subject to $\sum_{i=1}^t x_i = w$ and $x_i \in D_i$ for every i . Then, perform bottom-up dynamic programming as follows.

- Initialize: $\phi[0, w] = \begin{cases} 0 & \text{if } w = 0, \\ -\infty & \text{if } w > 0 \text{ (meaning Infeasible)}. \end{cases}$
- Update: $\phi[t, w] = \max_{x_t \in D_t \wedge x_t \leq w} f_t(x_t) + \phi[t-1, w-x_t]$
- Result: $\phi[n, w]$ for $0 \leq w \leq W$ is the optimal value for weight w .

The base case ($\phi[0, w]$) represents the state where no item is in the knapsack, so both the objective and weight are 0; otherwise, infeasible. For the inductive step, any optimal solution $\phi[t, w]$ can be decomposed into $f_t(x_t) + \sum_{i=1}^{t-1} f_i(x_i)$ for some x_t , and the latter term ($\sum_{i=1}^{t-1} f_i(x_i)$) must equal $\phi[t-1, w-x_t]$ by definition. We consider all possible integers x_t , and thus the algorithm is correct.

Since $0 \leq t \leq n$, $0 \leq w \leq W$, and the update takes time $\mathcal{O}(W)$, the total running time is $\mathcal{O}(nW^2)$. By using the standard technique of backlinks, one can reconstruct the solution $\{x_i\}$ within the same asymptotic running time. ◀

The following result is a natural by-product of the algorithm above.

► **Corollary 13.** *Given an integer W , f -KNAPSACK for all weight capacities $0 \leq w \leq W$ can be solved in total time $\mathcal{O}(nW^2)$.*

3.3 Integer Quadratic Programming

For MAXEHS, we use the following known result that INTEGER QUADRATIC PROGRAMMING is FPT by the number of variables and coefficients.

INTEGER QUADRATIC PROGRAMMING (IQP)

Input: An $n \times n$ integer matrix Q , an $m \times n$ integer matrix A and an m -dimensional integer vector b .

Problem: Find a vector $x \in \mathbb{Z}^n$ minimizing $x^T Q x$, subject to $Ax \leq b$.

► **Proposition 14** (Lokshtanov [25]). *There exists an algorithm that given an instance of IQP, runs in time $f(n, \alpha)L^{\mathcal{O}(1)}$, and outputs a vector $x \in \mathbb{Z}^n$. If the input IQP has a feasible solution then x is feasible, and if the input IQP is not unbounded, then x is an optimal solution. Here α denotes the largest absolute value of an entry of Q and A , and L is the total number of bits required to encode the input.*

It is convenient to have a linear term in the objective function. This can be achieved by introducing a new variable $\hat{x} = 1$ and adding $[0, q]$ as the corresponding row in Q [25].

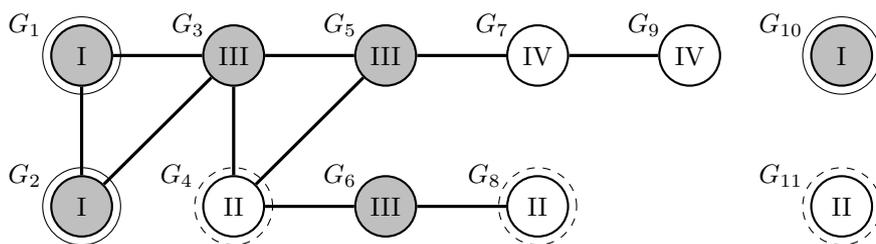
► **Corollary 15.** *Proposition 14 holds if we generalize the objective function from $x^T Q x$ to $x^T Q x + q^T x$ for some n -dimensional integer vector q . Here α is the largest absolute value of an entry of Q , q and A .*

4 Algorithms for Maximum Happy Set

Now we describe our FPT algorithms for MAXHS with respect to modular-width and clique-width. At a high level, we employ a bottom-up dynamic programming (DP) approach on the parse-tree of a given graph, considering each node once. At each node, we use several techniques on precomputed results to update the DP table. For simplicity, our DP tables store the maximum number of happy vertices. Like other DP applications, a *certificate*, i.e. the actual happy set, can be found by using backlinks within the same asymptotic running time.

4.1 Parameterized by modular-width

We give an algorithm whose running time is singly-exponential in the modular-width, quadratic in k and linear in the graph size.



■ **Figure 5** Four types of the subgraphs after applying operation (O4). Entire subgraphs (Type I and III) are shaded in gray. A subgraph becomes Type III or IV if it has a non-entire neighbor (e.g. G_3, G_9). In a Type I subgraph, all vertices are happy. Type II subgraphs may or may not admit a happy vertex. Type III and IV subgraphs cannot contain a happy vertex, as it is adjacent to a non-entire subgraph.

► **Theorem 16.** *MAXHS can be solved in time $\mathcal{O}(2^{\text{mw}} \cdot \text{mw} \cdot k^2 \cdot |V(G)|)$, where mw is the modular-width of the input graph G .*

Our algorithm follows the common framework seen in [17]. Given a graph G , a parse-tree with modular-width mw can be computed in linear-time [29]. The number of nodes in the parse-tree is linear in $|V(G)|$ [17]. Our algorithm traverses the parse-tree from the bottom, considering only operation (O4), as operations (O2)-(O3) can be replaced with a single operation (O4) with at most two arguments [17]. Further, we assume $2 \leq \text{mw} < k$ without loss of generality.

Each node in the parse-tree corresponds to an induced subgraph of G , which we write \mathcal{G} . We keep track of a table $\phi[\mathcal{G}, w]$, the maximum number of happy vertices for \mathcal{G} with regard to a happy set of size w . We may assume $0 \leq w \leq k$ because we do not have to consider a happy set larger than size k . The entries of the DP table are initialized with $\phi[\mathcal{G}, w] = -\infty$. For the base case, a graph G_0 with a single vertex introduced by operation (O1), we set $\phi[G_0, 0] = 0$ and $\phi[G_0, 1] = 1$. The solution to the original problem is given by $\phi[G, k]$.

Our remaining task is to compute, given a graph substitution $\mathcal{G} = H(G_1, \dots, G_n)$ ($n \leq \text{mw}$), the values of $\phi[\mathcal{G}, w]$ provided partial solutions $\phi[G_1, \cdot], \dots, \phi[G_n, \cdot]$. We first choose a set of entire subgraphs from G_1, \dots, G_n . Then, we identify the *subgraph type* for each G_i during a graph substitution.

► **Definition 17 (subgraph type).** *Given a graph substitution $H(G_1, \dots, G_n)$, where $v_i \in V(H)$ is substituted by G_i , and a happy set S , we categorize each substituted subgraph G_i into the following four types.*

- *Type I: G_i is entire and for every j such that $v_j \in N_H(v_i)$, G_j is entire.*
- *Type II: G_i is not entire and for every j such that $v_j \in N_H(v_i)$, G_j is entire.*
- *Type III: G_i is entire and not Type I.*
- *Type IV: G_i is not entire and not Type II.*

Intuitively, Type I and II subgraphs are surrounded by entire subgraphs in H , the *metagraph* to substitute, and Type I and III subgraphs are entire. A pictorial representation of this partition is presented in Figure 5. Observe that from Lemma 11, the subgraphs with Type III and IV cannot include any happy vertices. Further, Type II subgraphs are independent in H because their neighbors must be of Type III. This ensures that the choice of a happy set in Type II is independent of other subgraphs.

Lastly, we formulate an f -KNAPSACK instance as described in the following algorithm for updating the DP table on a single operation (O4).

► **Algorithm 1** (MaxHS-MW). *Given a graph substitution $\mathcal{G} = H(G_1, \dots, G_n)$ and partial solutions $\phi[G_1, \cdot], \dots, \phi[G_n, \cdot]$, consider all combinations of entire subgraphs from G_1, \dots, G_n and proceed the following steps.*

(Step 1) Identify subgraph types for G_1, \dots, G_n . (Step 2) Formulate an f -KNAPSACK instance with capacity k and value functions f_i , based on the subgraph G_i 's type as follows.

- *Type I : $f_i(x) = |G_i|$, $x = |G_i|$.*
- *Type II : $f_i(x) = \phi[G_i, x]$, $0 \leq x < |G_i|$.*
- *Type III : $f_i(x) = 0$, $x = |G_i|$.*
- *Type IV : $f_i(x) = 0$, $0 \leq x < |G_i|$.*

Then, update the DP table entries $\phi[\mathcal{G}, w]$ for $0 \leq w \leq k$ with the solution to f -KNAPSACK, if its value is greater than the current value.

We now prove that the runtime of this algorithm is FPT with respect to modular-width.

► **Lemma 18.** *Algorithm 1 correctly computes $\phi[\mathcal{G}, w]$ for every $0 \leq w \leq k$ in time $\mathcal{O}(2^{\text{mw}} \cdot \text{mw} \cdot k^2)$.*

Proof. First, the algorithm considers all possible sets of entire substituted subgraphs (G_1, \dots, G_n) . The optimal solution must belong to one of them. It remains to prove the correctness of the f -KNAPSACK formulation in step 2. From Lemma 11, the subgraphs of Type III and IV cannot increase the number of happy vertices, so we set $f_i(x) = 0$. For Type I, the algorithm has no option but to include all of $V(G_i)$ in the happy set, and they are all happy.

The subgraphs of Type II are the only ones that use previous results, $\phi[G_i, \cdot]$. Since the new neighbors to G_i are required to be in the happy set, for any choice of the happy set in G_i , happy vertices remain happy, and unhappy vertices remain unhappy. Thus, we can directly use $\phi[G_i, \cdot]$; its choice does not affect other substituted subgraphs, as Type II subgraphs are independent in H . The domain of functions f_i is naturally determined by the definition of entire subgraphs.

Now, consider the running time of Algorithm 1. It considers 2^n possible combinations of entire subgraphs. Step 1 can be done by checking neighbors for each vertex in H , so the running time is $\mathcal{O}(|E(H)|) = \mathcal{O}(n^2)$. And step 2 takes time $\mathcal{O}(nk^2)$ from Corollary 13. The total running time is $\mathcal{O}(2^n(n^2 + nk^2)) = \mathcal{O}(2^{\text{mw}} \cdot \text{mw} \cdot k^2)$ as we assume $n \leq \text{mw} < k$. ◀

Proof of Theorem 16. It is trivial to see that the base case of the DP is valid, and the correctness of inductive steps is given by Lemma 18. We process each node of the parse-tree once, and it has $\mathcal{O}(|V(G)|)$ nodes [17]. Thus, the overall runtime is $\mathcal{O}(2^{\text{mw}} \cdot \text{mw} \cdot k^2 \cdot |V(G)|)$. ◀

4.2 Parameterized by clique-width

We provide an algorithm for MAXHS parameterized only by clique-width (cw), which no longer requires a combined parameter with solution size k as presented in [1].

► **Theorem 19.** *Given a cw-expression tree of a graph G with clique-width cw, MAXHS can be solved in time $\mathcal{O}(8^{\text{cw}} \cdot k^2 \cdot |V(G)|)$.*

Here we assume that we are given a cw-expression tree, where each node t represents a labeled graph G_t . A labeled graph is a graph whose vertices are labeled by integers in $L = \{1, \dots, \text{cw}\}$. Every node must be one of the following: introduce node $i(v)$, union node $G_1 \oplus G_2$, relabel node $\rho(i, j)$, or join node $\eta(i, j)$. We write V_i for the set of vertices with label i .

Our algorithm traverses the cw-expression tree from the leaves and performs dynamic programming. For every node t , we keep track of the annotated partial solution $\phi[t, w, X, T]$, for every integer $0 \leq w \leq k$ and sets of labels $X, T \subseteq L$. We call X the *entire labels* and T the *target labels*. $\phi[t, w, X, T]$ is defined to be the maximum number of happy vertices having target labels T for G_t with respect to a happy set $S \subseteq V(G_t)$ of size w such that V_ℓ is entire to S if and only if $\ell \in X$. The entries of the DP table are initialized with $\phi[t, w, X, T] = -\infty$. The solution to the original graph G is computed by $\max_{X \subseteq L} \phi[r, k, X, L]$, where r is the root of the cw-expression tree. Now we claim the following recursive formula for each node type.

► **Lemma 20** (Formula for introduce nodes). *Suppose t is an introduce node, where a vertex v with label i is introduced. Then, the following holds.*

$$\phi[t, w, X, T] = \begin{cases} 1 & \text{if } w = 1, X = L \text{ and } i \in T; \\ 0 & \text{if } w = 1, X = L \text{ and } i \notin T; \\ 0 & \text{if } w = 0 \text{ and } X = L \setminus \{i\}; \\ -\infty & \text{otherwise.} \end{cases}$$

Proof. First, notice that all labels but i are empty and thus entire. If we include v in the happy set, then we get $w = 1$ and $X = L$ (all labels are entire). The resulting value depends on the target labels. If i is a target label, i.e. $i \in T$, then v is a happy vertex having a target label, resulting in $\phi[t, w, X, T] = 1$. Otherwise, $\phi[t, w, X, T] = 0$. If $w = 0$, then label i cannot be entire, and the only feasible solution is $\phi[t, 0, L \setminus \{i\}, T] = 0$. ◀

► **Lemma 21** (Formula for union nodes). *Suppose t is a union node with child nodes t_1 and t_2 . Then, the following holds.*

$$\phi[t, w, X, T] = \max_{0 \leq \tilde{w} \leq w} \max_{\substack{X_1, X_2 \subseteq L \\ X_1 \cap X_2 = X}} \phi[t_1, \tilde{w}, X_1, T] + \phi[t_2, w - \tilde{w}, X_2, T]$$

Proof. At a union node, since G_{t_1} and G_{t_2} are disjoint, any maximum happy set in G_t must be the disjoint union of some maximum happy set in G_{t_1} and that in G_{t_2} for the same target labels. We consider all possible combinations of partial solutions to G_{t_1} and G_{t_2} , so the optimality is preserved. Note that a label in G_t is entire if and only if it is entire in both G_{t_1} and G_{t_2} . ◀

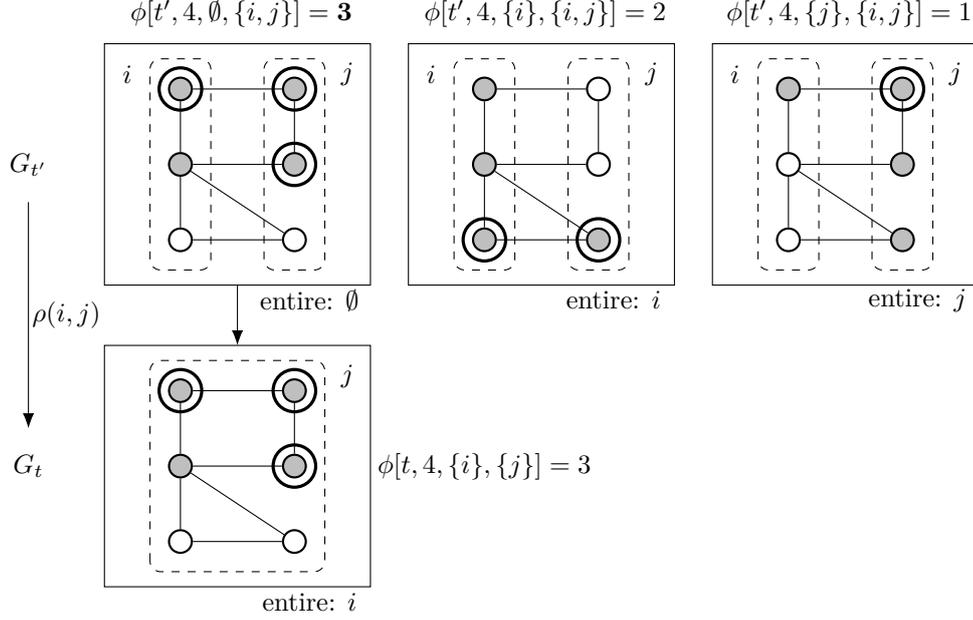
► **Lemma 22** (Formula for relabel nodes). *Suppose t is a relabel node with child node t' , where label i in graph $G_{t'}$ is relabeled to j . Then, the following holds.*

$$\phi[t, w, X, T] = \begin{cases} -\infty & \text{if } i \notin X; \\ \phi[t', w, X, T'] & \text{if } i \in X \text{ and } j \in X; \\ \max_{Y \in \{\emptyset, \{i\}, \{j\}\}} \phi[t', w, X \setminus \{i\} \cup Y, T'] & \text{if } i \in X \text{ and } j \notin X, \end{cases}$$

where $T' = T \cup \{i\}$ if $j \in T$ and $T \setminus \{i\}$ otherwise.

Proof. At a relabel node $\rho(i, j)$, label i becomes empty, so it must be entire in G_t , leading to the first case. The variable T' converts the target labels in $G_{t'}$ to those in G_t . If label j is a target in G_t , then i and j must be targets in $G_{t'}$. Likewise, if label j is not a target in G_t , then neither i nor j should be targets in $G_{t'}$.

If label j is entire in G_t , then the maximum happy set must be the same as the one in $G_{t'}$ where both labels i and j are entire. If j is not entire in G_t , then we need to choose the best solution from the following: i is entire but j is not, j is entire but i is not, neither i nor j is entire. Because G_t and $G_{t'}$ have the same underlying graph, the optimal solution must be one of these. ◀



■ **Figure 6** Visualization of the relabel operation $\rho(i, j)$ in the cw-expression tree of labels i, j . Figures show happy sets of size 4 (shaded in gray) maximizing the number of happy vertices (shown with double circles). After relabeling i to j , label i becomes empty and thus entire. Since label j in G_t corresponds to labels i and j in $G_{t'}$, to compute $\phi[t, 4, \{i\}, \{j\}]$, we need to look up three partial solutions $\phi[t', 4, \emptyset, T']$, $\phi[t', 4, \{i\}, T']$, and $\phi[t', 4, \{j\}, T']$, where $T' = \{i, j\}$, and keep the one with the largest value (the left one in this example).

Figure 6 illustrates the third case of the formula in Lemma 22.

► **Lemma 23** (Formula for join nodes). *Suppose t is a join node with the child node t' , where labels i and j are joined. Then, the following holds.*

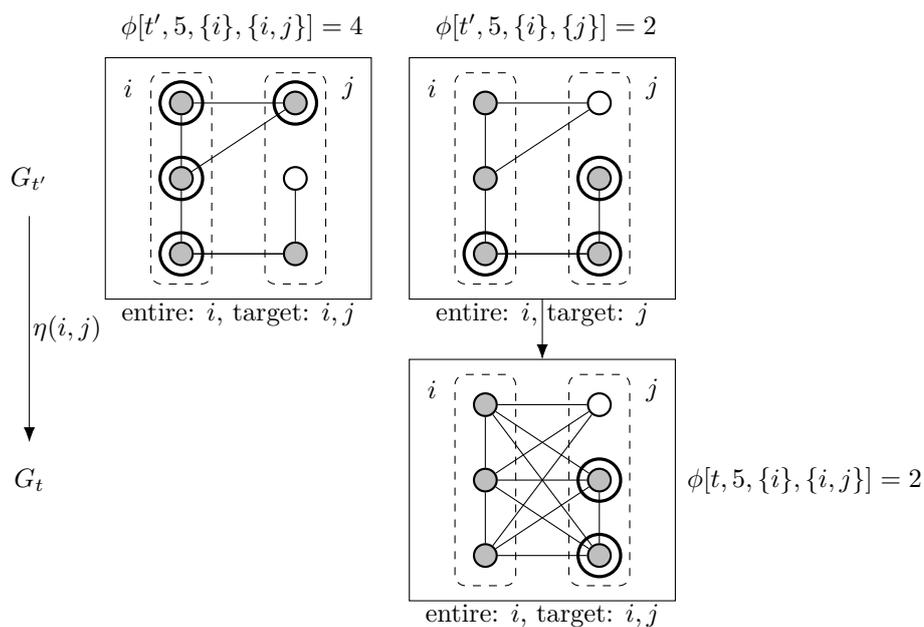
$$\phi[t, w, X, T] = \begin{cases} \phi[t', w, X, T] & \text{if } i \in X \text{ and } j \in X \\ \phi[t', w, X, T \setminus \{i\}] & \text{if } i \in X \text{ and } j \notin X \\ \phi[t', w, X, T \setminus \{j\}] & \text{if } i \notin X \text{ and } j \in X \\ \phi[t', w, X, T \setminus \{i, j\}] & \text{if } i \notin X \text{ and } j \notin X \end{cases}$$

Proof. At a join node $\eta(i, j)$, first observe that for any happy set, the vertices labeled other than i, j are unaffected; happy vertices remain happy. Further, if label j is not entire in G_t , then all vertices in V_i cannot be happy from Lemma 11. Thus, the maximum happy set in G_t is equivalent to the one in $G_{t'}$ such that label i is not a target label. The same argument applies to the other cases. ◀

Figure 7 illustrates the second case of the formula in Lemma 23. Lastly, we examine the running time of these computations.

► **Proposition 24.** *Given a cw-expression tree and its node t , and partial solutions $\phi[t', \cdot, \cdot, \cdot]$ for every child node t' of t , we can compute $\phi[t, w, X, T]$ for every w, X, T in time $\mathcal{O}(8^{\text{cw}} \cdot k^2)$.*

Proof. It is clear to see that for fixed t, w, X, T , the formulae for introduce, relabel, and join nodes take $\mathcal{O}(1)$. If we compute $\phi[t, \cdot, \cdot, \cdot]$ for every w, X, T , the total running time is $\mathcal{O}\left((2^{\text{cw}})^2 \cdot k\right)$ since w is bounded by k and there are 2^{cw} configurations for X and T .



■ **Figure 7** Visualization of the join operation $\eta(i, j)$ in the cw-expression tree of labels i, j . Figures show happy sets of size 5 (shaded in gray) maximizing the number of happy vertices (shown with double circles) for different target labels. We consider the case where label i is entire and j is not. The graph $G_{t'}$ admits 4 happy vertices if both i and j are target labels. However, this is no longer true after the join because label j is not entire. Instead, the optimal happy set for G_t can be found where label i is excluded from the target labels for $G_{t'}$, i.e. $\phi[t', 5, \{i\}, \{j\}]$, which admits 2 happy vertices with label j . Notice that we do not count the happy vertices with label i if it is not a target.

For the union node formula, observe that X can be determined by the choice of X_1 and X_2 , so it is enough to consider all possible values for $w, T, \tilde{w}, X_1, X_2$, which results in the running time $\mathcal{O}\left((2^{cw})^3 \cdot k^2\right)$, or $\mathcal{O}(8^{cw} \cdot k^2)$. ◀

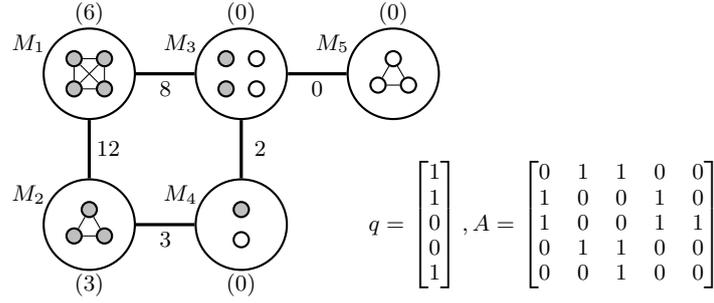
This completes the proof of Theorem 19, as we process each node of the cw-expression tree once, and it has $\mathcal{O}(|V(G)|)$ nodes.

5 Algorithms for Maximum Edge Happy Set

In addition to MAXHS, we also study its edge-variant MAXEHS. One difference from MAXHS is that when joining two subgraphs, we may increase the number of edges between those subgraphs, even if they are not entire. In other words, the number of edges between joining subgraphs depends on two variables, and quadratic programming naturally takes part in this setting. Here, we present FPT algorithms for two parameters – neighborhood diversity and cluster deletion number – to figure out the boundary between parameters that are FPT (e.g. treewidth) and W[1]-hard (e.g. clique-width) (see Figure 2).

5.1 Parameterized by neighborhood diversity

As shown in Figure 1, neighborhood diversity is a parameter specializing modular-width. To obtain a finer classification of structural parameters, we now show MAXEHS is FPT parameterized by neighborhood diversity.



■ **Figure 8** Example instance of MAXEHS with $\text{nd} = 5$. The figure shows the quotient graph H of the given graph G on its modules M_1, \dots, M_5 . Every edge in H forms a biclique in G . The maximum edge happy set for $k = 10$ are shaded in gray. It also shows the number of internal edges for each module (e.g. (6) for M_1), and that of external edges between modules (e.g. 12 between M_1 and M_2). Vector q indicates if each module is a clique or an independent set (e.g. $q_1 = 1$ because M_1 is a clique), and A is the adjacency matrix of the quotient graph.

Let nd be the neighborhood diversity of the given graph G . We observe that any instance (G, k) of MAXEHS can be reduced to the instance of INTEGER QUADRATIC PROGRAMMING (IQP) as follows.

▶ **Lemma 25.** MAXEHS can be reduced to IQP with $\mathcal{O}(\text{nd})$ variables and bounded coefficients in time $\mathcal{O}(|V(G)| + |E(G)|)$.

We defer our proof to Appendix. Figure 8 exemplifies a quotient graph of a graph with $\text{nd} = 5$, along with vector q and matrix A . The following is a direct result from Lemma 25 and Proposition 14.

▶ **Theorem 26.** MAXEHS can be solved in time $f(\text{nd}) \cdot |V(G)|^{\mathcal{O}(1)}$, where nd is the neighborhood diversity of the input graph G and f is a computable function.

5.2 Parameterized by cluster deletion number

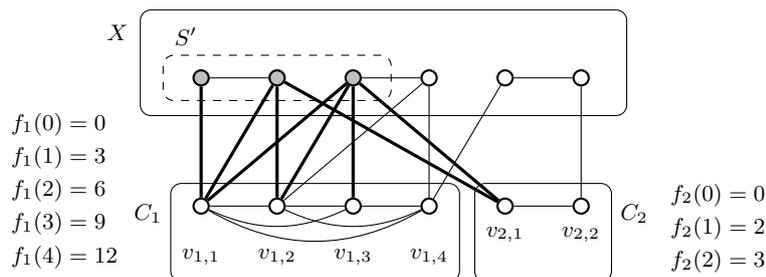
Finally, we present an FPT algorithm for MAXEHS parameterized by the cluster deletion number of the given graph.

▶ **Theorem 27.** Given a graph $G = (V, E)$ and its cluster deletion set X of size cd , MAXEHS can be solved in time $\mathcal{O}(2^{\text{cd}} \cdot k^2 \cdot |V|)$.

Recall that by definition, $G[V \setminus X]$ is a set of disjoint cliques. Let C_1, \dots, C_p be the clusters appeared in $G[V \setminus X]$. Our algorithm first guesses part of the happy set S' , defined as $S \cap X$, and performs f -KNAPSACK with p items.

▶ **Algorithm 2 (MaxEHS-CD).** Given a graph $G = (V, E)$ and its cluster deletion set X , consider all sets of $S' \subseteq X$ such that $|S'| \leq k$ and proceed the following steps.

(Step 1) For each clique C_i , sort its vertices in non-increasing order of the number of neighbors in S' . Let $v_{i,1}, \dots, v_{i,|C_i|}$ be the ordered vertices in C_i . (Step 2) For each $1 \leq i \leq p$, construct a function f_i as follows: $f_i(0) = 0$ and for every $1 \leq j \leq |C_i|$, $f_i(j) = f_i(j-1) + |N(v_{i,j}) \cap S'| + j - 1$. (Step 3) Formulate an f -KNAPSACK instance with capacity $k - |S'|$ and value functions f_i for every $1 \leq i \leq p$. Then, obtain the solution $\{x_i\}$ with the exact capacity $k - |S'|$ if feasible. (Step 4) Construct S as follows: Initialize with S' and for each clique C_i , pick x_i vertices in order and include them in S . That is, update $S \leftarrow S \cup \{v_{i,1}, \dots, v_{i,x_i}\}$ for every $1 \leq i \leq p$. Finally, return S that maximizes $|E(G[S])|$.



■ **Figure 9** Visualization of MaxEHS-CD, given a graph $G = (V, E)$ with its cluster deletion set X and a fixed partial solution S' ($|S'| = 3$, shaded in gray). The graph after removing X , i.e. $G[V \setminus X]$, forms cliques C_1 and C_2 . For each clique, vertices are sorted in decreasing order of the number of neighbors in S' (edges to S' in thicker lines). Functions f_1 and f_2 are constructed as described in the algorithm and used for f -KNAPSACK. For example, $f_1(3) = f_1(2) + 1 + (3 - 1)$ as vertex $v_{1,3}$ has one edge to S' and two edges to previously-added $v_{1,1}$ and $v_{1,2}$. If $k = 6$, then we pick $k - |S'| = 3$ vertices from C_1 and C_2 . The optimal solution would be $\{v_{1,1}, v_{1,2}, v_{1,3}\}$ because $f_1(3) + f_2(0) = 9$ gives the maximum objective value in the f -KNAPSACK formulation.

Intuitively, we construct function f_i in a greedy manner. When we add a vertex v in clique C_i to the happy set S , it will increase the number of happy edges by the number of v 's neighbors in S' and the number of the vertices in C_i that are already included in S . Therefore, it is always advantageous to pick a vertex having the most neighbors in S' . Figure 9 illustrates the key ideas of Algorithm 2. The following proposition completes the proof of Theorem 27.

► **Proposition 28.** *Given a graph $G = (V, E)$ and its cluster deletion set X of size cd , Algorithm 2 correctly finds the maximum edge happy set in time $\mathcal{O}(2^{cd} \cdot k^2 \cdot |V|)$.*

Proof. The algorithm considers all possible sets of $S \cap X$, so the optimal solution should extend one of them. It is clear to see that when the f -KNAPSACK instance is feasible, S ends up with k vertices, since the sum of the obtained solution must be $k - |S'|$. The objective of the f -KNAPSACK is equivalent to $|E(G[S])| - |E(G[S'])|$, that is, the number of happy edges extended by the vertices in $V \setminus X$. Since S' has been fixed at this point, the optimal solution to f -KNAPSACK leads to that to MAXEHS. Lastly, the value function f_i is correct because for each clique C_i , the number of extended edges is given by $\binom{|S_i|}{2} + \sum_{v \in S_i} |N(v) \cap S'|$, where $S_i = S \cap C_i$ and $x_i = |S_i|$. This is maximized by choosing $|S_i|$ vertices that have the most neighbors in S' , if we fix $|S_i|$, represented as x_i in f -KNAPSACK. This is algebraically consistent with the recursive form in step 2.

For the running time, the choice of S' adds the complexity of 2^{cd} to the entire algorithm. Having chosen S' , vertex sorting (step 1) can be accomplished by checking the edges between S' and $V \setminus X$, so it takes only $\mathcal{O}(k \cdot |V|)$. The f -KNAPSACK (step 3) takes time $\mathcal{O}(pk^2) = \mathcal{O}(k^2 \cdot |V|)$ from Corollary 13, because there are p items and weights are bounded by k . Steps 2 and 4 do not exceed this asymptotic running time. The total runtime is $\mathcal{O}(2^{cd} \cdot k^2 \cdot |V|)$. ◀

6 Conclusions & Future Work

We present four algorithms using a variety of techniques, two for MAXIMUM HAPPY SET (MAXHS) and two for MAXIMUM EDGE HAPPY SET (MAXEHS). The first shows that MAXHS is FPT with respect to the modular-width parameter, which is stronger than clique-width but generalizes several parameters such as neighborhood diversity and twin cover

number. We then give an FPT dynamic-programming algorithm for MAXHS parameterized by clique-width. This improves the best known complexity result of FPT when parameterized by clique-width plus k .

For MAXEHS, we prove that it is FPT by neighborhood diversity, using INTEGER QUADRATIC PROGRAMMING. Lastly, we show an FPT algorithm parameterized by cluster deletion number, the distance to a cluster graph, which then implies FPT by twin cover number. These results have answered several open questions of [1] (Figure 2). While it is FPT, there cannot be a polynomial kernel with respect to nd and cd , due to the lower-bounds on CLIQUE parameterized by vertex cover number, unless $NP \subseteq coNP/poly$ [2].

There are multiple potential directions for future research. As highlighted in Figure 2, the parameterized complexity of MAXEHS with respect to modular-width is still open. Another direction would be to find the lower bounds for known algorithms.

References

- 1 Yuichi Asahiro, Hiroshi Eto, Tesshu Hanaka, Guohui Lin, Eiji Miyano, and Ippei Terabaru. Parameterized algorithms for the happy set problem. *Discrete Applied Mathematics*, 304:32–44, 2021.
- 2 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014.
- 3 Nicolas Bourgeois, Aristotelis Giannakos, Giorgio Lucarelli, Ioannis Milis, and Vangelis Th. Paschos. Exact and approximation algorithms for densest k -subgraph. In *7th International Workshop on Algorithms and Computation (WALCOM 2013)*, volume 7748, pages 114–125, 2013.
- 4 Hajo Broersma, Petr A. Golovach, and Viresh Patel. Tight complexity bounds for fpt subgraph problems parameterized by the clique-width. *Theoretical Computer Science*, 485:69–84, 2013.
- 5 Maurizio Bruglieri, Matthias Ehrgott, Horst W. Hamacher, and Francesco Maffioli. An annotated bibliography of combinatorial optimization problems with fixed cardinality constraints. *Discrete Applied Mathematics*, 154(9):1344–1357, 2006.
- 6 Leizhen Cai. Parameterized Complexity of Cardinality Constrained Optimization Problems. *The Computer Journal*, 51(1):102–121, 2007.
- 7 D. G. Corneil and Y. Perl. Clustering and domination in perfect graphs. *Discrete Applied Mathematics*, 9(1):27–39, 1984.
- 8 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1):77–114, 2000.
- 9 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.
- 10 Reinhard Diestel. Graph theory. fifth. vol. 173. *Graduate Texts in Mathematics*. Springer, Berlin, 2018.
- 11 David Easley, Jon Kleinberg, et al. Networks, crowds, and markets: Reasoning about a highly connected world. *Significance*, 9(1):43–44, 2012.
- 12 Uriel Feige and Michael Seltser. On the Densest K -Subgraph Problem. *Algorithmica*, 29:2001, 1997.
- 13 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Clique-width: On the Price of Generality. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 825–834. Society for Industrial and Applied Mathematics, 2009.
- 14 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Algorithmic lower bounds for problems parameterized by clique-width. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete algorithms*, SODA '10, pages 493–502, USA, 2010. Society for Industrial and Applied Mathematics.

- 15 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of Clique-Width Parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010.
- 16 Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.
- 17 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In Gregory Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation*, pages 163–176. Springer International Publishing, 2013.
- 18 G. Gallo, P. L. Hammer, and B. Simeone. Quadratic knapsack problems. In M. W. Padberg, editor, *Combinatorial Optimization*, Mathematical Programming Studies, pages 132–149. Springer, Berlin, Heidelberg, 1980.
- 19 Robert Ganian. Improving Vertex Cover as a Graph Parameter. *Discrete Mathematics & Theoretical Computer Science*, Vol. 17 no.2, 2015.
- 20 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1979.
- 21 Tesshu Hanaka. Computing Densest k -Subgraph with Structural Parameters, 2022. [arXiv:2207.09803](https://arxiv.org/abs/2207.09803).
- 22 G. Kortsarz and D. Peleg. On choosing a dense subgraph. *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, 1993.
- 23 Michael Lampis. Algorithmic Meta-theorems for Restrictions of Treewidth. *Algorithmica*, 64(1):19–37, 2012.
- 24 Angsheng Li and Pan Peng. Community Structures in Classical Network Models. *Internet Mathematics*, 7(2), 2011.
- 25 Daniel Lokshtanov. Parameterized integer quadratic programming: Variables and coefficients, 2015. [arXiv:1511.00310](https://arxiv.org/abs/1511.00310).
- 26 Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001.
- 27 Hannes Moser. Exact algorithms for generalizations of vertex cover. *Institut für Informatik, Friedrich-Schiller-Universität Jena*, 2005.
- 28 Neil Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms. Academic Press Inc.*, 7(3):309–322, 1986.
- 29 Marc Tedder, Derek Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *Automata, Languages and Programming*, pages 634–645, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- 30 Peng Zhang and Angsheng Li. Algorithmic aspects of homophily of networks. *Theoretical Computer Science*, 593:117–131, 2015.

A Appendix

Here we present a proof of Lemma 25 on a reduction from MAXEHS to IQP.

Proof. First, we compute the set of twins (modules) $\mathcal{M} = M_1, \dots, M_{nd}$ of G , and obtain the quotient graph H on the modules \mathcal{M} in time $\mathcal{O}(|V(G)| + |E(G)|)$ [23]. Note that each module M_i is either a clique or an independent set. Let us define a vector $q \in \mathbb{Z}^{nd}$ such that $q_i = 1$ if M_i is a clique, and $q_i = 0$ if M_i is an independent set. Further, let $A \in \mathbb{Z}^{nd \times nd}$ be the adjacency matrix of H where $A_{ij} = 1$ if $M_i M_j \in E(H)$, and $A_{ij} = 0$ otherwise.

We then formulate an IQP instance as follows:

- Variables: $x \in \mathbb{Z}^{nd}$.
- Maximize: $f(x) = x^T(A + qq^T)x - q^T x$ (equivalently, minimize $-f(x)$).
- Subject to: $\sum_i x_i = k$ and $0 \leq x_i \leq |M_i|$ for every $1 \leq i \leq nd$.

23:18 Parameterized Complexity of Maximum Happy Set and Densest k -Subgraph

This formulation has nd variables, and its coefficients are either 0 or ± 1 , thus bounded. After finding the optimal vector x , pick any x_i vertices from module M_i and include them in the happy set S . We claim that S maximizes the number of happy edges.

For any happy set S , the number of happy edges is given by the sum of the number of happy edges inside each module M_i , which we call *internal edges*, and the number of edges between each module pair M_i and M_j , or *external edges*. Let $x \in \mathbb{Z}^{nd}$ be a vector such that $x_i = |S \cap M_i|$ for every i . Then, the number of internal edges of module M_i is $q_i \cdot \binom{x_i}{2}$, and the number of external edges between modules M_i and M_j is $A_{ij} \cdot x_i x_j$. The number of happy edges, i.e. $|E(G[S])|$, is given by: $h(x) = \left[\sum_{i=1}^{nd} q_i \cdot \binom{x_i}{2} \right] + \left[\sum_{1 \leq i < j \leq nd} A_{ij} \cdot x_i x_j \right]$. One can trivially verify $f(x) = 2h(x)$.

If the IQP instance is feasible, then we can find a happy set S of size k maximizing $h(x)$, which must be the optimal solution to MAXEHS. Otherwise, $\sum_i |M_i| = |V(G)| < k$, and MAXEHS is also infeasible. \blacktriangleleft

Parameterized Complexity of Streaming Diameter and Connectivity Problems

Jelle J. Oostveen 

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Erik Jan van Leeuwen 

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Abstract

We initiate the investigation of the parameterized complexity of DIAMETER and CONNECTIVITY in the streaming paradigm. On the positive end, we show that knowing a vertex cover of size k allows for algorithms in the Adjacency List (AL) streaming model whose number of passes is constant and memory is $\mathcal{O}(\log n)$ for any fixed k . Underlying these algorithms is a method to execute a breadth-first search in $\mathcal{O}(k)$ passes and $\mathcal{O}(k \log n)$ bits of memory. On the negative end, we show that many other parameters lead to lower bounds in the AL model, where $\Omega(n/p)$ bits of memory is needed for any p -pass algorithm even for constant parameter values. In particular, this holds for graphs with a known modulator (deletion set) of constant size to a graph that has no induced subgraph isomorphic to a fixed graph H , for most H . For some cases, we can also show one-pass, $\Omega(n \log n)$ bits of memory lower bounds. We also prove a much stronger $\Omega(n^2/p)$ lower bound for DIAMETER on bipartite graphs.

Finally, using the insights we developed into streaming parameterized graph exploration algorithms, we show a new streaming kernelization algorithm for computing a vertex cover of size k . This yields a kernel of $2k$ vertices (with $\mathcal{O}(k^2)$ edges) produced as a stream in $\text{poly}(k)$ passes and only $\mathcal{O}(k \log n)$ bits of memory.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Streaming, sublinear and near linear time algorithms; Theory of computation \rightarrow Lower bounds and information complexity

Keywords and phrases Stream, Streaming, Graphs, Parameter, Complexity, Diameter, Connectivity, Vertex Cover, Disjointness, Permutation

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.24

Related Version *Full Version:* <https://arxiv.org/abs/2207.04872> [47]

Funding *Jelle J. Oostveen:* This author is partially supported by the NWO grant OCENW.KLEIN.114 (PACAN).

Acknowledgements We thank the reviewers for their helpful comments and feedback.

1 Introduction

Graph algorithms, such as to compute the diameter of an unweighted graph (DIAMETER) or to determine whether it is connected (CONNECTIVITY), often rely on keeping the entire graph in (random access) memory. However, very large networks might not fit in memory. Hence, graph streaming has been proposed as a paradigm where the graph is inspected through a so-called stream, in which its edges appear one by one [38]. To compensate for the assumption of limited memory, multiple passes may be made over the stream and computation time is assumed to be unlimited. The complexity theory question is which problems remain solvable and which problems are hard in such a model, taking into account trade-offs between the amount of memory and passes.



© Jelle J. Oostveen and Erik Jan van Leeuwen;

licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 24; pp. 24:1–24:16

Leibniz International Proceedings in Informatics



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

Many graph streaming problems require $\Omega(n)$ bits of memory [33, 34] for a constant number of passes on n -vertex graphs. Any p -pass algorithm for CONNECTIVITY needs $\Omega(n/p)$ bits of memory [38]. Single pass algorithms for CONNECTIVITY or DIAMETER need $\Omega(n \log n)$ bits of memory on sparse graphs [50]. A 2-approximation of DIAMETER requires $\Omega(n^{3/2})$ bits of memory on weighted graphs [34]. A naive streaming algorithm for CONNECTIVITY or DIAMETER stores the entire graph, using $\mathcal{O}(m \log n) = \mathcal{O}(n^2 \log n)$ bits and a single pass. For CONNECTIVITY, union-find yields a 1-pass, $\mathcal{O}(n \log n)$ bits of memory, algorithm [43].

An intriguing aspect on DIAMETER and CONNECTIVITY is that some classic algorithms for them rely on breadth-first search (BFS) or depth-first search (DFS). These seem difficult to execute efficiently in a streaming setting. It was a longstanding open problem to compute a DFS tree using $o(n)$ passes and $o(m \log n)$ bits of memory. This barrier was recently broken [42], through an algorithm that uses $\mathcal{O}(n/k)$ passes and $\mathcal{O}(nk \log n)$ bits of memory, for any k . The situation for computing single-source shortest paths seems similar [30], although good approximations exist even on weighted graphs (see e.g. [43, 31]). We do know that DFS algorithms cannot be executed in logarithmic space [48]. In streaming, any BFS algorithm that explores k layers of the BFS tree must use at least $k/2$ passes or $\Omega(n^{1+1/k}/(\log n)^{1/k})$ space [34]. Hence, much remains unexplored when it comes to graph exploration- and graph distance-related streaming problems such as BFS/DFS, DIAMETER, and CONNECTIVITY. In particular, most lower bounds hold for general graphs. As such, a more fine-grained view of the complexity of these problems has so far been lacking.

In this paper, we seek to obtain this fine-grained view using parameterized complexity [25]. The idea of using parameterized complexity in the streaming setting was first introduced by Fafianie and Kratsch [32] and Chitnis et al. [21]. Many problems are hard in streaming parameterized by their solution size [32, 21, 18]. Crucially, however, deciding whether a graph has a vertex cover of size k has a one-pass, $\tilde{\mathcal{O}}(k^2)$ -memory kernelization algorithm by Chitnis et al. [19], and a 2^k -passes, $\tilde{\mathcal{O}}(k)$ -memory direct algorithm by Chitnis and Cormode [18].

Bishnu et al. [10] then showed that knowing a vertex cover of size k is useful in solving other deletion problems using $p(k)$ passes and $f(k) \log n$ memory, notably H -free deletion; this approach was recently expanded on by Oostveen and van Leeuwen [46]. This leads to the more general question how knowing a (small) H -free modulator, that is, a set X such that $G - X$ has no induced subgraph isomorphic to H (note that $H = P_2$ in VERTEX COVER [18]), would affect the complexity of streaming problems and of BFS/DFS, DIAMETER, and CONNECTIVITY in particular. We are not aware of any investigations in this direction.

An important consideration is the streaming model (see [38, 36, 41, 44] or the survey by McGregor [43]). In the Edge Arrival (EA) model, each edge of the graph appears once in the stream, and the edges appear in some fixed but arbitrary order. Most aforementioned results use this model. In the Vertex Arrival (VA) model the edges arrive grouped per vertex, and an edge is revealed at its endpoint that arrives latest. In the Adjacency List (AL) model the edges also arrive grouped per vertex, but each edge is present for both its endpoints. This means we see each edge twice and when a vertex arrives we immediately see all its adjacencies (rather than some subset dependent on the arrival order, as in the VA model). This model is quite strong, but as we shall see, unavoidable for our positive results. We do not consider dynamic streaming models in this paper, although they do exist.

Our Contributions. The main takeaway from our work is that the vertex cover number likely sits right at the frontier of parameters that are helpful in computing DIAMETER and CONNECTIVITY. As our main positive result, we show the following.

¹ See Section 2 for the notation.

► **Theorem 1.** *Given a vertex cover of size k , DIAMETER $[k]$ and CONNECTIVITY $[k]$ can be solved using $\mathcal{O}(2^k k)$ passes and $\mathcal{O}(k \log n)$ bits of space or using one pass and $\mathcal{O}(2^k + k \log n)$ bits of space, in the AL model.*

The crux to our approach is to perform a BFS in an efficient manner, using $\mathcal{O}(k)$ passes and $\mathcal{O}(k \log n)$ space. Knowledge of a vertex cover is not a restricting assumption, as one may be computed using similar memory requirements [19, 18]. An extension allows the one-pass result to work without a vertex cover being given, at the cost of increasing the memory use to $\mathcal{O}(4^k + k \log n)$ bits of space.

As a contrasting result, we observe that in the VA model, even a constant-size vertex cover does not help in computing DIAMETER and CONNECTIVITY. Moreover, the bound on the vertex cover seems necessary, as we can prove that any p -pass algorithm for DIAMETER requires $\Omega(n^2/p)$ bits of memory even on bipartite graphs and any p -pass algorithm for CONNECTIVITY requires $\Omega(n/p)$ bits of memory, both in the AL model. This indicates that both the permissive AL model and a low vertex cover number are truly needed.

In some cases, we are also able to prove that a single-pass algorithm requires $\Omega(n \log n)$ bits of memory (due to lack of space these proofs are fully deferred to the full version of the paper [47]).

More broadly, knowledge of being H -free (that is, not having a fixed graph H as an induced subgraph) or having an H -free modulator does not help even in the AL model:

► **Theorem 2.** *For any fixed graph H with $H \not\subseteq_i P_4$ (H is not an induced subgraph of P_4) and $H \neq 3P_1, P_3 + P_1, P_2 + 2P_1$, any streaming algorithm for DIAMETER in the AL model that uses p passes over the stream must use $\Omega(n/p)$ bits of memory on graphs G even when G is H -free.*

We note that these results hold for H -free graphs (without the need for a modulator). The case when $H \subseteq_i P_4$ is straightforward to solve with $\mathcal{O}(\log n)$ bits of memory, as the diameter is either 1 or 2 (an induced path of length 3 is a P_4). If the graph has diameter 1, it is a clique. This can be tested in a single pass by counting the number of edges.

► **Theorem 3.** *For any fixed graph H with $H \neq P_2 + sP_1$ for $s \in \{0, 1, 2\}$ and $H \neq sP_1$ for $s \in \{1, 2, 3\}$, any streaming algorithm for DIAMETER in the AL model that uses p passes over the stream must use $\Omega(n/p)$ bits of memory on graphs G even when given a set $X \subseteq V(G)$ of constant size such that $G - X$ is H -free. If $G - X$ must be connected and H -free, then additionally $H \neq P_3$.*

We note that the case when $H = P_2$ or $H = P_1$ is covered by Theorem 1. Cobipartite graphs seem to be a bottleneck class. The cases when $H = 2P_1$ or when $H = P_3$ and $G - X$ must be connected lead to a surprising second positive result.

► **Theorem 4.** *Given a set X of size k such that $G - X$ is a disjoint union of ℓ cliques, DIAMETER $[k, \ell]$ and CONNECTIVITY $[k, \ell]$ can be solved using $\mathcal{O}(2^k k \ell)$ passes and $\mathcal{O}((k + \ell) \log n)$ bits of space or one pass and $\mathcal{O}(2^k \ell + (k + \ell) \log n)$ bits of space, in the AL model.*

The approach for this result is similar as for Theorem 1. Moreover, we show a complementary lower bound in the VA model, even for $\ell = 1$ and constant k .

Our results for DIAMETER are summarized in Table 1 and 2. In words, generalizing Theorem 1 using the perspective of an H -free modulator does not seem to lead to a positive result (Theorem 4). Instead, connectivity of the remaining graph after removing the modulator seems crucial. However, this perspective only helps for Theorem 4, while the problem remains

hard for most other H -free modulators and even for the seemingly simple case of a modulator to a path. While Theorem 4 would also hint at the possibility of using a modulator to a few components of small diameter, this also leads to hardness.

We emphasize that all instances of DIAMETER in our hardness reductions are connected graphs. Hence, the hardness of computing DIAMETER is separated from the hardness of computing CONNECTIVITY.

For CONNECTIVITY, we also give two broad theorems that knowledge of being H -free or having an H -free modulator does not help even in the AL model.

► **Theorem 5.** *For any fixed graph H that is not a linear forest containing only paths of length at most 5, any streaming algorithm for CONNECTIVITY in the AL model that uses p passes over the stream must use $\Omega(n/p)$ bits of memory on graphs G even when G is H -free.*

► **Theorem 6.** *For any fixed graph H that is not a linear forest containing only paths of length at most 1, any streaming algorithm for CONNECTIVITY in the AL model that uses p passes over the stream must use $\Omega(n/p)$ bits of memory on graphs G even when given a set $X \subseteq V(G)$ of constant size such that $G - X$ is H -free.*

As a final result, we use our insights into graph exploration on graphs of bounded vertex cover to show a result on the VERTEX COVER problem itself. In particular, a kernel on $2k$ vertices for VERTEX COVER $[k]$ can be obtained as a stream in $\mathcal{O}(k^3)$ passes in the EA model using only $\tilde{\mathcal{O}}(k)$ bits of memory. In the AL model, the number of passes is only $\mathcal{O}(k^2)$. This kernel still may have $\mathcal{O}(k^2)$ edges, which means that saving it in memory would not give a better result than that of Chitnis et al. [19] (which uses $\tilde{\mathcal{O}}(k^2)$ bits of memory). Indeed, a better kernel seems unlikely to exist [24]. However, the important point is that storing the (partial) kernel in memory is not needed during its computation. Hence, it may be viewed as a possible first step towards a streaming algorithm for VERTEX COVER $[k]$ using $\tilde{\mathcal{O}}(k)$ bits of memory and $\text{poly}(k)$ passes, which is an important open problem in the field, see [18]. Our kernel is constructed through a kernel by Buss and Goldsmith [14], and then finding a maximum matching in an auxiliary bipartite graph (following Chen et al. [16]) of bounded size through repeated DFS applications.

Related work. There has been substantial work on the complexity of graph-distance and reachability problems in the streaming setting. For example, Guruswami and Onak [37] showed that any p -pass algorithm needs $n^{1+\Omega(1/p)}/p^{\mathcal{O}(1)}$ memory when given vertices s, t to test if s, t are at distance at most $2p + 2$ in undirected graphs or to test s - t reachability in directed graphs. Further work on directed s - t reachability [6] recently led to a lower bound that any $o(\sqrt{\log n})$ -pass algorithm needs $n^{2-o(1)}$ bits of memory [17]. Other recent work considers p -pass algorithms for ϵ -property testing of connectivity [51, 39, 4], including strong memory lower bounds $n^{1-\mathcal{O}(\epsilon/p)}$ on bounded-degree planar graphs [5]. Further problems in graph streaming are extensively discussed and referenced in these works; see also [3].

In the non-streaming setting, the DIAMETER problem can be solved in $\mathcal{O}(nm)$ time by BFS. There is a lower bound of $n^{2-\epsilon}$ for any $\epsilon > 0$ under the Strong Exponential Time Hypothesis (SETH) [49]. Parameterizations of DIAMETER have been studied with parameter vertex cover [13], treewidth [1, 40, 13], and other parameters [23, 8], leading to a $2^{\mathcal{O}(k)}n^{1+\epsilon}$ time algorithm on graphs of treewidth k [13]. Running time $2^{o(k)}n^{2-\epsilon}$ for graphs of treewidth k is not possible under SETH [1]. Subquadratic algorithms are known for various hereditary graph classes; see e.g. [15, 22, 26, 27, 28, 29, 35] and references in [22].

2 Preliminaries

We work on undirected, unweighted graphs. We denote a computational problem A with a parameterization $A[k]$, where $[.]$ denotes the parameterization. The default parameter is solution size, if not mentioned otherwise. DIAMETER is to compute $\max_{s,t \in V} d(s,t)$ where $d(s,t)$ denotes the distance between s and t . CONNECTIVITY asks to decide whether or not the graph is connected. A *twin class* consists of all vertices with the same open neighbourhood. In a graph with vertex cover size k , we have $\mathcal{O}(2^k)$ twin classes. For two graphs G, H , $G + H$ denotes their disjoint union. We also use $2G$ to denote $G + G$; $3G$ is $G + G + G$, etc. A *linear forest* is a disjoint union of paths. A path on a vertices is denoted P_a and has length $a - 1$.

We employ the following problem in communication complexity.

DISJ $_n$ (Disjointness)

Input: Alice has a string $x \in \{0,1\}^n$ given by $x_1x_2 \dots x_n$. Bob has a string $y \in \{0,1\}^n$ given by $y_1y_2 \dots y_n$.

Question: Bob wants to check if $\exists 1 \leq i \leq n$ such that $x_i = y_i = 1$. (Formally, the answer is NO if this is the case.)

The communication complexity necessary between Alice and Bob to solve this problem is well understood, and can be used to prove lower bounds on the memory use of streaming algorithms. This was first done by Henzinger et al. [38]. The following formulation by Bishnu et al. [9] comes in very useful.

► **Proposition 7** (Rephrasing of item (ii) of [9], Proposition 5.6). *If we can show a reduction from DISJ $_n$ to problem Π in streaming model \mathcal{M} such that in the reduction, Alice and Bob construct one model- \mathcal{M} pass for a streaming algorithm for Π by communicating the memory state of the algorithm only a constant number of times to each other, then any streaming algorithm working in the model \mathcal{M} for Π that uses p passes requires $\Omega(n/p)$ bits of memory, for any $p \in \mathbb{N}$ [20, 11, 2].*

If we can show a reduction from DISJOINTNESS, we call a problem “hard”, as it does not admit algorithms using only poly-logarithmic memory.

Any upper bound for the EA model holds for all models, and an upper bound for the VA model also holds for the AL model. On the other hand, a lower bound in the AL model holds for all models, and a lower bound for the VA model also holds for the EA model.

3 Upper Bounds for Diameter

We give an overview of our upper bound results for DIAMETER in Table 1. The memory-efficient results rely on executing a BFS on the graph, which is made possible by both the parameter and the use of the AL model. The one-pass results rely on the possibility to save the entire graph in a bounded fashion. Our upper bounds assume the deletion set related to the parameter is given, that is, it is in memory.

► **Lemma 8** (♣). *In a graph with vertex cover size k , any simple path has length at most $2k$.*

(Further discussions and proofs for results marked with ♣ appear in the full online version of the paper [47].)

Lemma 8 is useful in that the diameter of such a graph can be at most $2k$ if the graph is connected. Our algorithm will simulate a BFS for $2k$ rounds, deciding on the distance of a vertex to all other vertices.

24:6 Parameterized Complexity of Streaming Diameter and Connectivity Problems

■ **Table 1** Overview of the algorithms and their complexity for DIAMETER and CONNECTIVITY. The results for the VERTEX COVER parameter are given in Theorems 10,11.

Parameter (k)	Passes	Memory (bits)	Model
VERTEX COVER	$\mathcal{O}(2^k k)$	$\mathcal{O}(k \log n)$	AL
	1	$\mathcal{O}(2^k + k \log n)$	AL
DISTANCE TO ℓ CLIQUES	$\mathcal{O}(2^k \ell k)$	$\mathcal{O}((k + \ell) \log n)$	AL
	1	$\mathcal{O}(2^k \ell + (k + \ell) \log n)$	AL

► **Lemma 9.** *Given a graph G as an AL stream with a vertex cover X of size k , we can compute the distance from a vertex v to all others using $\mathcal{O}(k)$ passes and $\mathcal{O}(k \log n)$ bits of memory.*

Proof. We simulate a BFS originating at v for at most $2k$ rounds on our graph, using a pass for each round. Contrary to a normal BFS, we only remember whether we visited the vertices in the vertex cover and their distances, to reduce memory complexity.

For every vertex $w \in X$, we save its tentative distance $d(w)$ from v ; if this is not yet decided, this field has value ∞ . Our claim will be that after round i , the value of $d(w)$ for vertices w within distance i from v is correct. We initialize the distance of v as $d(v) = 0$ (we store $d(v)$ regardless of whether $v \in X$).

Say we are in round $i \geq 1$. We execute a pass over the stream. Say we view a vertex $w \in X \cup \{v\}$ in the stream with its adjacencies. If w has a distance of $d(w)$, we update the neighbours of w in X to have distance $d(u) = \min(d(u), d(w) + 1)$. If instead we view a vertex $w \notin X \cup \{v\}$ in the stream, we do the following. Locally save all the neighbours and look at their distances, and let z be the neighbour with minimum $d(z)$ value. For every $u \in N(w)$ we update the distance as $d(u) = \min(d(z) + 2, d(u))$. This simulates the distance of a path passing through w (note that this may not be the shortest path, but this may be resolved by other vertices). This completes the procedure for round i .

Notice that we use only $\mathcal{O}(k \log n)$ bits of memory during the procedure, and that the total number of passes is indeed $\mathcal{O}(k)$ as we execute $2k$ rounds, using one pass each.

For the correctness, let us first argue the correctness of the claim *after round i , the value of $d(w)$ of vertices $w \in X$ within distance i from v is correct*. We proceed by induction, clearly the base case of 0 is correct. Now consider some vertex w at distance i from v . Consider a shortest path from v to w . Look at the last vertex on the path before visiting w . If this vertex is in X , then by induction, this vertex has a correct distance after round $i - 1$, and so, in round i this vertex will update the distance of w to be i . If this vertex is not in X , then it has a neighbour with distance $i - 2$, which is correct after round $i - 2$ by induction, and so, the vertex not in X will (have) update(d) the distance of w to be i in round i .

The correctness of the algorithm now follows from the claim, together with Lemma 8, and the fact that we can now output all distances using a single pass by either outputting the value of the field $d(w)$ for a vertex $w \in X$, or by looking at all neighbours of a vertex $w \notin X$ and outputting the smallest value $+1$. ◀

Related is a lower bound result by Feigenbaum et al. [34], which says that any BFS procedure that explores k layers of the BFS tree must use at least $k/2$ passes or super-linear memory. This indicates that memory- and pass-efficient implementations of BFS, as in Lemma 9, are hard to come by.

We can now use Lemma 9 to construct an algorithm for finding the diameter of a graph parameterized by vertex cover, essentially by executing Lemma 9 for every twin class, which considers all options for vertices in the graph.

► **Theorem 10** (♣). *Given a graph G as an AL stream with vertex cover X of size k , we can solve DIAMETER $[k]$ in $\mathcal{O}(2^k k)$ passes and $\mathcal{O}(k \log n)$ bits of memory.*

We show an alternative one-pass algorithm, by saving the graph as a representation by its twin classes, thereby completing the proof of Theorem 1.

► **Theorem 11** (♣). *Given a graph G as an AL stream, we can solve DIAMETER $[k]$ in one pass and $\mathcal{O}(4^k + k \log n)$ bits of memory, or correctly report that a vertex cover of size k does not exist. When a vertex cover of size k is given, the memory use is $\mathcal{O}(2^k + k \log n)$.*

The ideas of Theorem 10 and Theorem 11 also work for a similar setting with a few adjustments. This is the setting of Theorem 4, that our problem is parameterized by DISTANCE TO ℓ CLIQUES, where both the deletion distance k and the number of remaining cliques ℓ are bounded. The BFS idea works here as well, as shortest paths are of bounded length, and we can save information for every vertex in the deletion set, as well as some information for every clique. There is also a concept of twin classes in such an instance, where we also distinguish which clique a vertex belongs to, and this is useful for the BFS from “every” vertex, as well as a one-pass algorithm where we save the entire graph in a compressed representation. The details of the theorems and proofs that make up Theorem 4 are given in the full version (♣). When the number of cliques is not bounded, this setting admits a lower bound, which we will see in Section 4.

4 Lower Bounds for Diameter

We work with reductions from DISJ $_n$, and we construct graphs where Alice controls some of the edges, and Bob controls some of the edges, depending on their respective input of the DISJ $_n$ problem, and some parts of the graph are fixed. The aim is to create a gap in the diameter of the graph, that is, the answer to DISJ $_n$ is YES if and only if the diameter is above or below a certain value. The lower bound then follows from Proposition 7. Here n may be the number of vertices in the graph construction, but may also be different (possibly forming a different lower bound). Our lower bounds hold for connected graphs.

We start by proving simple lower bounds for the VA model when our problem is parameterized by the vertex cover number, and when our problem is parameterized by the distance to ℓ cliques. This shows that we actually need the AL model to achieve the upper bounds in Section 3. The constructions are illustrated in Figure 1 and Figure 2. Generally, a -vertices (b -vertices) and their incident edges are controlled by Alice (Bob). To give an idea of the reduction technique, we describe how a VA stream is constructed by Alice and Bob, in the construction of Figure 1. First, Alice reveals the middle vertices including the vertex c and the fixed edges, then reveals the vertex a with the edges dependent on her input. Then the memory state of the algorithm is given to Bob who can reveal his vertex b with the edges dependent on his input. Notice that this is a valid VA stream, and Alice and Bob need no information about the input of the other.

► **Theorem 12** (♣). *Any streaming algorithm for DIAMETER on graphs of vertex cover number at least 3 in the VA model that uses p passes over the stream requires $\Omega(n/p)$ bits of memory.*

► **Theorem 13** (♣). *Any streaming algorithm for DIAMETER on graphs of distance 2 to $\ell = 1$ clique in the VA model that uses p passes over the stream requires $\Omega(n/p)$ bits of memory.*

■ **Table 2** An overview of the lower bounds for DIAMETER, with the parameter (k) on the left. These results hold for connected graphs. (\mathcal{M}, m, p) -hard means that any algorithm using p passes in model \mathcal{M} (or weaker) requires $\Omega(m)$ bits of memory. FVS stand for Feedback Vertex Set number, FEN for Feedback Edge Set number. Most proofs of the results in this table are deferred to the full paper (♣).

Parameter (k) / Graph class	Size	Bound
General and Bipartite Graphs		(AL, $n^2/p, p$)-hard
VERTEX COVER	≥ 3	(VA, $n/p, p$)-hard
DISTANCE TO ℓ CLIQUES	$k \geq 2, \ell \geq 1$	(VA, $n/p, p$)-hard
FVS, FES	≥ 0	(AL, $n/p, p$)-hard
	≥ 0	(AL, $n \log n, 1$)-hard
DISTANCE TO MATCHING	≥ 3	(AL, $n/p, p$)-hard
DISTANCE TO PATH	≥ 2	(AL, $n/p, p$)-hard
	≥ 2	(AL, $n \log n, 1$)-hard
DISTANCE TO DEPTH ℓ TREE	$k \geq 3, \ell \geq 2$	(AL, $n/p, p$)-hard
	$k \geq 0, \ell \geq 5$	(AL, $n/p, p$)-hard
	$k \geq 0, \ell \geq 7$	(AL, $n \log n, 1$)-hard
DIST. TO ℓ COMPS. OF DIAM. x	$k, x \geq 2$	(AL, $n/p, p$)-hard
DOMINATION NUMBER	≥ 3	(AL, $n/p, p$)-hard
MAXIMUM DEGREE	≥ 3	(AL, $n/p, p$)-hard
	≥ 3	(AL, $n \log n, 1$)-hard
Split graphs		(AL, $n/p, p$)-hard

The lower bounds in Figure 1 and Figure 2 do not work for the AL model because there are vertices that may or may not be adjacent to both a and b , so neither Alice nor Bob can produce the adjacency list of such a vertex alone. For the “Simple VA” construction, we can “fix” this by extending these vertices to edges but this is destructive to the small vertex cover number of the construction. This “fixed” construction is fully deferred to the full version of the paper (♣). It should be clear that AL reductions require care: no vertex may be incident to variable edges of both Alice and Bob.

The following theorem is a combination of several constructions, implying AL hardness on trees, splits graphs, and for many deletion-distance-to- x parameters. An overview of all hardness results for DIAMETER is given in Table 2. See Figures 3, 4, 5 for illustrations of the constructions and an idea of the proof.

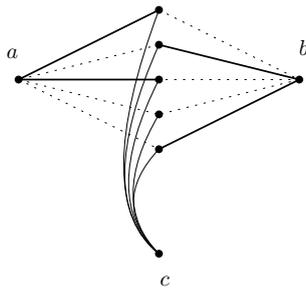
► **Theorem 14** (♣). *Any streaming algorithm for DIAMETER that works on a family of graphs that includes the “Windmill”, “Diamond”, or “Split” construction in the AL model using p passes over the stream requires $\Omega(n/p)$ bits of memory.*

We can now prove Theorem 2 and Theorem 3; full proofs are deferred (♣). Intuitively, if H contains a cycle or a vertex of degree 3, a modification of “Windmill” is H -free; if H is a linear forest, a modification of “Split” is (almost) H -free.

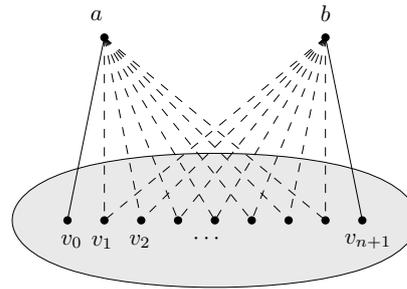
We can also prove a quadratic bound for general graphs; see Figure 6 for the construction.

► **Theorem 15** (♣). *Any streaming algorithm for DIAMETER on general (dense) graphs in the AL model using p passes over the stream requires $\Omega(n^2/p)$ bits of memory.*

Splitting up u_A and u_B into two vertices each, and making the tails from t'_i to t_i at least three edges longer for each i makes the lower bound work for bipartite graphs (♣).



■ **Figure 1** VA lower bound for diameter with vertex cover size 3, called “Simple VA”. The vertices in the middle are indexed $1, \dots, n$. An edge incident to a (b) is present when the entry of Alice (Bob) at the corresponding index is 1. The vertex c ensures the graph is connected.

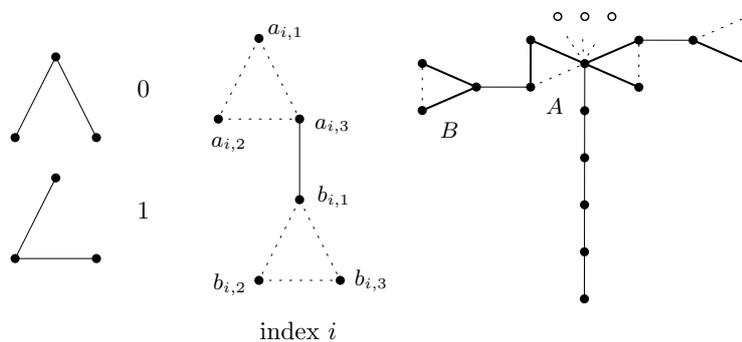


■ **Figure 2** VA lower bound for diameter with distance 2 to 1 clique, called “Clique VA”. A dashed edge is present when the entry at the corresponding index is 1. The vertices inside the grey area form a clique. Hence, deletion distance to a clique is 2 (remove a and b).

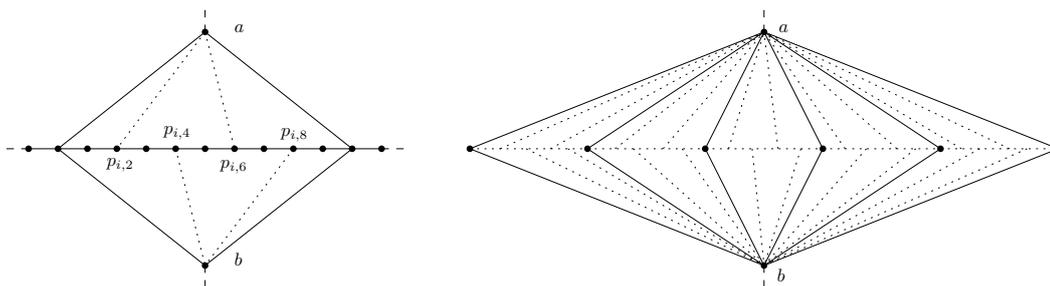
5 Connectivity

In this section, we show results for CONNECTIVITY. CONNECTIVITY is an easier problem than DIAMETER, that is, solving DIAMETER solves CONNECTIVITY as well, but not the other way around. Hence, lower bounds in this section also imply lower bounds for DIAMETER (in non-connected graphs). In general graphs, a single pass, $\mathcal{O}(n \log n)$ bits of memory algorithm exists by maintaining connected components in a Disjoint Set data structure [43], which is optimal in general graphs [50]. The interesting part about CONNECTIVITY is that some graph classes admit fairly trivial algorithms by a counting argument. For example, if the input is a forest, we can decide on CONNECTIVITY by counting the number of edges, which is a 1-pass, $\mathcal{O}(\log n)$ bits of memory, algorithm. An overview of the results in this section is given in Table 3. The following upper bounds follow from applications of the Disjoint Set data structure.

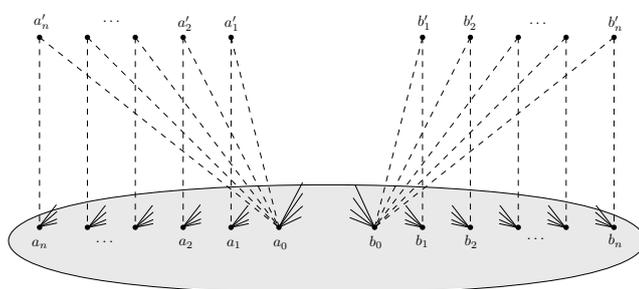
► **Observation 16** (♣). *Given a graph G as an AL stream with vertex cover number k , we can solve CONNECTIVITY $[k]$ in 1 pass and $\mathcal{O}(k \log n)$ bits of memory.*



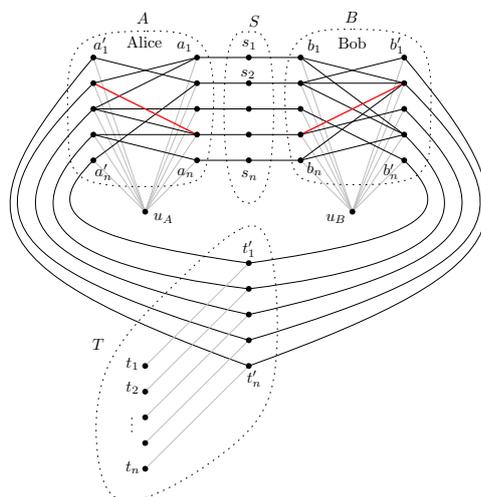
■ **Figure 3** AL lower bound for diameter consisting of a tree, called “Windmill”. The difference in an entry 1 or 0 is shown on the left. The gadget for index i combines a 0/1-gadget for Alice and a 0/1-gadget for Bob. It makes two 1 entries at this index a path of length 5, and a tree structure of depth at most 4 otherwise. These n gadgets are then identified at $a_{i,1}$ and a tail is added.



■ **Figure 4** AL lower bound for diameter consisting of a path and 2 vertices, called “Diamond”. Note that a is connected to b with an edge (indicated with a dashed line here). On the left the gadget for a single index i is shown, where the dotted edges are present when the entry at index i is 0 (for Alice incident on a , for Bob incident on b). On the right, the construction is sketched in full.



■ **Figure 5** AL lower bound for diameter on split graphs, called “Split”. Depending on the input, some a'_i either has an edge to a_0 or a_i when $x_i = 0$ or 1. The same holds for b'_i with y_i . The grey area forms a clique, and each a_i is connected to all b'_j where $i \neq j$, and the same holds for b_i and a'_j .



■ **Figure 6** AL lower bound for diameter where Alice and Bob have n^2 bits but the graph has $\mathcal{O}(n)$ vertices. The bits are seen as an adjacency matrix in the bipartite graphs A and B , identically: the red edge a'_i to a_j in A is the same index as the red edge b_j to b'_i in B . Edges are present when the entry is a 0. Then, each s_i, t_j pair can discern whether or not at least one of the edges a_i to a'_j or b_i to b'_j is present, hence deciding whether or not both Alice and Bob have a 1 at that entry.

■ **Table 3** Overview of the results for CONNECTIVITY. All hardness results listed here are through reductions from DISJOINTNESS. (\mathcal{M}, m, p) -hard means that any algorithm using p passes in model \mathcal{M} (or weaker) requires $\Omega(m)$ bits of memory. (\mathcal{M}, m, p) -str. means that there is an algorithm that uses p passes in model \mathcal{M} (or stronger) using $\mathcal{O}(m)$ bits of memory. FVS stand for Feedback Vertex Set number, FEN for Feedback Edge Set number. We state most upper bounds only as observations, and most proofs of the results in this table are deferred to the full paper (♣).

Parameter (k) / Graph class	Size	Bound
General Graphs		(EA, $n \log n, 1$)-str. via Disjoint Set [43] (EA, $n \log n, 1$)-hard by Sun and Woodruff [50]
VERTEX COVER NUMBER	≥ 0 ≥ 2	(AL, $k \log n, 1$)-str. Disjoint Set on Vertex Cover (VA, $n/p, p$)-hard by Henzinger et al. [38]
DISTANCE TO ℓ CLIQUES	≥ 0	(AL, $(k + \ell) \log n, 1$)-str. via Disjoint Set
FVS	$= 0$ ≥ 1	(EA, $\log n, 1$)-str. by counting. (AL, $n/p, p$)-hard
FES	≥ 0	(EA, $\log n, 1$)-str. by counting.
DISTANCE TO MATCHING	≥ 2	(AL, $n/p, p$)-hard
DISTANCE TO PATH	≥ 0	(EA, $k \log n, 1$)-str. by checking connection to path
DISTANCE TO DEPTH ℓ TREE	≥ 0	(EA, $k \log n, 1$)-str. by checking connection to tree
DOMINATION NUMBER	≥ 2	(AL, $n/p, p$)-hard
DISTANCE TO CHORDAL	≥ 1	(AL, $n/p, p$)-hard
MAXIMUM DEGREE	≥ 2	(AL, $n/p, p$)-hard, (AL, $n \log n, 1$)-hard
Bipartite Graphs		(AL, $n/p, p$)-hard, (AL, $n \log n, 1$)-hard
Interval Graphs		(VA, $n/p, p$)-hard
Split graphs		(EA, $n/p, p$)-str. by finding degree 0 vertex (VA, $n/p, p$)-hard

► **Observation 17 (♣).** *Given a graph G as an AL stream with a deletion set X of size k to ℓ cliques, we can solve CONNECTIVITY $[k, \ell]$ in 1 pass and $\mathcal{O}((k + \ell) \log n)$ bits of memory.*

We fully defer a simple lower bound construction for the AL model to the full version (♣).

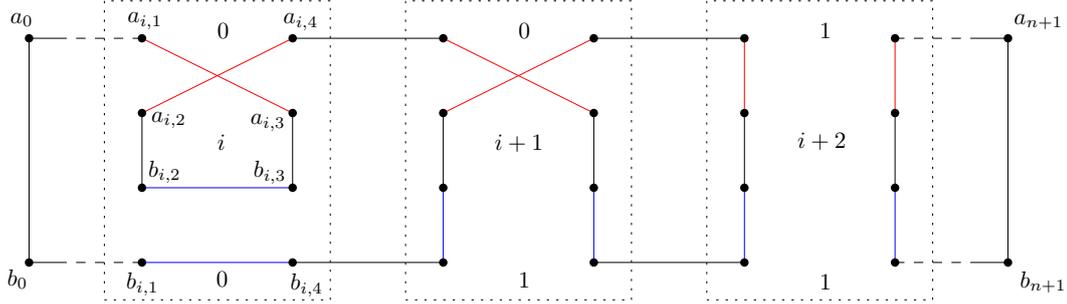
An interesting lower bound is for a unique case: graphs of maximum degree 2. We mentioned that for a forest we have a simple counting algorithm for CONNECTIVITY, so the hardness must be for some graph which consists of one or more cycles. Although we show (♣) that CONNECTIVITY is hard for graphs with a Feedback Vertex Set of constant size, we now show that in the specific case of maximum degree 2-graphs, the problem is still hard, see Figure 7 for an illustration of the construction. We note that this reduction is similar to the problem tackled by Verbin and Yu [51] and Assadi et al. [4], but our result is slightly stronger in this setting, as it concerns a distinction between 1 or 2 disjoint cycles.

► **Theorem 18 (♣).** *Any streaming algorithm for CONNECTIVITY that works on a family of graphs that includes graphs of maximum degree 2 in the AL model using p passes over the stream requires $\Omega(n/p)$ bits of memory.*

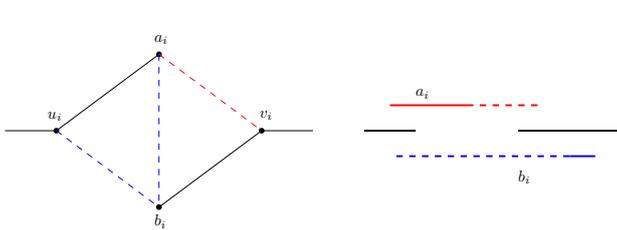
We note that we can make the result of Theorem 18 hold for bipartite graphs of maximum degree 2 by subdividing every edge, making the graph odd cycle-free, and thus bipartite. The proofs of Theorems 5 and 6 follow, and are deferred to the full version (♣).

Interval and split graphs are hard in the VA model, see Figures 8 and 9.

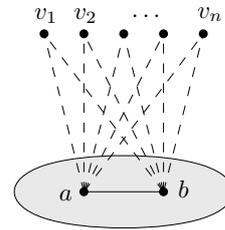
► **Theorem 19 (♣).** *Any streaming algorithm for CONNECTIVITY that works on a family of graphs that includes interval graphs or split graphs in the VA model using p passes over the stream requires $\Omega(n/p)$ bits of memory.*



■ **Figure 7** AL lower bound for connectivity, called “Cycles”. The graph consists of one or multiple cycles depending on the output of DISJ_n . The black edges are always present. The red (blue) edges are controlled by Alice (Bob) and are in a crossing (horizontal) or vertical configuration depending on whether the i -th entry of Alice (Bob) is 0 or 1.



■ **Figure 8** VA lower bound for connectivity on interval graphs, called “Interval”. We see the gadget for index i , where the dotted lines are present when the corresponding value is 0. The black edges are always present, and the red (blue) edges correspond to the input of Alice (Bob). The n gadgets are placed consecutively.



■ **Figure 9** VA lower bound for connectivity on split graphs, called “Split-Conn”. The dashed edges towards v_i are present when there is a 0 at index i .

For split graphs, in any model, CONNECTIVITY admits a one-pass, $\mathcal{O}(n)$ bits of memory algorithm by counting if there is a vertex of degree 0 (and so also for any p p -pass algorithm using $\mathcal{O}(n/p)$ bits by splitting up the work in p parts)². If there can be no isolated vertices, then a split graph is always connected.

6 Vertex Cover kernelization

We now show how our insights into parameterized, streaming graph exploration can aid in producing a new kernelization algorithm for $\text{VERTEX COVER } [k]$ ³. The basis for our result is a well-known kernel for the $\text{VERTEX COVER } [k]$ problem of Buss and Goldsmith [14], consisting of $\mathcal{O}(k^2)$ edges. Constructing this kernel is simple: find all vertices with degree bigger than k , and remove them from the graph, and decrease the parameter with the number of vertices removed, say to k' . Then, there is no solution if there are more than $k \cdot k'$ edges. Therefore, we have a kernel consisting of $\mathcal{O}(k^2)$ edges. We are able to achieve this same kernel in the AL model, as counting the degree of a vertex is possible in this model. Interestingly, we do not require $\tilde{\mathcal{O}}(k^2)$ bits of memory to produce a stream corresponding to the kernel of $\mathcal{O}(k^2)$ edges. This result is also possible in the EA model, by allowing vertices up to degree $2k$.

² This assumes the vertices are labelled $1 \dots n$ and do not have arbitrary labels.

³ This section is based on the master thesis “Parameterized Algorithms in a Streaming Setting” by the first author.

► **Theorem 20** (♣). *Given a graph G as an AL stream, we can make an AL stream corresponding to an $\mathcal{O}(k^2)$ -edge kernel for the VERTEX COVER $[k]$ problem using two passes and $\tilde{\mathcal{O}}(k)$ bits of memory. When we work with an EA stream, we can make an EA stream corresponding to an $\mathcal{O}(k^2)$ -edge kernel using four passes and $\tilde{\mathcal{O}}(k)$ bits of memory.*

Next, we show how to use Theorem 20 to produce a kernel of even smaller size, using only $\tilde{\mathcal{O}}(k)$ bits of memory. This requires Theorem 20 to convert the original graph stream into the kernel input for the next theorem, which only increases the number of passes by a factor 2 or 4 (we have to apply Theorem 20 every time the other procedure uses a pass).

Interestingly, Chen et al. [16] show a way to convert the kernel of Buss and Goldsmith into a $2k$ -vertex kernel for VERTEX COVER $[k]$, using a theorem by Nemhauser and Trotter [45]. We will adapt this method in the streaming setting. The kernel conversion is done by converting the $\mathcal{O}(k^2)$ edges kernel into a bipartite graph (two copies of all vertices V, V' , and an edge (x, y) translates to the edges $(x, y'), (x', y)$), in which we find a minimum vertex cover using a maximum matching (see for example [12, Page 74, Theorem 5.3]). The minimum vertex cover we find gives us the sets stated in the theorem by Nemhauser and Trotter [45], as indicated by the constructive proof of the same theorem by Bar-Yehuda and Even [7]. Lastly, we use these sets to give the $2k$ kernel in the streaming setting as indicated by Chen et al. [16]. This also works for the EA model, because we only require the input kernel to consist of $\mathcal{O}(k^2)$ edges, not that it specifically is the kernel by Buss and Goldsmith.

► **Lemma 21** (♣). *Given a graph G as a stream in model AL or EA, we can produce a stream in the same model corresponding to the Phase 1 bipartite graph of [7, Algorithm NT] using two passes and $\tilde{\mathcal{O}}(1)$ bits of memory.*

By making some observations on the conversions by Chen et al. [16], we can conclude that the maximum matching we need to find in the bipartite graph consists of at most $4k = \mathcal{O}(k)$ edges, and otherwise we can return NO. For more details, see ♣. To find the maximum matching we execute a DFS procedure, which can be done with surprising efficiency in this restricted bipartite setting.

► **Theorem 22** (♣). *Given a bipartite graph B as an AL stream with $\mathcal{O}(k)$ vertices, we can find a maximum matching of size at most $\mathcal{O}(k)$ using $\mathcal{O}(k^2)$ passes and $\tilde{\mathcal{O}}(k)$ bits of memory. For the EA model this can be done in $\mathcal{O}(k^3)$ passes.*

Using this maximum matching, we can find a vertex cover kernel of size $2k$. The final result is as follows, which consists of putting the original stream through each step for every time we require a pass, i.e. the number of passes of each of the parts of this theorem combine in a multiplicative fashion.

► **Theorem 23** (♣). *Given a graph G as an AL stream, we can produce a kernel of size $2k$ for the VERTEX COVER $[k]$ problem using $\mathcal{O}(k^2)$ passes and $\tilde{\mathcal{O}}(k)$ bits of memory. In the EA model, this procedure takes $\mathcal{O}(k^3)$ passes.*

7 Conclusion

We studied the complexity of DIAMETER and CONNECTIVITY in the streaming model, from a parameterized point of view. In particular, we considered the viewpoint of an H -free modulator, showing that a vertex cover or a modulator to the disjoint union of ℓ cliques effectively forms the frontier of memory- and pass-efficient streaming algorithms. Both problems remain hard for almost all other H -free modulators of constant size (often even of

size 0). We believe that this forms an interesting starting point for further investigations into which other graph classes or parameters might be useful when computing DIAMETER and CONNECTIVITY in the streaming model.

On the basis of our work, we propose four concrete open questions:

- What is the streaming complexity of computing DISTANCE TO ℓ CLIQUES? On the converse of VERTEX COVER [k], we are not aware of any algorithms to compute this parameter, even though it is helpful in computing DIAMETER and CONNECTIVITY.
- Are there algorithms or lower bounds for DIAMETER or CONNECTIVITY in the AL model for interval graphs?
- Assuming isolated vertices are allowed in the graph, can we solve CONNECTIVITY in the AL model on split graphs using $O(\log n)$ bits of memory?
- Is there a streaming algorithm for VERTEX COVER [k] using $\mathcal{O}(\text{poly}(k))$ passes and $\mathcal{O}(\text{poly}(k, \log n))$ bits of memory, or can it be shown that one cannot exist? This result would be relevant in combination with our kernel.

References

- 1 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proc. SODA 2016*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.
- 2 Deepak Agarwal, Andrew McGregor, Jeff M. Phillips, Suresh Venkatasubramanian, and Zhengyuan Zhu. Spatial scan statistics: approximations and performance study. In *Proc. SIGKDD 2006*, pages 24–33. ACM, 2006. doi:10.1145/1150402.1150410.
- 3 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Polynomial pass lower bounds for graph streaming algorithms. *CoRR*, abs/1904.04720, 2019. arXiv:1904.04720.
- 4 Sepehr Assadi, Gillat Kol, Raghuvansh R. Saxena, and Huacheng Yu. Multi-pass graph streaming lower bounds for cycle counting, max-cut, matching size, and other problems. In *Proc. FOCS 2020*, pages 354–364. IEEE, 2020. doi:10.1109/FOCS46700.2020.00041.
- 5 Sepehr Assadi and Vishvajeet N. Graph streaming lower bounds for parameter estimation and property testing via a streaming XOR lemma. In *Proc. STOC 2021*, pages 612–625. ACM, 2021. doi:10.1145/3406325.3451110.
- 6 Sepehr Assadi and Ran Raz. Near-quadratic lower bounds for two-pass graph streaming algorithms. In *Proc. FOCS 2020*, pages 342–353. IEEE, 2020. doi:10.1109/FOCS46700.2020.00040.
- 7 Reuven Bar-Yehuda and Shimon Even. A local-ratio theorem for approximating the weighted vertex cover problem. In *Proc. WG 1983*, pages 17–28. Universitätsverlag Rudolf Trauner, Linz, 1983. URL: <http://www.gbv.de/dms/tib-ub-hannover/022054669.pdf>.
- 8 Matthias Bentert and André Nichterlein. Parameterized complexity of diameter. In *Proc. CIAC 2019*, volume 11485 of *LNCS*, pages 50–61. Springer, 2019. doi:10.1007/978-3-030-17402-6_5.
- 9 Arijit Bishnu, Arijit Ghosh, Sudeshna Kolay, Gopinath Mishra, and Saket Saurabh. Fixed-parameter tractability of graph deletion problems over data streams. *CoRR*, abs/1906.05458, 2019. arXiv:1906.05458.
- 10 Arijit Bishnu, Arijit Ghosh, Sudeshna Kolay, Gopinath Mishra, and Saket Saurabh. Fixed parameter tractability of graph deletion problems over data streams. In *Proc. COCOON 2020*, volume 12273 of *LNCS*, pages 652–663. Springer, 2020. doi:10.1007/978-3-030-58150-3_53.
- 11 Arijit Bishnu, Arijit Ghosh, Gopinath Mishra, and Sandeep Sen. On the streaming complexity of fundamental geometric problems. *CoRR*, abs/1803.06875, 2018. arXiv:1803.06875.
- 12 J. Adrian Bondy and Uppaluri S. R. Murty. *Graph Theory with Applications*. Macmillan Education UK, 1976. doi:10.1007/978-1-349-03521-2.

- 13 Karl Bringmann, Thore Husfeldt, and Måns Magnusson. Multivariate analysis of orthogonal range searching and graph distances. *Algorithmica*, 82(8):2292–2315, 2020. doi:10.1007/s00453-020-00680-z.
- 14 Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. *SIAM J. Comput.*, 22(3):560–572, 1993. doi:10.1137/0222038.
- 15 Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. *ACM Trans. Algorithms*, 15(2):21:1–21:38, 2019. doi:10.1145/3218821.
- 16 Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *Journal of Algorithms*, 41(2):280–301, 2001. doi:10.1006/jagm.2001.1186.
- 17 Lijie Chen, Gillat Kol, Dmitry Paramonov, Raghuvansh R. Saxena, Zhao Song, and Huacheng Yu. Almost optimal super-constant-pass streaming lower bounds for reachability. In *Proc. STOC 2021*, pages 570–583. ACM, 2021. doi:10.1145/3406325.3451038.
- 18 Rajesh Chitnis and Graham Cormode. Towards a theory of parameterized streaming algorithms. In *Proc. IPEC 2019*, volume 148 of *LIPICs*, pages 7:1–7:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.IPEC.2019.7.
- 19 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, Mohammad Taghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proc. SODA 2016*, pages 1326–1344. SIAM, 2016. doi:10.1137/1.9781611974331.ch92.
- 20 Rajesh Hemant Chitnis, Graham Cormode, Hossein Esfandiari, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Brief announcement: New streaming algorithms for parameterized maximal matching & beyond. In *Proc. SPAA 2015*, pages 56–58. ACM, 2015. doi:10.1145/2755573.2755618.
- 21 Rajesh Hemant Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *Proc. SODA 2015*, pages 1234–1251. SIAM, 2015. doi:10.1137/1.9781611973730.82.
- 22 Derek G. Corneil, Feodor F. Dragan, Michel Habib, and Christophe Paul. Diameter determination on restricted graph families. *Discret. Appl. Math.*, 113(2-3):143–166, 2001. doi:10.1016/S0166-218X(00)00281-X.
- 23 David Coudert, Guillaume Ducoffe, and Alexandru Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. *ACM Trans. Algorithms*, 15(3):33:1–33:57, 2019. doi:10.1145/3310228.
- 24 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.
- 25 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.
- 26 Guillaume Ducoffe. Beyond helly graphs: The diameter problem on absolute retracts. In *Proc. WG 2021*, volume 12911 of *LNCS*, pages 321–335. Springer, 2021. doi:10.1007/978-3-030-86838-3_25.
- 27 Guillaume Ducoffe and Feodor F. Dragan. A story of diameter, radius, and (almost) helly property. *Networks*, 77(3):435–453, 2021. doi:10.1002/net.21998.
- 28 Guillaume Ducoffe, Michel Habib, and Laurent Viennot. Fast diameter computation within split graphs. In *Proc. COCOA 2019*, volume 11949 of *LNCS*, pages 155–167. Springer, 2019. doi:10.1007/978-3-030-36412-0_13.
- 29 Guillaume Ducoffe, Michel Habib, and Laurent Viennot. Diameter computation on H -minor free graphs and graphs of bounded (distance) vc-dimension. In *Proc. SODA 2020*, pages 1905–1922. SIAM, 2020. doi:10.1137/1.9781611975994.117.
- 30 Michael Elkin. Distributed exact shortest paths in sublinear time. *J. ACM*, 67(3):15:1–15:36, 2020. doi:10.1145/3387161.
- 31 Michael Elkin and Chhaya Trehan. $(1 + \epsilon)$ -approximate shortest paths in dynamic streams. *CoRR*, abs/2107.13309, 2021. arXiv:2107.13309.
- 32 Stefan Fafianie and Stefan Kratsch. Streaming kernelization. In *Proc. MFCS 2014*, volume 8635 of *LNCS*, pages 275–286. Springer, 2014.

- 33 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005. doi:10.1016/j.tcs.2005.09.013.
- 34 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. *SIAM J. Comput.*, 38(5):1709–1727, 2008. doi:10.1137/070683155.
- 35 Pawel Gawrychowski, Haim Kaplan, Shay Mozes, Micha Sharir, and Oren Weimann. Voronoi diagrams on planar graphs, and computing the diameter in deterministic $\tilde{O}(n^{5/3})$ time. *SIAM J. Comput.*, 50(2):509–554, 2021. doi:10.1137/18M1193402.
- 36 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In Yuval Rabani, editor, *Proc. SODA 2012*, pages 468–485. SIAM, 2012. doi:10.1137/1.9781611973099.41.
- 37 Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. *Algorithmica*, 76(3):654–683, 2016. doi:10.1007/s00453-016-0138-7.
- 38 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. In *Proc. DIMACS 1998*, volume 50 of *DIMACS*, pages 107–118. DIMACS/AMS, 1998. doi:10.1090/dimacs/050/05.
- 39 Zengfeng Huang and Pan Peng. Dynamic graph stream algorithms in $o(n)$ space. *Algorithmica*, 81(5):1965–1987, 2019. doi:10.1007/s00453-018-0520-8.
- 40 Thore Husfeldt. Computing graph distances parameterized by treewidth and diameter. In *Proc. IPEC 2016*, volume 63 of *LIPIcs*, pages 16:1–16:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.16.
- 41 Michael Kapralov. Better bounds for matchings in the streaming model. In Sanjeev Khanna, editor, *Proc. SODA 2013*, pages 1679–1697. SIAM, 2013. doi:10.1137/1.9781611973105.121.
- 42 Shahbaz Khan and Shashank K. Mehta. Depth first search in the semi-streaming model. In Rolf Niedermeier and Christophe Paul, editors, *Proc. STACS 2019*, volume 126 of *LIPIcs*, pages 42:1–42:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.42.
- 43 Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Rec.*, 43(1):9–20, 2014. doi:10.1145/2627692.2627694.
- 44 Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better algorithms for counting triangles in data streams. In Tova Milo and Wang-Chiew Tan, editors, *Proc. PODS 2016*, pages 401–411. ACM, 2016. doi:10.1145/2902251.2902283.
- 45 George L. Nemhauser and Leslie E. Trotter Jr. Vertex packings: Structural properties and algorithms. *Math. Program.*, 8(1):232–248, 1975. doi:10.1007/BF01580444.
- 46 Jelle J. Oostveen and Erik Jan van Leeuwen. Streaming deletion problems parameterized by vertex cover. In *Proc. FCT 2021*, volume 12867 of *LNCS*, pages 413–426. Springer, 2021. doi:10.1007/978-3-030-86593-1_29.
- 47 Jelle J. Oostveen and Erik Jan van Leeuwen. Parameterized complexity of streaming diameter and connectivity problems. *CoRR*, abs/2207.04872, 2022. doi:10.48550/arXiv.2207.04872.
- 48 John H. Reif. Depth-first search is inherently sequential. *Inf. Process. Lett.*, 20(5):229–234, 1985. doi:10.1016/0020-0190(85)90024-9.
- 49 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proc. STOC 2013*, pages 515–524. ACM, 2013. doi:10.1145/2488608.2488673.
- 50 Xiaoming Sun and David P. Woodruff. Tight bounds for graph problems in insertion streams. In *Proc. APPROX/RANDOM 2015*, volume 40 of *LIPIcs*, pages 435–448. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.435.
- 51 Elad Verbin and Wei Yu. The streaming complexity of cycle counting, sorting by reversals, and other problems. In Dana Randall, editor, *Proc. SODA 2011*, pages 11–25. SIAM, 2011. doi:10.1137/1.9781611973082.2.

Applying a Cut-Based Data Reduction Rule for Weighted Cluster Editing in Polynomial Time

Hjalmar Schulz ✉

Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

André Nichterlein ✉ 

Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Rolf Niedermeier ✉ 

Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Christopher Weyand ✉

Karlsruhe Institute of Technology, Germany

Abstract

Given an undirected graph, the task in CLUSTER EDITING is to insert and delete a minimum number of edges to obtain a cluster graph, that is, a disjoint union of cliques. In the weighted variant each vertex pair comes with a weight and the edge modifications have to be of minimum overall weight. In this work, we provide the first polynomial-time algorithm to apply the following data reduction rule of Böcker et al. [Algorithmica, 2011] for WEIGHTED CLUSTER EDITING: For a graph $G = (V, E)$, merge a vertex set $S \subseteq V$ into a single vertex if the minimum cut of $G[S]$ is at least the combined cost of inserting all missing edges within $G[S]$ plus the cost of cutting all edges from S to the rest of the graph. Complementing our theoretical findings, we experimentally demonstrate the effectiveness of the data reduction rule, shrinking real-world test instances from the PACE Challenge 2021 by around 24% while previous heuristic implementations of the data reduction rule only achieve 8%.

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

Keywords and phrases Correlation Clustering, Minimum Cut, Maximum s - t -Flow

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.25

Supplementary Material *Software (Source Code)*: <https://github.com/venondev/AlmostClique> Poly, archived at `swh:1:dir:8aca200ba4c16f1b357d20904271c630e5c4fa5a`

Acknowledgements In memory of Rolf Niedermeier, our colleague, friend, and mentor, who sadly passed away before this paper was published.

1 Introduction

The NP-hard CLUSTER EDITING problem [4, 26], also known as CORRELATION CLUSTERING [3], is one of the most popular graphs clustering approaches in algorithmics. Given an undirected graph, the task is to transform it into a disjoint union of cliques (also known as a cluster graph) by applying a minimum number of edge modifications (deletions or insertions). In the weighted variant WEIGHTED CLUSTER EDITING each pair of vertices comes with a weight and to goal is a find edge modifications of minimum summed weight to create a cluster graph. (WEIGHTED) CLUSTER EDITING has applications in fields such as bioinformatics [4], data mining [3], and psychology [27]. It gained high popularity in studies concerning parameterized algorithmics [1, 2, 9, 11, 14, 15, 22, 5, 7, 6] and algorithm engineering [15, 8, 17, 5]. The unweighted CLUSTER EDITING was the problem selected for the PACE implementation challenge 2021 [21]. An important aspect of solving (WEIGHTED) CLUSTER EDITING, both in theory and practice, is kernelization. From a theoretical side,



© Hjalmar Schulz, André Nichterlein, Rolf Niedermeier, and Christopher Weyand; licensed under Creative Commons License CC-BY 4.0

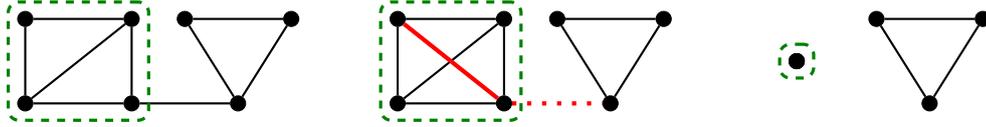
17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 25; pp. 25:1–25:14

Leibniz International Proceedings in Informatics



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



■ **Figure 1** *Left:* Input graph with two “obvious” clusters, one being highlighted in green. *Middle:* Optimal CLUSTER EDITING-solution for the input graph: the thick red edge in the green cluster is inserted and the red dotted edge between the clusters is deleted. *Right:* The graph obtained by merging the vertices in the green vertex set in the input (weights are not shown for visibility). If all weights are 1 in the input graph, then the vertex subset highlighted in green satisfies the condition of the data reduction rule: the minimum cut has weight 2 (cutting two edges), which is at least the cost of 2 to make the green vertex set an isolated clique (see middle for the two modified edges).

both the weighted and unweighted version of CLUSTER EDITING admit polynomial-size problem kernels. Studies in this direction were initialized by Gramm et al. [15] for CLUSTER EDITING, who provided a kernel with $\mathcal{O}(k^2)$ vertices and sparked follow up work [13, 16]. The smallest known kernels have $2k$ vertices [9, 10]. The $2k$ -vertex kernel of Cao and Chen [9] also holds for WEIGHTED CLUSTER EDITING.

As to the practical side, state-of-the-art solvers for CLUSTER EDITING and WEIGHTED CLUSTER EDITING rely on polynomial-time computable data reduction rules and the preprocessing routines are heavily optimized over time [8, 17, 5]. In fact, the winning solver of the PACE challenge 2021 solves about half of the instances by data reduction alone [5]. We contribute to this line of work by providing a divide & conquer-based, polynomial-time algorithm to apply a data reduction rule by Böcker et al. [8, Rule 4] for WEIGHTED CLUSTER EDITING. This data reduction rule works intuitively as follows: Let $S \subseteq V$ be a vertex subset. If the cost of splitting S into at least two parts is at least as high as the cost of cutting S from the rest of the graph and making S a clique, then merge all vertices in S ; see Figure 1 for an illustration and Section 2 for the exact formulation of the data reduction rule. Given a vertex subset S , it is easy to check in polynomial time whether the data reduction rule is applicable on S (call such vertex subsets *applicable*). However, there was no efficient algorithm to find applicable vertex subsets; thus only heuristics were applied [8]. We provide experiments demonstrating that these heuristics miss many applicable vertex subsets in real world data sets: The heuristics merge on average only 8.1% of the vertices in the input. However, the exhaustive application of the data reduction rule with our polynomial-time algorithm reveals that on average 24.2% of the vertices in the input could be merged.

Our polynomial-time algorithm runs in $\mathcal{O}(n \cdot (T_{\text{mincut}}(n, m) + T_{s-t\text{-maxflow}}(n, m) + n^2)) \subseteq \mathcal{O}(nm^{1+o(1)} + n^3)$ time, where $T_{\text{mincut}}(n, m)$ and $T_{s-t\text{-maxflow}}(n, m)$ denote the time to compute in an edge-weighted graph with n vertices and m edges a minimum cut and a maximum s - t -flow, respectively.

2 Preliminaries

We set $\mathbb{N} := \{0, 1, 2, \dots\}$ and set $\binom{S}{2}$ to be the set of all two-element subsets of a set S . Let Δ denote the symmetric difference. All graphs considered in this work are simple and undirected. Moreover, we assume that the input graph is always connected as connected components can be solved independently. A graph is a *cluster graph* if each connected component is a clique. For a weighted graph $G = (V, E, \omega)$, the weight function $\omega: \binom{V}{2} \rightarrow \mathbb{Z}$ implicitly defines the edges $E := \{uv \mid \omega(uv) > 0\}$. That is, a positive value of the weight function indicates an edge and a negative value (or zero) a non-edge. The cost of modifying

an edge uv is then the absolute value $|\omega(uv)|$ of its weight. For a vertex set $S \subseteq V$, we denote with $G[S]$ the graph induced by S . The decision variant of WEIGHTED CLUSTER EDITING is defined as follows:

WEIGHTED CLUSTER EDITING

Input: An undirected edge-weighted graph $G = (V, E, \omega)$ and $k \in \mathbb{N}$.

Question: Is there a set $P \subseteq \binom{V}{2}$ with $\sum_{uv \in P} |\omega(uv)| \leq k$, such that $G' = (V, E \Delta P)$ is a cluster graph?

Cuts and the Picard-Queyranne DAG. Let $G = (V, E, \omega)$ be a weighted graph. Let $U, W \subseteq V$ with $U \cap W = \emptyset$. We denote with $E(U, W)$ the edges between U and W ; with $V(U, W)$ the vertices incident to $E(U, W)$; and with $\text{cost}(U, W)$ the summed cost of removing the edges $E(U, W)$. A cut c_V of G is a partitioning of the vertex set V into two non-empty partitions $U \subseteq V$ and $V \setminus U$, each being called a *side* of the cut. The minimum cut (*mincut*) of G is the cut of G with minimum cost; we denote its cost with $\text{mincut}(G) := \min_{U \subseteq V} \{\text{cost}(U, V \setminus U)\}$. Note that, the cuts consider just the edges of the graph and ignore the non-edges.

Let $s, t \in V$. All minimum s - t -cuts can be represented by a structure called a Picard-Queyranne DAG (PQ-DAG for short) [25]. It is constructed by considering the reachability relation in the residual network of any maximum s - t -flow and contracting strongly connected components. *Contracting* two vertices u and v means to consider them as one, new vertex x with neighborhood $N(u) \cup N(v)$. More precisely, for $w \in N(u) \cup N(v) \setminus \{u, v\}$, the weight of the edge to x becomes $\omega(xw) = \max(0, \omega(uw)) + \max(0, \omega(vw))$. Therefore, each node of the PQ-DAG represents a set of vertices of the original graph. For better distinction we say the PQ-DAG has *nodes* which represent subsets of *vertices* of the input graph. If the graph is undirected, then the DAG has only one sink (the strongly connected component containing s) and only one source (the strongly connected component containing t). A *closure* of a DAG is a set of nodes without outgoing arcs. Each closure of the PQ-DAG represents a minimum s - t -cut [25], that is, one side of a minimum s - t -cut. Thus, any postfix and any prefix in any topological ordering of the PQ-DAG represents a minimum s - t -cut. Moreover, for any minimum s - t -cut, there exists some topological ordering of the PQ-DAG with a prefix representing this cut. In this case, we say that the ordering *respects* the cut.

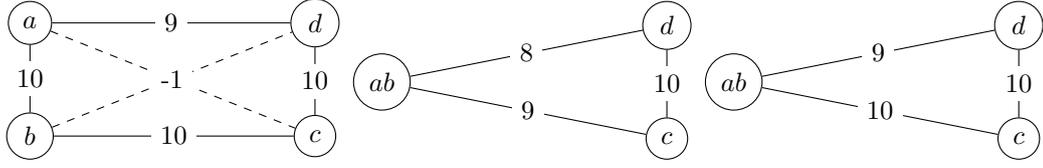
Merging Vertices & Data Reduction Rule. We will contract vertices when working with cuts. For WEIGHTED CLUSTER EDITING we need to *merge* vertices (these two notions coincide if each non-edge has weight 0). Merging two vertices u and v means to consider them as one, new vertex x . For each $w \in V \setminus \{u, v\}$, the weight of the (non-)edge to x becomes $\omega(xw) = \omega(uw) + \omega(vw)$. If w was adjacent to exactly one of u and v , then the solution size k is reduced by $\min(|\omega(uw)|, |\omega(vw)|)$. Intuitively, if a vertex w is (non-)adjacent to both u and v , then w is also (non-)adjacent to the new vertex. However, if w is adjacent to exactly one of u and v , then the new vertex is adjacent to w iff, of the pairs uw and vw , the pair representing an edge has higher weight than the one representing a non-edge. Merging a and b in the graph on the left side of Figure 2 results in the graph on the right side.

For a vertex set $S \subseteq V$, we call the summed cost of all non-edges in $G[S]$ the *deficiency* of S and define $\text{def}_G(S) := \sum_{u, v \in S} |\min(0, \omega(uv))|$. We can now formally state the condition triggering the data reduction rule.

► **Definition 2.1.** A vertex subset $S \subseteq V$ with $|S| \geq 2$ is applicable if

$$\text{mincut}(G[S]) \geq \text{def}_G(S) + \text{cost}(S, V \setminus S). \quad (1)$$

25:4 Applying a Cut-Based Data Reduction Rule in Polynomial Time



■ **Figure 2** *Left:* A graph where the only applicable vertex set is $S = \{a, b, c, d\}$. *Middle:* Result of merging a and b . *Right:* Result of contracting a and b .

■ Algorithm 1 Applying Reduction Rule 2.2.

Input: A connected weighted graph $G = (V, E, \omega)$ and a vertex subset $A \subseteq V$.

Output: A largest applicable set $S \subseteq A$ if it exists, \emptyset otherwise.

```

1 Function FindMergeSet( $G, A$ )
2   if  $|A| < 2$  then return  $\emptyset$ 
3    $c_A = (A_1, A_2) \leftarrow$  arbitrary mincut in  $G[A]$ 
4    $S \subseteq A \leftarrow$  largest applicable set that is cut by  $c_A$ 
   // here max returns the largest set
5   return  $\max(S, \text{FindMergeSet}(G, A_1), \text{FindMergeSet}(G, A_2))$ 

```

► **Reduction Rule 2.2** (Almost clique; Böcker et al. [8, Rule 4]). Let $S \subseteq V$ be an applicable vertex set. Then merge the vertices within S and reduce the solution size accordingly.

3 A Polynomial-Time Algorithm to Apply Reduction Rule 2.2

In this section, we present a polynomial-time algorithm for applying Reduction Rule 2.2. More precisely, for a graph $G = (V, E, \omega)$ and set $A \subseteq V$, our algorithm finds the largest applicable set $S \subseteq A$ if such a set exists. Our algorithm follows a simple divide & conquer approach (see Algorithm 1) and starts with $A = V$. First, compute a mincut in the graph. Then, finding an applicable vertex set that is within a side of the cut is simply a recursive call. The interesting part is the conquer-step to find the largest applicable vertex set that has vertices on both sides of the cut. Such a set is either A itself or a proper subset of A . To find the latter in polynomial time, we need some structural insights presented in the following lemma. It provides some restrictive conditions on such sets (see left side of Figure 3 for an illustration of the setting).

► **Lemma 3.1.** *Let $G = (V, E, \omega)$ be a graph, $A \subseteq V$, and let $c_A = (A_1, A_2)$ be a mincut of $G[A]$. If there is an applicable set $S \subset A$ that is cut by c_A , that is, $S \cap A_1 \neq \emptyset$ and $S \cap A_2 \neq \emptyset$, then*

(a) $\text{def}_G(S) = 0$ and

(b) $\text{mincut}(G[S]) = \text{mincut}(G[A]) = \text{cost}(S, A \setminus S) = \text{cost}(S, V \setminus S)$.

Proof. By assumption, c_A is a mincut in $G[A]$ that also cuts S , thus $\text{mincut}(G[S]) \leq \text{mincut}(G[A])$. Since S is a proper subset of A we have $\text{mincut}(G[A]) \leq \text{cost}(S, A \setminus S) \leq \text{cost}(S, V \setminus S)$. Since $\text{def}_G(S) \geq 0$ and S is applicable, we conclude that

$$\begin{aligned} \text{mincut}(G[S]) &\leq \text{mincut}(G[A]) \leq \text{cost}(S, A \setminus S) \leq \text{cost}(S, V \setminus S) \\ &\leq \text{cost}(S, V \setminus S) + \text{def}_G(S) \stackrel{(1)}{\leq} \text{mincut}(G[S]). \end{aligned}$$

Hence all inequalities are equalities and $\text{def}_G(S) = 0$. ◀

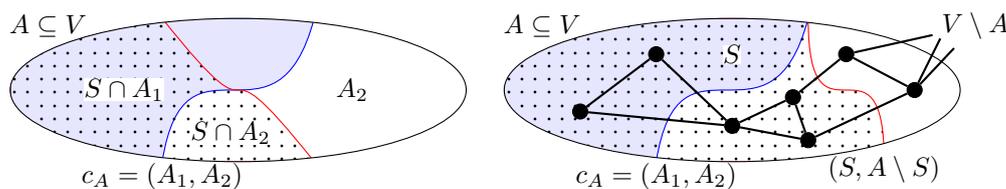


Figure 3 *Left:* Illustration of the setting of Lemmas 3.1 and 3.2 with a mincut (A_1, A_2) for $G[A]$ (black ellipse) and S (dotted area) overlapping with both sides of the cut (one side with blue background). *Right:* An example where a vertex set S (dotted area) with $A_1 \subseteq S$ and $S \cap A_2 \neq \emptyset$ exists. For simplicity, all edges have weight 1 and all non-edges have weight 0. The set A_1 (highlighted by blue background) contains two connected vertices. The mincut of $G[S]$ is 2, the same as both the cost of $c_A = (A_1, A_2)$ and of $(S, V \setminus S) = (S, A \setminus S)$ (two edges cut).

Lemma 3.1 already provides enough information to perform the conquer step in polynomial time: The set S induces another mincut $(S, A \setminus S)$ of $G[A]$. As all mincuts in a graph can be computed in polynomial time [24, 20], we can simply iterate over those and check if another mincut of $G[A]$ has an applicable set S as one side. We can, however, improve this and avoid computing all mincuts. To this end, we use the following observations.

Lemma 3.2. *Let $G = (V, E, \omega)$ be a graph, $A \subseteq V$, and let $c_A = (A_1, A_2)$ be a mincut of $G[A]$. If there is an applicable set $S \subset A$ that is cut by c_A , that is, $S \cap A_1 \neq \emptyset$ and $S \cap A_2 \neq \emptyset$, then*

- (a) $(S, A \setminus S)$ is a mincut of $G[A]$,
- (b) the endpoints of edges crossing c_A are part of S , i. e. $V(A_1, A_2) \subseteq S$,
- (c) S includes either A_1 or A_2 ,
- (d) S is not connected to vertices outside A , i. e. $\text{cost}(S, V \setminus A) = 0$, and
- (e) $\text{def}_G(A_i \cup V(A_1, A_2)) = 0 = \text{cost}(A_i \cup V(A_1, A_2), V \setminus A)$ for $i = 1$ or $i = 2$.

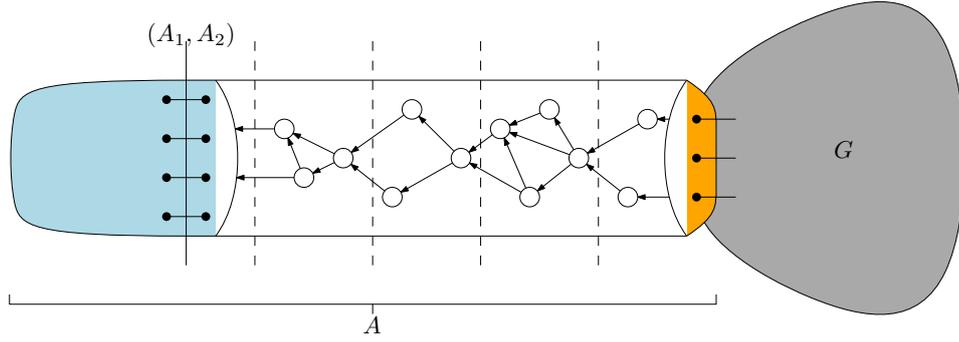
Proof.

- (a) This follows from $\text{mincut}(G[A]) = \text{cost}(S, A \setminus S)$ from Lemma 3.1.
- (b) If a vertex $v \in V(A_1, A_2)$ were not in S , then $c_S = (A_1 \cap S, A_2 \cap S)$ would be cut of $G[S]$ with $E(c_S) \subset E(c_A)$ and $\text{cost}(c_S) < \text{cost}(c_A) = \text{mincut}(G[A])$, contradicting Lemma 3.1.
- (c) If S includes neither A_1 nor A_2 , then we have $\text{cost}(S, A \setminus S) = \text{cost}(S, A_1 \setminus S) + \text{cost}(S, A_2 \setminus S) > \text{cost}(S, A_1 \setminus S) = \text{cost}(S \cup A_2, A \setminus (S \cup A_2))$ contradicting that $(S, A \setminus S)$ is a mincut of $G[A]$.
- (d) This follows from $\text{cost}(S, A \setminus S) = \text{cost}(S, V \setminus S)$.
- (e) This directly follows from (b), (c), (d), and $\text{def}_G(S) = 0$ (by Lemma 3.1). \blacktriangleleft

In practice, Lemma 3.2 (e) almost always rules out the existence of a set S going over the cut (A_1, A_2) . In theory, however, such a set S fulfilling all the restrictions of Lemmas 3.1 and 3.2 can exist, see Figure 3 (right) for an example.

We already know that isolating S is a mincut of $G[A]$. In the following we identify vertices that must be on opposite sides of this cut. This reduces the mincut problem to a more manageable s - t -cut problem. Those s - t -cuts can be represented by a PQ-DAG.

Since we assume the graph to be connected, only one side of c_A can be isolated from $V \setminus A$. Let A_1 be this side, hence $A_1 \subset S$ by Lemma 3.2 (c). By Lemma 3.2 (b, d), the sought-after set S contains also all vertices incident to edges crossing c_A and cannot contain vertices of A that are connected to the rest of the graph. Thus, if we contract all vertices in $A_1 \cup V(A_1, A_2)$ into one source s and all vertices in A with neighbors in $V \setminus A$ into one sink t , then S must be a minimum s - t -cut. See Figure 4 for an overview. Unfortunately, in the case that $V = A$,



■ **Figure 4** Layout of S , A , G and the PQ-DAG. Vertices that must be in S are colored blue. Vertices that must not be in S are colored orange. Those two colored sets are contained in the sink/source component, respectively, of the PQ DAG representing all minimum s - t -cuts between them.

there is no $V \setminus A$ and we need another approach. Recall that $\text{def}_G(S) = 0$. Thus, if both sides of c_A have deficit, there is no applicable S . If both sides have no deficit, then the instance is trivial because the solution that transforms V to a clique has cost zero. Only if one side has deficit and the other has not, there may exist an applicable set. In this case the side without deficit must be included in S and at least one endpoint of an edge with negative weight cannot be in S . Therefore we can solve the $A = V$ case with two computations similar to the $A \subset V$ case by trying both endpoints of an arbitrary deficit edge as the sink.

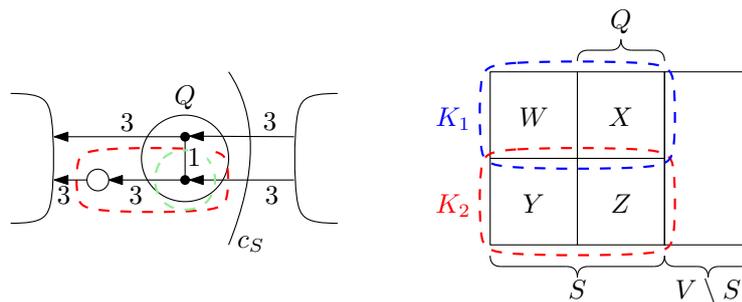
▶ **Lemma 3.3.** *Let $G = (V, E, \omega)$ be a weighted graph, $s, t \in V$, and $c_S = (S, V \setminus S)$ a minimum s - t -cut with $s \in S$, $t \in V \setminus S$ and $\text{mincut}(G[S]) = \text{cost}(c_S)$. Then, there exists a cut node $Q \subseteq S$ of the PQ-DAG that separates $S \setminus Q$ from $V \setminus S$.*

Proof. Let Q_0, Q_1, \dots, Q_p be a reverse topological ordering of the PQ-DAG of the residual graph of G that respects c_S . Let $Q_{\leq i} := \bigcup_{j \leq i} Q_j$ and $Q_{> i} := \bigcup_{j > i} Q_j$. The minimum s - t -cuts respected by this ordering have the form $c_i = (Q_{\leq i}, Q_{> i})$ (by definition they all have the same cost). Thus, $s \in Q_0$, $t \in Q_p$, $c_S = c_\ell$, and $S = Q_{\leq \ell}$, for some $\ell < p$.

$$\begin{aligned}
 \text{cost}(c_S) &= \text{cost}(c_\ell) = \text{cost}(c_{\ell-1}) = \text{cost}(Q_{\leq \ell-1}, Q_{> \ell-1}) \\
 &= \text{cost}(Q_{\leq \ell-1}, Q_\ell) + \text{cost}(Q_{\leq \ell-1}, Q_{> \ell}) \geq \text{cost}(Q_{\leq \ell-1}, Q_\ell) \geq \text{mincut}(G[S]).
 \end{aligned}$$

Since $\text{mincut}(G[S]) = \text{cost}(c_S)$, the inequalities become equalities and $\text{cost}(Q_{\leq \ell-1}, Q_{> \ell}) = 0$. Therefore, Q_ℓ is the desired cut node. ◀

The ordering of cut-nodes in a DAG with one sink and one source is the same in all possible topological orderings. Hence, Lemma 3.3 gives potential candidate sets $S_1 \subseteq \dots \subseteq S_\ell$ with $\ell < n$. For each set we need to verify inequality (1), that is, compute the mincut of $G[S]$ and check that the deficiency of $G[S]$ is zero (recall that $\text{cost}(S, V \setminus S) = \text{mincut}(G[A])$ by construction of the PQ-DAG). The deficiency can be easily verified in $\mathcal{O}(n^2)$ time in total for all candidate sets due to them being subsets of each other. The computation of the mincuts is not as easy. While we are not aware of a way to circumvent ℓ separate mincut computations, we can reduce the size of the involved graphs such that all these graphs have in total size $\mathcal{O}(n + m)$. To this end, we require another lemma with structural insights; see Figure 5 (left) for an illustration of the setting.



■ **Figure 5** *Left:* The figure shows an example where $\text{mincut}(G[S]) < \text{cost}(c_S)$ given the setting of Lemma 3.4. Two mincuts of $G[S]$ are highlighted; one with a side contained in Q (green) and one with no side contained in Q (red). *Right:* Layout of the sets used during the proof of Lemma 3.4.

► **Lemma 3.4.** *Let $G = (V, E, \omega)$ be a weighted graph and $s, t \in V$. Further, let $c_S = (S, V \setminus S)$ be a minimum s - t -cut such that $s \in S$, $t \in V \setminus S$, $\text{mincut}(G[S]) < \text{cost}(c_S) = \text{mincut}(G)$, and there exists a cut node $Q \subseteq S$ of the PQ-DAG that separates $S \setminus Q$ from $V \setminus S$. Then there exists a cut $(Z, S \setminus Z)$ in $G[S]$ with $\text{cost}(Z, S \setminus Z) < \text{mincut}(G)$ and $Z \subseteq Q$.*

Proof. Let $c_K = (K_1, K_2)$ be a mincut of $G[S]$ with cost less than $\text{mincut}(G)$. First, we argue that c_K must split Q . Assume that c_K does not split Q and w.l.o.g. $Q \subseteq K_2$. Then, Q (and therefore also K_2) separates K_1 from $V \setminus S$. But this means that $(K_1, V \setminus K_1)$ is a cut of G with $\text{cost}(K_1, V \setminus K_1) = \text{cost}(K_1, K_2) < \text{mincut}(G)$. Hence, c_K must split Q .

Let $W = K_1 \setminus Q$, $X = K_1 \cap Q$, $Y = K_2 \setminus Q$, and $Z = K_2 \cap Q$. In the following, we will show that the cut of $G[S]$ that isolates Z is not larger than c_K which would prove the claim. The layout of the sets is visualized in Figure 5 (right).

Assume towards a contradiction that $\text{cost}(c_K) < \text{cost}(Z, S \setminus Z)$, i.e., $\text{cost}(Y \cup Z, W \cup X) < \text{cost}(Z, Y \cup W \cup X)$. Splitting the costs into their parts results in

$$\begin{aligned} \text{cost}(Y, W) + \text{cost}(Y, X) + \text{cost}(Z, W) + \text{cost}(Z, X) &= \text{cost}(Y \cup Z, W \cup X) \\ &< \text{cost}(Z, Y \cup W \cup X) = \text{cost}(Z, Y) + \text{cost}(Z, W) + \text{cost}(Z, X) \end{aligned}$$

and hence $\text{cost}(Y, W) + \text{cost}(Y, X) < \text{cost}(Z, Y)$.

With this we will get that the cut in $G[S]$ isolating $Y \cup W$ is strictly more expensive than the cut that isolates just W . In detail,

$$\begin{aligned} \text{cost}(X \cup Z, Y \cup W) &= \text{cost}(X, W) + \text{cost}(Z, W) + \text{cost}(Z, Y) + \text{cost}(X, Y) \\ &> \text{cost}(X, W) + \text{cost}(Z, W) + \text{cost}(Z, Y) \\ &> \text{cost}(X, W) + \text{cost}(Z, W) + \text{cost}(Y, W) + \text{cost}(Y, X) \\ &> \text{cost}(X, W) + \text{cost}(Z, W) + \text{cost}(Y, W) \\ &= \text{cost}(W, X \cup Y \cup Z) \end{aligned}$$

Since Q separates W and Y from $V \setminus S$, the cuts that isolate $Y \cup W$ or just W , respectively, have the same cost in G as they have in $G[S]$. Furthermore, the cut that isolates $Y \cup W$ in $G[S]$ is a mincut of G , because it is the cut in the PQ-DAG just before the cut node Q and thus has a cost of $\text{cost}(c_S) = \text{mincut}(G)$. But this means that isolating W in G is cheaper than the mincut. Refuting our assumption we prove the claim. ◀

■ **Algorithm 2** Details to Line 4 in Algorithm 1.

Input: A connected weighted graph $G = (V, E, \omega)$, a vertex subset $A \subseteq V$, and a mincut $c_A = (A_1, A_2)$ for $G[A]$.

Output: A largest applicable set $S \subseteq A$ with $S \cap A_1 \neq \emptyset$ and $S \cap A_2 \neq \emptyset$ if existing, \emptyset otherwise.

```

1 Function LargestApplicableSetOverCut( $G, A, c_A$ )
2   if  $A$  is applicable then return  $A$ 
3   if Lemma 3.2 (e) excludes existence of  $S$  then return  $\emptyset$ 
4   if  $V = A$  then
5      $uv \leftarrow$  arbitrary non-edge in  $A_i$ ,  $i \in \{1, 2\}$ , with  $\omega(uv) < 0$ 
6      $A_j \leftarrow$  side of  $c_A$  not containing  $uv$ 
7     return  $\max(\text{ApplicableSet}(G, A_j, \{u\}), \text{ApplicableSet}(G, A_j, \{v\}))$ 
8   else
9      $A_j \leftarrow$  side of  $c_A$  with  $\text{def}_G(A_j) = 0$  and  $\text{cost}(A_j, V \setminus A) = 0$ 
10     $U \leftarrow$  vertices in  $A$  with neighbors in  $V \setminus A$  // thus  $U \cap A_j = \emptyset$ 
11    return  $\text{ApplicableSet}(G[A], A_j, U)$ 

```

```

12 Function ApplicableSet( $G, X, Y$ )
13   contract  $X$  into node  $s$  and  $Y$  into node  $t$  and construct PQ-DAG  $D$ 
14    $Q_0, Q_1, \dots, Q_p \leftarrow$  a reverse topological ordering of  $D$  //  $s \in Q_0$  and  $t \in Q_p$ 
15    $S \leftarrow \emptyset$ 
16   for  $i \leftarrow 0$  to  $q - 1$  do
17      $C_i \leftarrow$  vertices in  $G$  represented by  $\bigcup_{j \leq i} Q_j$ 
18     if  $Q_i$  is cut node in  $D$  and  $C_i$  is applicable then  $S \leftarrow C_i$ 
19   return  $S$ 

```

Using Lemma 3.4, we can contract all vertices except Q_i before each mincut computation of S_i . Hence, we can efficiently compute the mincut for each candidate set S , resulting in the following overall theorem (see Algorithm 2 for pseudocode). We denote with $T_{\text{mincut}}(n, m)$ and $T_{s-t-\text{maxflow}}(n, m)$ the time to compute in an edge-weighted graph with n vertices and m edges a mincut and a maximum s - t -flow, respectively.

► **Theorem 3.5.** *Reduction Rule 2.2 can be applied in $\mathcal{O}(n(T_{\text{mincut}}(n, m) + T_{s-t-\text{maxflow}}(n, m) + n^2))$ time.*

Proof. We use Algorithm 1 with Line 4 being implemented with Algorithm 2.

All we need to show for the correctness of Algorithm 1 is that Algorithm 2 is correct. To this end, we need to show that if there is an applicable set S over the cut c_A , then Algorithm 2 will return such a set. (Note that Algorithm 2 never returns a non-applicable set as applicability is checked before returning a set.) By Lemma 3.3, the set S is characterized by a cut node in the PQ-DAG with s representing the side of c_A with deficit zero and t representing the vertices with neighbors in $V \setminus A$ (or t representing an endpoint of a non-edge with non-zero weight if $V = A$). Hence, all sets that could possibly be applicable are considered in Line 18. Thus, if an applicable set S exists, then an applicable set is returned.

It remains to show the claimed running time. To this end, start with Algorithm 1: If Lines 2 to 4 can be done in $\mathcal{O}(T_{\text{mincut}}(n, m) + T_{s-t-\text{maxflow}}(n, m) + n^2)$ time, then the claimed running time follows as there are at most $\mathcal{O}(n)$ recursive calls. The only nontrivial work in Lines 2 and 3 is the computation of a mincut. This can be done in $\mathcal{O}(T_{\text{mincut}}(n, m))$ time.

It remains to argue why Line 4 (that is, Algorithm 2) runs in $\mathcal{O}(T_{\text{mincut}}(n, m) + T_{s-t\text{-maxflow}}(n, m) + n^2)$ time: First, observe that the cost of the edges from A to $V \setminus A$ and of the edges from A_1 to A_2 can be simply transmitted in a recursive call of Algorithm 1. Hence, with simple bookkeeping, we can access $\text{cost}(A, V \setminus A)$ in constant time during the whole algorithm. Moreover, since to Algorithm 2 a mincut c_A of A is given, it follows that Lines 2 and 3 of Algorithm 2 can be done in $\mathcal{O}(n^2)$ time by simply iterating over all vertex pairs. The time to perform Lines 4 to 11 is dominated by the time required to execute the function in Line 12. Thus, it remains to show that this function can be computed in $\mathcal{O}(T_{\text{mincut}}(n, m) + T_{s-t\text{-maxflow}}(n, m) + n^2)$ time: Line 13 requires contraction of two vertex sets, the computation of one maximum s - t -flow, and the construction of the PQ-DAG from said flow. The maximum s - t -flow can be computed in $\mathcal{O}(T_{s-t\text{-maxflow}}(n, m))$ time (note that the contraction of X and Y results in a graph of size $\mathcal{O}(n + m)$). The contraction of the vertices, the construction of the PQ-DAG [25], and the computation of a reverse topological order can all be done in linear time. Thus, Lines 13 and 14 require $\mathcal{O}(T_{s-t\text{-maxflow}}(n, m))$ time.

It remains to argue that the for-loop in Lines 16 to 18 can be done in $\mathcal{O}(T_{\text{mincut}}(n, m) + n^2)$ time. The bottleneck here is the check whether the candidate set C_i is applicable in Line 18. By inequality (1) and Lemma 3.1, this involves checking for each C_i : (a) $\text{cost}(C_i, V \setminus C_i) = \text{cost}(C_i, A \setminus C_i)$, (b) $\text{def}_G(C_i) = 0$, (c) $\text{mincut}(G[C_i]) = \text{cost}(C_i, A \setminus C_i)$. By construction of the PQ-DAG we have $\text{cost}(C_i, V \setminus C_i) = \text{cost}(C_i, A \setminus C_i)$ (only vertices in $Y \subseteq Q_p$ can have neighbors outside A). For (b) we only need to check vertex pairs within C_i that were not checked in the previous iteration C_{i-1} as $C_{i-1} \subseteq C_i$. Thus, for all sets C_0, \dots, C_{q-1} this can be done in $\mathcal{O}(n^2)$. (In fact, if the first set C_i has deficiency larger than zero, then this holds for all subsequent sets and the loop can be aborted.) It remains to show that (c) can be done in $\mathcal{O}(T_{\text{mincut}}(n, m))$ time: To this end, we only check if $\text{mincut}(G[C_i]) < \text{cost}(C_i, A \setminus C_i)$ as $\text{mincut}(G[C_i]) > \text{cost}(C_i, A \setminus C_i)$ would contradict $(C_i, A \setminus C_i)$ being a mincut of A . Exploiting Lemma 3.4, we contract all vertices in $C_i \setminus Q_i$ into one vertex x and compute a mincut c_x in the resulting graph $G[\{x\} \cup Q_i]$. If $\text{cost}(c_x) < \text{cost}(C_i, A \setminus C_i)$, then we know that C_i is not applicable as $\text{mincut}(G[C_i]) \leq \text{cost}(c_x)$. Otherwise, if $\text{cost}(c_x) \geq \text{cost}(C_i, A \setminus C_i)$, then, by Lemma 3.4, we know that $\text{mincut}(G[C_i]) = \text{cost}(C_i, A \setminus C_i)$ and C_i is applicable. Thus, the running time is

$$\sum_{i=1}^{p-1} T_{\text{mincut}}(|Q_i + 1|, |E(Q_i)| + |Q_i|) \in \mathcal{O}(T_{\text{mincut}}(n, n + m)) = \mathcal{O}(T_{\text{mincut}}(n, m))$$

as we assume the input graph to be connected. Thus, Reduction Rule 2.2 can be applied in $\mathcal{O}(n \cdot (T_{\text{mincut}}(n, m) + T_{s-t\text{-maxflow}}(n, m) + n^2))$ time. \blacktriangleleft

Since $T_{\text{mincut}}(n, m) \in \mathcal{O}(m^{1+o(1)})$ [20, 23] and $T_{s-t\text{-maxflow}}(n, m) \in m^{1+o(1)}$ for polynomially bounded capacities [12], it follows that Reduction Rule 2.2 can be applied in $\mathcal{O}(nm^{1+o(1)} + n^3)$ time for polynomially bounded weights.

► **Corollary 3.6.** *If all weights are polynomially bounded, then Reduction Rule 2.2 can be applied in $\mathcal{O}(nm^{1+o(1)} + n^3)$ time*

4 Experimental Evaluation

In this section, we discuss the effectiveness and the recursion behavior of Algorithm 1. To this end, we provide a basic implementation as proof of concept to demonstrate by how much several graphs can be reduced when Reduction Rule 2.2 is applied exhaustively. Moreover, we briefly analyze the recursion depth of Algorithm 1.

Implementation. We implemented Algorithm 1 in the programming language Julia. The implementation and the test setup can be found on Github¹.

Our implementation of Algorithm 1 resolves the conquer step of finding a candidate set over the mincut $c_A = (A_1, A_2)$ in Line 4 as follows: First A and the restrictions in Lemma 3.2 (e) are tested (exactly as in Lines 2 and 3 in Algorithm 2). Note that in *all* test instances one of the two if statements were triggered, that is, either the current set A was applicable or Lemma 3.2 (e) certifies that there is no applicable set over the cut c_A . Hence we did not implement the sophisticated approach presented in Algorithm 2. Instead, as a fallback, our implementation would find such an applicable set S by simply iterating over *all* mincuts (see discussion after Lemma 3.1).

If an applicable vertex set S is found in the conquer step (Line 4 of Algorithm 1), then we do *not* recurse (Line 5). Instead, we merge the vertices in S and run the algorithm again on the newly created graph. Thus, our implementation *exhaustively* applies Reduction Rule 2.2.

To compute one (and all) mincut(s) of a graph, we use the implementation by Henzinger et al. [19], which uses integer weighted graphs. Therefore our implementation also requires integer weighted input graphs.

Setup. To test the implementation, we used the weighted instances that were converted into unweighted instances for the *PACE Challenge 2021*². This dataset includes primarily biologically motivated graphs and additionally randomly generated graphs. For a more detailed description of the dataset we refer to the PACE report [21].

We treat each connected component of the test graphs as a single graph. We only tested the algorithms on graphs with 100 or more vertices and only graphs which are not cluster graphs already, as such instances are easy to solve. This resulted in 204 different graphs from the PACE challenge dataset, with 150 real-world instances and 54 randomly generated instances. The largest graphs have around 3,000 vertices. The edge weights are floating-point values. As our implementation uses integer weights, we multiplied the edge weights by a factor of 1000 and rounded afterwards.

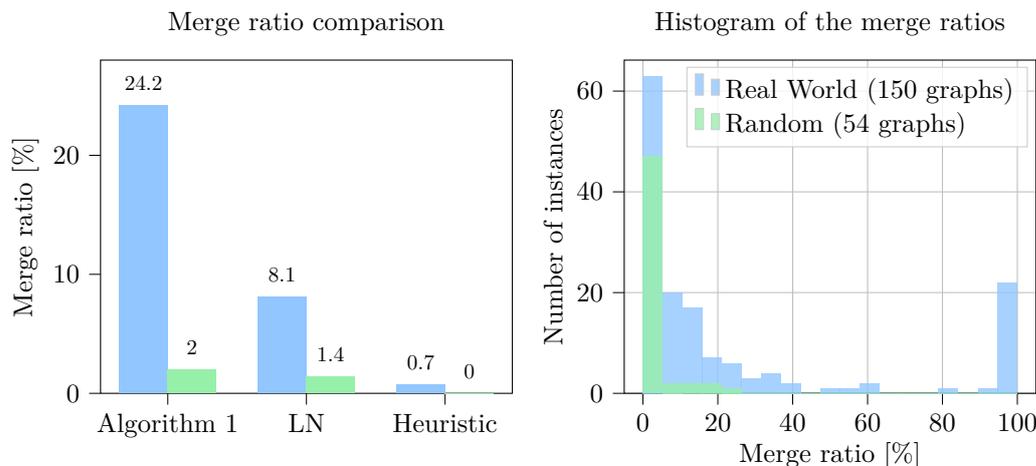
We also implemented two other approaches for finding applicable sets to compare the results of our algorithm. The first one is the *Large Neighborhood* (LN) approach, similar to the data reduction rule used by Cao and Chen [9] in their $2k$ -vertex kernel for WEIGHTED CLUSTER EDITING. They essentially test for every vertex $u \in V$ whether the closed neighborhood $N[u]$ is applicable. The second one, simply called *Heuristic*, is the implementation of the heuristic presented by Böcker et al. [8] for applying Reduction Rule 2.2, which we embedded in our implementation. Both of these approaches are run exhaustively. Note that both these approaches can fail to find applicable sets although the graph contains such a set, see Figure 2 for an example where the *Large Neighborhood* approach fails.

4.1 Results

Effectiveness. We were first interested in how much the graph size shrank after applying the data reduction rule with the various algorithms. As one can see in Figure 6 (left side), Algorithm 1 merges 24.2% of the vertices of the real-world instances, roughly three times the amount of vertices, compared to the LN approach, which merges 8.1% of the vertices. The algorithms perform very poorly on the random instances, with nearly the same amount

¹ <https://github.com/venondev/AlmostCliquePoly>

² The scripts for collecting and converting the graphs can be found at <https://github.com/PACE-challenge/Cluster-Editing-PACE-2021-instances>.



■ **Figure 6** *Left:* Comparison of the average merge ratio (number of vertices merged / n) of the three algorithmic approaches Algorithm 1, “Large Neighborhood” (LN) by Cao and Chen [9], and “Heuristic” by Böcker et al. [8] on the two instance categories real world and random. *Right:* Two histograms (blue behind green) showing the number of instances with respect to the merge ratio (number of vertices merged / n) for Algorithm 1. Note that the input graphs were not cluster graphs.

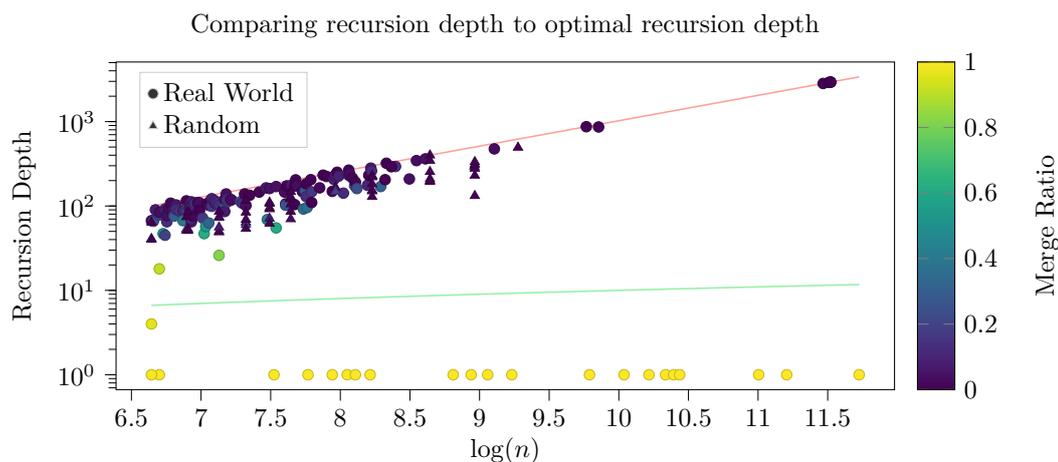
of merged vertices, 1.4% using the LN approach and 2% using Algorithm 1. The heuristic implementation by Böcker et al. [8] does not perform well on the dataset, with 0.7% for real-world instances and 0% for random instances.

When looking at the histogram of merge ratios for Algorithm 1 in Figure 6 (right side), it shows that for most graphs of the test set the rule reduces the graph size only slightly or not at all. But there are also 23 graphs which got solved *almost* entirely, that is, the resulting instance contained at most 5% of the initial number of vertices. When using Algorithm 1, 11.3 % of the instances got solved completely, around 1.5 % when using the LN approach, and 0.5% when using the heuristic by Böcker et al. [8].

Recursive calls of Algorithm 1. The depth of the recursion of Algorithm 1 is between $\log(n)$ (splitting the graph in half in each recursive call) and n (cutting exactly one vertex off in each recursive call). In our implementation the depth can be smaller than $\log(n)$: If we find an applicable set, then we merge it and run the algorithm again. The reported recursion depth is then the maximum over all runs on the instance.

Unfortunately, we observe that for most instances the recursion depth is close to n , see Figure 7 for an overview. This means that in most recursion steps, the mincut only cuts out a single vertex. Consequently, the size of the set $A \subseteq V$ that the algorithm looks at within each recursion step only decreases slowly. As a result, on most on the instances Algorithm 1 computes many mincuts on large graphs, resulting in high running times.

A preliminary test underlined the issue with the running time: For most of the instances, our basic implementation of Algorithm 1 is more than ten times slower than the LN approach. For some instances, our basic implementation is up to 1000 times slower.



■ **Figure 7** Comparing the recursion depth of the instances. If Algorithm 1 was run multiple time on an instance (due to merging some an applicable vertex set), then the maximum recursion depth over all runs is plotted. The red line denotes the upper bound of the recursion depth, which is the number of vertices n . If the algorithm does not find a set of vertices to merge, then the green line denotes the lower bound $\log(n)$ of the recursion depth. The instances are colored according to their respective merge ratio (number of vertices merged / n). The only instances below (or close) to the green line are instances with merge ratio close to one.

4.2 Summary

Our experiments show that Reduction Rule 2.2 can work well on the test dataset, reducing the real-world graphs by 24% on average if applied exhaustively. In this regard Algorithm 1 outperforms the other approaches. Notably, Algorithm 1 solves 11% of the graphs completely, including some larger graphs with up to 3,000 vertices. Unfortunately, Algorithm 1 still comes at the cost of a high running time. Hence, we suggest that (an improved implementation of) Algorithm 1 could be used as preprocessing before running a branch&bound solver but probably not during branching itself.

One avenue for improving the implementation is the computation of a mincut. Currently, if a mincut separates one vertex from the graph, then in the next recursive call the algorithm of Henzinger et al. [19] is cold-started to compute a new mincut on the slightly altered graph. Here the use of, for example, the dynamic algorithm of Henzinger et al. [18] seems promising.

5 Conclusion

In this work we provided the first polynomial-time algorithm to apply Reduction Rule 2.2. As the current running time is still quite high, an immediate open question is about better theoretical guarantees. Dealing with the bad recursive behavior would be a first step in this direction. Another question is whether our algorithm could be made working with approximate cuts instead of (optimum) mincuts. On the practical side, our experiments demonstrate the potential effectiveness of Reduction Rule 2.2 in real world instances, if applied exhaustively. Thus, the question is whether there are better tradeoffs between effectiveness and efficiency and whether state-of-the-art solvers [5] would benefit.

References

- 1 Faisal N. Abu-Khzam. On the complexity of multi-parameterized cluster editing. *Journal of Discrete Algorithms*, 45:26–34, 2017.
- 2 Faisal N. Abu-Khzam, Judith Egan, Serge Gaspers, Alexis Shaw, and Peter Shaw. Cluster editing with vertex splitting. In *Proceedings of the 5th International Symposium on Combinatorial Optimization (ISCO '18)*, volume 10856 of *LNCS*, pages 1–13. Springer, 2018.
- 3 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56:89–113, 2004. doi:10.1023/B:MACH.0000033116.57574.95.
- 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 Thomas Bläsius, Philipp Fischbeck, Lars Gottesbüren, Michael Hamann, Tobias Heuer, Jonas Spinner, Christopher Weyand, and Marcus Wilhelm. A branch-and-bound algorithm for cluster editing. In *Proceedings of the 20th International Symposium on Experimental Algorithms (SEA '22)*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- 6 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.
- 7 Sebastian Böcker and Jan Baumbach. Cluster Editing. In *Proceedings of the 9th Conference on Computability in Europe, CiE 2013*, volume 7921 of *LNCS*, pages 33–44. Springer, 2013.
- 8 Sebastian Böcker, Sebastian Briesemeister, and Gunnar W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011. doi:10.1007/s00453-009-9339-7.
- 9 Yixin Cao and Jianer Chen. Cluster Editing: Kernelization based on edge cuts. *Algorithmica*, 64(1):152–169, 2012.
- 10 Jianer Chen and Jie Meng. A $2k$ kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211–220, 2012.
- 11 Jiehua Chen, Hendrik Molter, Manuel Sorge, and Ondrej Suchý. Cluster editing in multi-layer and temporal graphs. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC '18)*, volume 123 of *LIPIcs*, pages 24:1–24:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
- 12 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *CoRR*, abs/2203.00671, 2022. doi:10.48550/arXiv.2203.00671.
- 13 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.
- 14 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of Cluster Editing with a small number of clusters. *Journal of Computer and System Sciences*, 80(7):1430–1447, 2014.
- 15 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.
- 16 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.
- 17 Sepp Hartung and Holger H. Hoos. Programming by optimisation meets parameterised algorithmics: a case study for cluster editing. In *Proceedings of the 9th International Conference on Learning and Intelligent Optimization, LION 2015*, volume 8994 of *LNCS*, pages 43–58. Springer, 2015. doi:10.1007/978-3-319-19084-6_5.
- 18 Monika Henzinger, Alexander Noe, and Christian Schulz. Practical fully dynamic minimum cut algorithms. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX 2022)*, pages 13–26. SIAM, 2022. doi:10.1137/1.9781611977042.2.

25:14 Applying a Cut-Based Data Reduction Rule in Polynomial Time

- 19 Monika Henzinger, Alexander Noe, Christian Schulz, and Darren Strash. Finding all global minimum cuts in practice. In *Proceedings of the 28th Annual European Symposium on Algorithms (ESA '20)*, volume 173, pages 59:1–59:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 20 David R Karger. Minimum cuts in near-linear time. *Journal of the ACM*, 47(1):46–76, 2000.
- 21 Leon Kellerhals, Tomohiro Koana, André Nichterlein, and Philipp Zschoche. The PACE 2021 Parameterized Algorithms and Computational Experiments Challenge: Cluster Editing. In *Proceedings of the International Symposium on Parameterized and Exact Computation (IPEC '21)*, volume 214, pages 26:1–26:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.IPEC.2021.26.
- 22 Christian Komusiewicz and Johannes Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012.
- 23 Jason Li. Deterministic mincut in almost-linear time. *CoRR*, abs/2106.05513, 2021.
- 24 Hiroshi Nagamochi, Yoshitaka Nakao, and Toshihide Ibaraki. A fast algorithm for cactus representations of minimum cuts. *Japan Journal of Industrial and Applied Mathematics*, 17(2):245–264, 2000.
- 25 Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. In *Combinatorial Optimization II*, pages 8–16. Springer, 1980.
- 26 Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.
- 27 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.

The PACE 2022 Parameterized Algorithms and Computational Experiments Challenge: Directed Feedback Vertex Set

Ernestine Großmann  

Universität Heidelberg, Germany

Tobias Heuer  

Karlsruhe Institute of Technology, Germany

Christian Schulz  

Universität Heidelberg, Germany

Darren Strash  

Hamilton College, Clinton, NY, USA

Abstract

The Parameterized Algorithms and Computational Experiments challenge (PACE) 2022 was devoted to engineer algorithms solving the NP-hard Directed Feedback Vertex Set (DFVS) problem. The DFVS problem is to find a minimum subset $X \subseteq V$ in a given directed graph $G = (V, E)$ such that, when all vertices of X and their adjacent edges are deleted from G , the remainder is acyclic.

Overall, the challenge had 90 participants from 26 teams, 12 countries, and 3 continents that submitted their implementations to this year's competition. In this report, we briefly describe the setup of the challenge, the selection of benchmark instances, as well as the ranking of the participating teams. We also briefly outline the approaches used in the submitted solvers.

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

Keywords and phrases Feedback Vertex Set, Algorithm Engineering, FPT, Kernelization, Heuristics

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.26

Funding Partially supported by DFG grant SCHU 2567/3-1.

Acknowledgements The PACE challenge was supported by Networks [1]. The prize money (€4000) was generously provided by Networks [1], 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.

1 Introduction

Over the last two decades, significant advances have been made in the design and analysis of fixed-parameter algorithms for a wide variety of graph-theoretic problems. This has resulted in an algorithmic toolbox that is by now well-established. Recently, these theoretical algorithmic ideas have received attention from the practical perspective [2, 3, 25, 40, 50]. A large part of this effort is driven by the Parameterized Algorithms and Computational Experiments Challenge (PACE) which was conceived in Fall 2015 to deepen the relationship between parameterized algorithms and practice. Topics from multivariate algorithms, exact algorithms, fine-grained complexity, and related fields are in scope. The mission of PACE is to bridge the divide between the theory of algorithm design and analysis, and the practice of



© Ernestine Großmann, Tobias Heuer, Christian Schulz, and Darren Strash;
licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 26; pp. 26:1–26:18

Leibniz International Proceedings in Informatics



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

algorithm engineering, inspire new theoretical developments, investigate in how far theoretical algorithms from parameterized complexity and related fields are competitive in practice, produce universally accessible libraries of implementations and repositories of benchmark instances as well as to encourage the dissemination of these findings in scientific papers. In each iteration of the challenge [20, 21, 12, 24, 51, 47] participants of the competition have been asked to provide implementations for one or two specifically chosen problems. Moreover, there are often two types of tracks: a track in which participants have to provide algorithms that solve a problem to optimality and a track in which heuristic solvers are allowed (and solutions are ranked accordingly). The challenge tackled already a wide range of problems. In previous iterations, the challenge tackled the following problems:

- First Iteration: Treewidth and Undirected Feedback Vertex Set [20]
- Second Iteration: Treewidth and Minimum Fill-In [21]
- Third Iteration: Steiner Tree [12]
- Fourth Iteration: Vertex Cover and Hypertree Width [24]
- Fifth Iteration: Treedepth [51]
- Sixth Iteration: Cluster Editing [47]

Since its inception, PACE challenges have established themselves as highly competitive with typically around 50 participants submitting their solvers from all over the world. Moreover, the challenges have already had a significant impact on the community as a whole. There is a wide range of research articles based on concrete implementations competing in previous editions of PACE that were published in prestigious conferences on algorithm engineering such as ACDA, ALENEX, ESA Track B, SEA, and WADS. Moreover, the challenges already successfully inspired new research, i.e. after the challenge there are also new results that improve on the previously best implementations from a particular challenge.

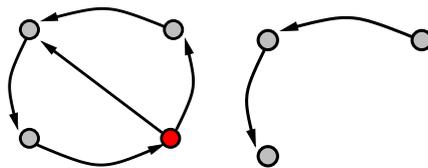
In this article, we report on the seventh iteration of the PACE implementation challenge. The problem chosen for this year's iteration has been the directed feedback vertex set problem. The challenge featured two tracks: an exact track and a heuristic track. In the *exact track*, the task was to find an optimal solution of each directed feedback vertex set instance within a time limit of 30 minutes. In the *heuristic track*, the task was to compute a valid solution that was as small as possible within a time limit of ten minutes.

The PACE 2022 challenge was announced and tracks were specified in September 2021. In January 2022, public instances were made available to the challenge participants. In March 2022, challenge participants were able to submit their solvers into the `optil.io` platform in which they could test their solvers on the instances that were publicly available. The platform also provided a provisional ranking. The final version of the submissions was due 1st June 2022. Afterwards, the submissions were evaluated on the publicly available as well as the private (hidden) instances. The results were announced in July 2022. The award ceremony took place during the International Symposium on Parameterized and Exact Computation (IPEC 2022).

2 Directed Feedback Vertex Set

The Directed Feedback Vertex Set (DFVS) problem is to find a minimum subset $X \subseteq V$ in a given directed graph $G = (V, E)$ such that, when all vertices of X and their adjacent edges are deleted from G , the remainder is acyclic. Thus a feedback vertex set of a graph is a set of vertices whose deletion leaves a graph acyclic. Figure 1 shows an example.

The DFVS problem has a wide range of applications including deadlock resolution [33], program verification [57] and VLSI chip design [54]. The decision variant of DFVS (asking if there exists a feedback vertex set of size at most k) is NP-complete [46] even if restricted to



■ **Figure 1** An input graph and a feedback vertex set (red) is shown on the left. In this example, deleting/removing the red vertex and its edges in the left graph results in the graph on the right hand side and leaves the remaining graph without any cycles.

graphs with maximum in- and out-degree two. The optimization variant of DFVS can be solved in $O^*(1.9977^n)$ time due to an algorithm by Razgon [69]. Chen et al. [17] showed that the problem is fixed-parameter tractable if parameterized with the solution size k , giving an algorithm with running time $O(4^k k! k^3 n^4) = 4^k k! n^{O(1)}$. With improvements to solving the Skew Edge Multicut problem, this running time is reduced to $O(4^k k! k^4 nm)$ [18, Corollary 8.47]. Lokshtanov et al. developed an improved algorithm with running time $O(4^k k! k^5 (n+m))$ [60], which has only linear dependence on the input size. It is open whether DFVS has a polynomial kernel in k , however a polynomial kernel exists when parameterized on the feedback vertex set of the underlying undirected graph [10], and for the compound parameterization of k plus the size of a treewidth- η modulator for any constant η [61]. Note that the problem is equivalent to the edge-deletion variant commonly called Feedback Arc Set: there are reductions in both directions that preserve the value of the optimal solution and only increase the size of the graph (sum of vertices and edges) by a polynomial factor [27]. Faster algorithms exist for undirected graphs [49, 76, 29], as well as for orientations of complete graphs (called *tournament* graphs) [34, 29].

The best approximation algorithm for DFVS is due to Even et al. [27], who gave an algorithm with approximation factor $O(\min\{\log \tau^* \log \log \tau^*, \log n \log \log n\})$ where τ^* is a lower bound, such as the optimal fractional solution in the LP relaxation. By Karp's reduction [46], DFVS is APX-hard, meaning that there is no polynomial-time approximation scheme (PTAS) for DFVS assuming $P \neq NP$. Furthermore, assuming the Unique Games Conjecture, DFVS does not admit a polynomial-time constant factor approximation [37, 38]. However, Lokshtanov et al. give a 2-approximation algorithm for DFVS when the input is a tournament graph [59].

3 Challenge Setup

There were two tracks in which the participants could compete: an exact and a heuristic track. For each track the 200 instances were selected by the Program Committee (PC), half of them publicly available before the submission deadline. The instances were sorted by the time our internal solvers needed to solve the instance. In the testing phase the instances were evaluated on the online judging platform optil.io [75]. For the final evaluation, we tested the instances on a local machine: an AMD EPYC 7702P 64-Core CPU, 200W, 2.00GHz, 256MB L3 Cache, DDR4-3200, Turbo Core max. 3.35GHz. Both evaluations used the same time limits: 30 minutes for the exact track and 10 minutes for the heuristic track.

3.1 Track Descriptions

The exact and the heuristic track followed essentially the same rules as in previous iterations of PACE. We now shortly describe the tracks:

Exact Track. In this track submissions had to find an optimal (minimum) feedback vertex set within 30 minutes. We expected each submission to be an exact algorithm, although we did not ask for proof of it. If we found through code checks or experiments that the algorithm of a submission is not an exact algorithm, it was excluded from the track. If for some instance the program returned a solution that has not been optimal within the time limit, either because it is not minimum or not a feedback vertex set, then the submission has been disqualified. The ranking has been determined by the number of solved instances. In case of a tie, the winner has been determined by the time required to solve all instances.

Heuristic Track. In the heuristic track, submissions had to provide a feedback vertex set within 10 minutes for a given instance. The submissions have been ranked by the geometric mean over all instances of $100 \times \frac{\text{best solution size}}{\text{solution size}}$. Here, solution size is the size of the solution returned by the submission and best solution size is the size of the smallest solution known to the PC (which may not be optimal). If the output of the program turned out to be not a feedback vertex set (or there is no output on SIGKILL) for some instance, solution size for the instance has been considered as $|V|$.

3.2 Internal Solver

Our goal was to create instances that are easy to solve, as well as instances that are as challenging as possible for the submissions. We implemented several data reduction rules and heuristics which we then used in our ILP solver to compute optimal solutions for the instances described in Section 3.3. We will now give a brief description of the algorithms.

Reduction Rules. We implemented four data reduction rules which are also summarized in the work of Lin and Jou [58]. We first compute all strongly connected components (SCC) using Tarjan’s algorithm [74] and remove all edges connecting two SCCs. We then solve each SCC separately. Further, we contract each node u with in-degree or out-degree one onto its unique predecessor or successor v . Intuitively, all cycles containing u also contain v . The last reduction rule creates an auxiliary graph G' by removing all undirected edges from the original graph (an edge $\{u, v\}$ is undirected if the graph contains the directed edges (u, v) and (v, u)) and then computes all SCCs of G' . An edge that connects two SCCs in G' can be removed from the original graph. For each undirected edge $\{u, v\}$ either u or v must be part of a DFVS (each undirected edge induces a cycle of size two) and therefore, the directed edges (u, v) and (v, u) are not part of the subgraph induced by removing any DFVS. Edges that connect two SCCs in G' are not part of a cycle when we remove one node of each undirected edge, and thus can be removed from the original graph. We apply the data reductions until none of them are applicable.

Random Walk Heuristic. A random walk on a directed graph $G = (V = \{v_1, \dots, v_n\}, E)$ can be modeled as a Markov chain with transition probabilities $p_{ij} = \frac{1}{d(v_i)}$ where $d(v_i)$ is the out-degree of node v_i . The stationary distribution $\pi = (\pi_1, \dots, \pi_n)$ of this Markov chain describes the probability distribution of visiting a node after a sufficiently long time (π is the solution of the linear equation $P\pi = \pi$). Moreover, π_i^{-1} represents the mean return time to node v_i in random walks. Thus, the stationary distribution π encodes information about the global structure of all cycles and a node v_i with the highest π_i value is very likely to be visited most frequently (and also contained in many cycles). A heuristic based on this idea was proposed by Lemaic and Speckenmeyer [70, 55]. We implemented a simple version of this algorithm that performs the random walk explicitly. We select a random start node and

visit $10|V|$ nodes. We remove the node that is visited most often. Afterwards, we recompute the SCC that the corresponding node was part of and remove all edges connecting two SCCs. The algorithm terminates if the graph is acyclic.

Maximum Acyclic Subgraph Heuristic. The DFVS problem is equivalent to finding a set V' of maximum cardinality such that the subgraph $G[V']$ is acyclic. The set $V \setminus V'$ is then a minimum DFVS. If a graph $G = (V, E)$ is acyclic, then there exists a topological ordering $T = \langle v_1, \dots, v_n \rangle$ of the nodes V such that for all edges $(v_i, v_j) \in E$ holds that $i < j$. Galinier et al. [32] propose a local search algorithm that constructs an acyclic subgraph $G[V']$ such that $|V'|$ is maximized. Consider an acyclic subgraph $G[V']$ with $V' \subseteq V$ and its topological ordering $T = \langle v_1, \dots, v_{n'} \rangle$ with $n' = |V'|$. If we insert a node $u \notin V'$ into T after the position i , we have to remove all nodes in $V_{\text{in}}(u, i) := \{v_j \in V' \mid (v_j, u) \in E \wedge j > i\}$ and $V_{\text{out}}(u, i) := \{v_j \in V' \mid (u, v_j) \in E \wedge j \leq i\}$ from T such that T still represents a valid topological ordering of the subgraph $G[V'']$ with $V'' = (V' \cup \{u\}) \setminus (V_{\text{in}}(u) \cup V_{\text{out}}(u))$. Thus, we can efficiently evaluate if a node $u \notin V'$ increases the cardinality of the acyclic subgraph $G[V']$ by computing the *gain* $g(u, i) := 1 - |V_{\text{in}}(u, i)| - |V_{\text{out}}(u, i)|$.

Our local search algorithm uses the well-known label propagation heuristic [68, 79]. The algorithm works in rounds and in each round it visits the nodes in random order. We initially start with an empty topological ordering T ($V' = \emptyset$). If we visit a node $u \notin V'$, we insert u into T after position i that maximizes $g(u, i)$ and remove all nodes in $V_{\text{in}}(u, i)$ and $V_{\text{out}}(u, i)$ from T . Note that we only evaluate positions in $\{i \mid (v_i, u) \in E \vee (u, v_i) \in E\}$ and add u to T if $g(u, i) \geq 0$. Further, insertions with $g(u, i) = 0$ naturally perturb the solution and we observed that this enables frequent improvements also in later iterations of the algorithm. The algorithm terminates if we reach a predefined number of rounds (= 200). To maintain the topological ordering, we use a sparse table priority queue implementation [42] that provides (amortized) constant time operations for access, insertions and removals of nodes.

We additionally made two major improvements to the original algorithm proposed by Galinier et al. [32]. Both exploit the fact that the topological ordering of the subgraph $G[V']$ is not unique and therefore provide some flexibility in the ordering of the nodes in T . If we are not able to insert a node u into the topological ordering T (i.e., $g(u, i) < 0$ for all possible positions i), we shift all nodes $v_i \in T$ adjacent to u via an in-arc $(v_i, u) \in E$ to the left and all nodes $v_j \in T$ adjacent to u via an out-arc $(u, v_j) \in E$ to the right in T (both as far as possible such that T still represents a valid topological ordering of $G[V']$). If then the indices of all nodes in T adjacent via an in-arc to u are smaller than the ones adjacent via an out-arc to u , we can increase the cardinality of the acyclic subgraph by inserting u in between. We further diversify the search by periodically computing a random topological ordering of $G[V']$ using Kahn's algorithm [45] (every fifth round).

Both techniques significantly improved the solution quality of the original algorithm (more than 10% on most instances). In practice, this heuristic was often an order of magnitude faster than our random walk algorithm. We also see more potential in this method as the concepts of gains allow the development of more sophisticated local search techniques.

Exact Solver. For the exact track, we solved the instances using a branch-and-cut ILP formulation and used Gurobi as a solver. Moreover, we integrate the data reduction rules described above into the solver. On the obtained irreducible instance, we run acyclic subgraph heuristics from above to get an initial feasible DFVS and provide the solution as an upper bound to the ILP solver that solves the following ILP to compute an optimal solution on the instance:

$$\begin{aligned} \min \quad & \sum_{v \in V} x_v \\ \text{s.t.} \quad & \sum_i x_{v_i} \geq 1 \quad \forall \text{ cycles } C = \{v_1, v_2, \dots, v_k\} \text{ in } G. \end{aligned}$$

Note that the number of constraints here can be exponential. As this would result in an intractable ILP, we add constraints *lazily* as follows. Initially our solver adds all constraints for cycles of length two. In addition, for each node $u \in V$, we add three cycle constraints representing cycles in which u is the node with smallest ID (ensures that the cycle constraints are distinct). We will call such a cycle an elementary cycle of u . We then solve the ILP using Gurobi and check if the solution is a feasible DFVS, i.e. after removal of the solution vertices there is no cycle remaining. If this is the case, then the solution is also an optimal solution to our input instance. If it is not a DFVS then after removing the vertices from the graph there must be a cycle. We then add for each node u in the remaining graph one additional elementary cycle and repeat the process.

Surprisingly, our exact solver was able to compute optimal solutions for some of the real-world instances in the heuristic track with up to $500k$ edges within a few minutes. However, the running time increases drastically for denser graphs (even if they contain only a few thousand edges). Thus, we believe that the density of a graph is a good indicator for the hardness of an instance.

3.3 Instances

We obtained instances from a wide range of different sources. In particular, as there is a wide range of random graph models available [66], we generated instances from different graph classes using *KaGen* [30], included several real-world instances from public graph repositories, and lastly generated instances that are hard for typical heuristic solvers such as heuristics based on random walks.

Generation and Selection Process. Our instance generation and selection process worked as follows: for both the exact and the heuristic track we generated a very large set of instances. From the graphs that are bidirected, we removed a random amount of edges $p \in \{0, 10, 20, 30, 40, 50\}$ to also obtain directed instances from those models. The amount of instances that we generated internally has been much larger the necessary 200 instances for the public and private set of instances for each track. The instances that we generated are described by a wide-range of parameters of different graph families (see below for more details). From the large set, we filtered instances that had more than 1 000 strongly connected components, less edges than nodes and excluded instances that had a file size above 50MB. On the remaining set of instances, we ran our exact and heuristic solvers. For the exact track, we then further excluded instances that our solver could handle in less than a second. Afterwards, we sampled instances uniformly at random. In particular, we included easy instances that could be solved within a couple of seconds as well as instances that were hard to solve. For the heuristic track, we tried to include instances that are hard for heuristics. From the instances that our exact solver could solve in this track, we included the ones where the result of heuristic solver had significantly more vertices than the optimum solution. Here, we also included real-world instances as well as instances designed to be hard for heuristics.

The instances we used can be categorized as follows:

Erdős-Rényi Graphs. The first version of the Erdős-Rényi (ER) model was proposed by Gilbert [35] and is denoted as the $G(n, p)$ model. Here, each of the $n(n-1)/2$ possible edges of an n -node graph is independently sampled with probability $0 < p < 1$ (Bernoulli sampling of the edges).

The second version, proposed by Erdős and Rényi [26], is denoted as the $G(n, m)$ model. In the $G(n, m)$ model, we choose a graph uniformly at random from the set of all possible graphs which have n vertices and m edges.

Random Geometric Graphs. Random geometric graphs (RGGs) are bidirected spatial networks where we place n vertices uniformly at random in a d -dimensional unit cube $[0, 1]^d$. Two vertices $p, q \in V$ are connected by an edge iff their d -dimensional Euclidean distance $\text{dist}(p, q) = \sqrt{\sum_{i=1}^d (p_i - q_i)^2}$ is within a given threshold radius r . Thus, the RGG model can be fully described using the two parameters n and r . Note that the expected degree of any vertex that does not lie on the border, i.e. whose neighborhood sphere is completely contained within the unit cube, is $\bar{d}(v) = \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)} r^d$ [65]. Here, we used two and three dimensional random geometric graphs as available in KaGen.

Random Hyperbolic Graphs. Random hyperbolic graphs (RHGs) are bidirected spatial networks generated in the hyperbolic plane with negative curvature. Analogous to RGGs, RHGs are parameterized by the number of vertices n and a hyperbolic radius $R = 2 \log n + C$.¹ Additionally, this model is given a power-law exponent $\gamma \geq 2$. To generate a RHG graph, n vertices are placed on a disk of radius R in the hyperbolic plane.

Each vertex has an angular coordinate ϕ and a radial coordinate r . The angular coordinate is sampled uniformly at random from the interval $[0, 2\pi)$. The radial coordinate r is chosen using the probability density function

$$f(r) = \alpha \frac{\sinh(\alpha r)}{\cosh(\alpha R) - 1}.$$

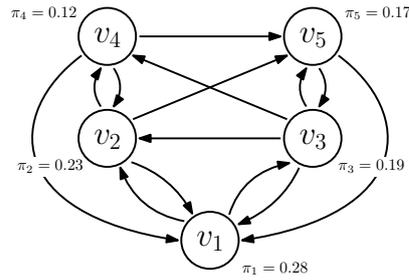
The parameter $\alpha = \frac{\gamma-1}{2}$ controls the growth of the random graph and determines the vertex density. Krioukov et al. [52, 36] show that for $\gamma \geq 2$ the degree distribution follows a power-law distribution with exponent γ . Two vertices p, q are connected iff their hyperbolic distance

$$\text{dist}_H(p, q) = \cosh r_p \cosh r_q - \sinh r_p \sinh r_q \cos |\phi_p - \phi_q|$$

is less than R . Therefore, the neighborhood of a vertex consists of all the vertices that are within the hyperbolic circle of radius R around it.

Random Delaunay Graphs. A d -simplex is a generalization of a triangle ($d = 2$) to d -dimensional space. A d -simplex s is a d -dimensional polytope, i.e. the convex hull of $d+1$ points. The convex hull of a subset of size $m+1$ of these $d+1$ points is called an m -face of s . Specifically, the 0-faces are the vertices of s and the $(d-1)$ -faces are its facets. Given a d -dimensional point set $V = \{v_1, v_2, \dots, v_n\}$ with $v_i \in \mathbb{R}^d$ for all $i \in \{1, \dots, n\}$, a triangulation $T(V)$ is a subdivision of the convex hull of V into d -simplices, such that the set of the vertices of $T(V)$ coincides with V and any two simplices of T intersect in a common

¹ The parameter C controls the average degree \bar{d} of the graph [52].



■ **Figure 2** Graph that is hard to solve for heuristics based on random walks. The π_i values denote the stationary distribution if we interpret the graph as a Markov chain with transition probabilities $p_{ij} = \frac{1}{d(v_i)}$.

$d - 1$ facet or not at all. The union of all simplices in $T(V)$ is the convex hull of point set V . A Delaunay triangulation $DT(V)$ is a triangulation of V such that no point of V is inside the circumsphere of any simplex in $DT(V)$.

Barabási-Albert Graph Model. Barabási and Albert [7] define the model that is perhaps most widely used because of its simplicity and intuitive definition: One starts with an arbitrary seed network consisting of nodes $0..n_0 - 1$ ($a..b$ is used as a shorthand for $\{a, \dots, b\}$ here). Nodes $i \in n_0..n - 1$ are added one at a time. They randomly connect to d neighbors using *preferential attachment*, i.e., the probability to connect to node $j \leq i$ is chosen proportionally to the degree of j . The seed graph, n_0 , d , and n are parameters defining the graph family. Since all edges only point to nodes with a smaller node ID the resulting directed network is acyclic. Hence, we generated graphs according to the Barabási Albert and modified them to become cyclic in the following way:

First, we computed a topological ordering of the instance. Then we inserted $p \cdot m$ random edges, where $p \in [0.05, 0.2]$. More precisely, we picked a random node to be a source, and afterwards picked a random node with a smaller number in the topological ordering.

Real-World Instances. This category includes instances from the SNAP [56] data set. We took 15 large directed graphs having between 7 115 and 2 394 385 nodes. In particular, we used web graphs, social networks, wikipedia graphs as well as purchase networks and autonomous system graphs. These instances were used in the heuristic track only.

Generated Hard Instances for Heuristic Solvers. In Figure 2, we show a graph where heuristics based on random walks choose a node that is not in the optimal DFVS with high probability. The optimal solution is to remove v_2 and v_3 . However, v_1 has the highest probability in the stationary distribution ($\pi_1 = 0.28$) and, if removed, leads to a DFVS of size 3. We create larger instances by replicating this five-node graph N times (optimal DFVS has size $2N$). We additionally connect the N copies with directed edges (10 edges per copy) such that the size of the optimal DFVS does not change and the probabilities in the stationary distribution of visiting v_2 and v_3 in each copy do not increase. Furthermore, we generate a random graph $G_R = (V_R, E_R)$ with $|V_R| \in [2.5N, 5N]$ nodes (chosen uniformly and at random) and an average degree of 5. We then connect each node $u \in V_R$ to a random node representing v_1 in one of the copies and connect the nodes $\{v_2, v_3, v_4, v_5\}$ of each copy to a random node $v \in V_R$. This hides the internal structure of the graph and adds some noise to the size of the DFVS. We generate 25 instances of this graph with $N \in [10^2, 10^5]$. In

our experiments, the size of the DFVS computed by our random walk algorithm was in most cases 1.5 times larger than the size of the optimal DFVS. The instances were used in the heuristic track only.

4 Participants and Results

There were 13 and 17 teams that officially submitted a solution to the exact and heuristic tracks, respectively. Several teams participated in more than one track; in total there were 26 distinct teams. The participants represented 3 continents and the following 12 countries (number of authors from the respective country is given in brackets): Germany (12), China (12), Czechia (10), France (7), Austria (5), India (5), Portugal (5), Norway (3), Romania (2), United States (2), Netherlands (1), Poland (1). The results are listed below.

4.1 Exact Track

The ranking for the exact track is listed subsequently; We list the number of solved instances from the 200 overall instances.

- **Rank 1** goes to solver raki123 having solved a total number of 185 instances; **Authors:** Andre Schidler and Rafael Kiesel; **Affiliation:** TU Wien; **URL** of solver: <https://github.com/ASchidler/dfvs>. **Zenodo:** [48]
- **Rank 2** goes to solver grapa-java having solved a total number of 165 instances; **Authors:** Enna Gerhard, Jona Dirks, Moritz Bergenthal, Jakob Gahde, Thorben Freese, Mario Grobler and Sebastian Siebertz; **Affiliation:** University of Bremen; **URL** of solver: <https://gitlab.informatik.uni-bremen.de/grapa/java/>. **Zenodo:** [9]
- **Rank 3** goes to solver mt-doom having solved a total number of 152 instances; **Authors:** Sebastian Angrich, Ben Bals, Niko Hastrich, Theresa Hradilak, Otto Kissig, Jonas Schmidt, Leo Wendt, Katrin Casel, Sarel Cohen and Davis Issac; **Affiliation:** Hasso Plattner Institute; **URL** of solver: <https://github.com/BenBals/mount-doom/tree/exact>. **Zenodo:** [4]
- **Rank 4** goes to solver goat_exact having solved a total number of 151 instances; **Authors:** Radovan Červený, Michal Dvořák, Xuan Thang Nguyen, Jan Pokorný, Lucie Procházková, Jaroslav Urban, Václav Blažej, Dušan Knop, Šimon Schierreich and Ondrej Suchy; **Affiliation:** Czech Technical University in Prague, Faculty of Information Technology; **URL** of solver: <https://gitlab.fit.cvut.cz/pace-challenge/2022/goat/exact>. **Zenodo:** [80]
- **Rank 5** goes to solver THS_exact having solved a total number of 140 instances; **Authors:** Henri Froese, Jonathan Guthermuth, Lars Huth, Marius Lotz, Johannes Meintrup, Timo Mertin, Manuel Penschuck and Hung Tran; **Affiliation:** Goethe University Frankfurt and THM, University of Applied Sciences Mittelhessen; **URL** of solver: <https://github.com/goethe-tcs/breaking-the-cycle>. **Zenodo:** [19]
- **Rank 6** goes to solver mndmky having solved a total number of 130 instances; **Authors:** Timon Behr; **Affiliation:** University of Konstanz; **URL** of solver: <https://github.com/mndmky/duck-and-cover>. **Zenodo:** [8]
- **Rank 7** goes to solver DUM having solved a total number of 125 instances; **Authors:** Henri Dickel, Matija Miskovic and Lennart Uhrmacher; **Affiliation:** Philipps-Universität Marburg; **URL** of solver: <https://github.com/HenriDickel/DFVS-Solver/tree/PACE>. **Zenodo:** [22]

26:10 PACE 2022: Directed Feedback Vertex Set

- **Rank 8** goes to solver yos having solved a total number of 120 instances; **Authors:** Yosuke Mizutani; **Affiliation:** University of Utah; **URL** of solver: <https://github.com/mogproject/dfvs-2022>. **Zenodo:** [64]
- **Rank 9** goes to solver rubengoetz having solved a total number of 88 instances; **Authors:** Ruben Götz; **Affiliation:** Karlsruher Institut für Technologie; **URL** of solver: <https://gitlab.com/rubenGoetz/dfvs-algo>. **Zenodo:** [39]
- **Rank 10** goes to solver DRIP having solved a total number of 32 instances; **Authors:** Aman Jain, Sachin Agarwal, Nimish Agrawal, Soumyajit Karmakar and Srinibas Swain; **Affiliation:** IIIT, Guwahati; **URL** of solver: <https://zenodo.org/record/6618812>. **Zenodo:** [43]

The following teams submitted a solver, but as described in the rules, their submissions were disqualified because of at least one suboptimal solution. Afterwards the teams sent us an updated version of their solver which then computed only correct results in the challenge. The number of solved instances reported below.

1. Solver Timeroot has solved a total number of 175 instances; **Authors:** Alexander Meiburg; **Affiliation:** UC Santa Barbara; **URL** of solver: https://github.com/Timeroot/DVFS_PACE2022/tree/pace-2022. **Zenodo:** [63], **ArXiv:** [62]
2. Solver swats has solved a total number of 160 instances; **Authors:** Sylwester Swat; **Affiliation:** Poznań University Of Technology; **URL** of solver: <https://github.com/swacisko/pace-2022>. **Zenodo:** [72]
3. Solver satanja has solved a total number of 144 instances; **Authors:** Stefan Tanja; **Affiliation:** Eindhoven University of Technology; **URL** of solver: <https://github.com/satanja/Hex>. **Zenodo:** [73]

Strategies Used in the Submissions

Winning Team. The approach by Andre Schidler and Rafael Kiesel [48] from TU Wien, Austria, applies a wide range of preprocessing techniques. These techniques stem a) from well-known reduction rules as well as b) non-trivial adaptations of reduction rules originally designed for the vertex cover problem. On the reduced instance, the team runs a MaxSAT solver that incrementally adds constraints.

Runner-Up. The approach by Enna Gerhard, Jona Dirks, Moritz Bergenthal, Jakob Gahde, Thorben Freese, Mario Grobler and Sebastian Siebertz from University of Bremen also applies a wide range of reduction rules to first decrease the size of the input. The team uses known as well as new reduction rules. Depending on the number of remaining undirected edges (forward and backward edges are present), the authors employ different strategies: a hitting set ILP formulation, an ILP formulation that models the problem as finding a topological order and if the later does not terminate within a specific time limit, the team runs a vertex cover solver to tackle the problem. If the solution of the last solver does not return a solution for the feedback vertex set problem, then no solution is returned.

Third Place. The team that achieved the third place are Sebastian Angrich, Ben Bals, Niko Hastrich, Theresa Hradilak, Otto Kissig, Jonas Schmidt, Leo Wendt, Katrin Casel, Sarel Cohen and Davis Issac from the Hasso Plattner Institute. As the first and second place, the team applies reduction rules first to reduce the input size. The team solves the remaining instance by repeatedly solving vertex cover instances. These instances are further reduced by the reductions of the PACE 2019 winning solver [41] and afterwards the instance is solved using a SAT-and-Reduce solver for the vertex cover problem [67].

4.2 Heuristic Track

The ranking for the heuristic track is listed subsequently; We list the score for the 200 overall instances. The score has been computed as outline in Section 3.1. Larger is better.

1. **Rank 1** goes to the solver swats with a score of 99.912; **Authors:** Sylwester Swat; **Affiliation:** Poznań University Of Technology; **URL of solver:** <https://github.com/swacisko/pace-2022>. **Zenodo:** [72]
2. **Rank 2** goes to the solver Nanored with a score of 99.911; **Authors:** Gabriel Bathie, Gaétan Berthe, Yoann Coudert-Osmont, David Desobry, Amadeus Reinald and Mathis Rocton; **Affiliation:** École normale supérieure de Lyon and Université de Lorraine, CNRS, Inria, LORIA; **URL of solver:** <https://github.com/Nanored4498/DreyFVS>. **Zenodo:** [31]
3. **Rank 3** goes to the solver hust_huawei with a score of 99.852; **Authors:** Yuming Du, Qingyun Zhang, Junzhou Xu, Shungen Zhang, Chao Liao, Zhihuai Chen, Zhibo Sun, Zhouxing Su, Junwen Ding, Chen Wu, Pinyan Lu and Zhipeng Lv; **Affiliation:** SMART, School of Computer Science and Technology, Huazhong University of Science & Technology and Huawei TCS Lab Shanghai; **URL of solver:** <https://github.com/1774150545/PACE-2022>. **Zenodo:** [77]
4. **Rank 4** goes to the solver KennethLangedal with a score of 99.832; **Authors:** Kenneth Langedal, Johannes Langguth and Fredrik Manne; **Affiliation:** University of Bergen and Simula Research Laboratory; **URL of solver:** <https://github.com/KennethLangedal/DFVS>. **Zenodo:** [53]
5. **Rank 5** goes to the solver fedrer with a score of 99.611; **Authors:** Aman Jain, Sachin Agarwal, Nimish Agrawal, Soumyajit Karmakar and Srinibas Swain; **Affiliation:** IIIT, Guwahati; **URL of solver:** <https://zenodo.org/record/6618777>. **Zenodo:** [44]
6. **Rank 6** goes to the solver Florian with a score of 99.435; **Authors:** Florian Sikora; **Affiliation:** LAMSADE; **URL of solver:** <https://github.com/fsikora/pace22>. **Zenodo:** [28]
7. **Rank 7** goes to the solver `_UAIC_ANDREIARHIRE_` with a score of 99.156; **Authors:** Andrei Arhire and Paul Diac; **Affiliation:** Alexandru Ioan Cuza University of Iași; **URL of solver:** <https://github.com/AndreeiArhire/PACE2022>. **Zenodo:** [6]
8. **Rank 8** goes to the solver INESCIDteam with a score of 98.619; **Authors:** Daniel Castro, Luis Russo, Aleksandar Ilic, Paolo Romano and Ana Correia; **Affiliation:** INESC-ID & IST; **URL of solver:** <https://github.com/Daniel1993/pace-2022>. **Zenodo:** [16]
9. **Rank 9** goes to the solver goat_heuristic with a score of 98.278; **Authors:** Radovan Červený, Michal Dvořák, Xuan Thang Nguyen, Jan Pokorný, Lucie Procházková, Jaroslav Urban, Václav Blažej, Dušan Knop, Šimon Schierreich and Ondrej Suchy; **Affiliation:** Czech Technical University in Prague, Faculty of Information Technology; **URL of solver:** <https://gitlab.fit.cvut.cz/pace-challenge/2022/goat/heuristic>. **Zenodo:** [81]
10. **Rank 10** goes to the solver orodruin with a score of 98.245; **Authors:** Sebastian Angrich, Ben Bals, Niko Hastrich, Theresa Hradilak, Otto Kißig, Jonas Schmidt, Leo Wendt, Katrin Casel, Sarel Cohen and Davis Issac; **Affiliation:** Hasso Plattner Institute, Potsdam, Germany and Digital Engineering Faculty, University of Potsdam, Potsdam, Germany; **URL of solver:** <https://github.com/BenBals/mount-doom/tree/heuristic>. **Zenodo:** [5]

11. **Rank 11** goes to the solver THS_heuristic with a score of 95.357; **Authors:** Jonathan Guthermuth, Lars Huth, Marius Lotz, Johannes Meintrup, Timo Mertin, Manuel Penschuck, Lukas Schwarz and Hung Tran; **Affiliation:** Goethe University Frankfurt and THM, University of Applied Sciences Mittelhessen; **URL** of solver: <https://github.com/goethe-tcs/breaking-the-cycle>. **Zenodo:** [19]
12. **Rank 12** goes to the solver grapa-rust with a score of 94.744; **Authors:** Ozan Can Heydt, Leon Stichternath, Kenneth Dietrich and Philipp Haker; **Affiliation:** Universität Bremen; **URL** of solver: <https://gitlab.informatik.uni-bremen.de/grapa/rust/mimung>. **Zenodo:** [71]
13. **Rank 13** goes to the solver dfvsp-julia with a score of 92.644; **Authors:** Maria Bresich, Günther Raidl and Johannes Varga; **Affiliation:** TU Wien; **URL** of solver: <https://github.com/NunuNoName/dfvsp-solver>. **Zenodo:** [13]
14. **Rank 14** goes to the solver grapa-java with a score of 91.369; **Authors:** Enna Gerhard, Jona Dirks, Moritz Bergenthal, Jakob Gahde, Thorben Frese, Mario Grobler and Sebastian Siebertz; **Affiliation:** Universität Bremen; **URL** of solver: <https://gitlab.informatik.uni-bremen.de/grapa/java/>. **Zenodo:** [9]
15. **Rank 15** goes to the solver BreakingCycles with a score of 74.613; **Authors:** Mert Biyikli; **Affiliation:** Heidelberg University; **URL** of solver: <https://github.com/MertBiyikli/BreakingCycles.git>. **Zenodo:** [11]

There were two more submissions from the team of the hust_huawei solver (Rank 3), however, only their best solver (hust_huawei) was ranked:

1. Solver xjz_huawei with a score of 99.651; **Authors:** Yuming Du, Qingyun Zhang, Junzhou Xu, Shungen Zhang, Chao Liao, Zhihuai Chen, Zhibo Sun, Zhouxing Su, Junwen Ding, Chen Wu, Pinyan Lu and Zhipeng Lv; **Affiliation:** SMART, School of Computer Science and Technology, Huazhong University of Science & Technology and Huawei TCS Lab Shanghai; **URL** of solver: <https://github.com/xuxu9110/PACE2022.git>. **Zenodo:** [23]
2. Solver adu with a score of 99.618; **Authors:** Yuming Du, Qingyun Zhang, Junzhou Xu, Shungen Zhang, Chao Liao, Zhihuai Chen, Zhibo Sun, Zhouxing Su, Junwen Ding, Chen Wu, Pinyan Lu and Zhipeng Lv; **Affiliation:** SMART, School of Computer Science and Technology, Huazhong University of Science & Technology and Huawei TCS Lab Shanghai; **URL** of solver: https://github.com/Zhang-qingyun/pace_2022_HUST_solver.git. **Zenodo:** [78]

Strategies Used in the Submissions

Winning Team. The winning solver was due to Sylwester Swat from Poznań University Of Technology. The solver reduces the input using data reduction rules. It then finds some initial solution of the reduced graph using fast heuristics. It then tries to improve the found solution by using a variety of heuristic approaches. In the end the solution is transferred to the input instance. More specifically, if the input instance is somewhat close to a bidirected graph, then fast vertex cover solvers NuMVC [15] and FastVC [14] are used to compute an initial solution. Another heuristic employed is based on agent flows. Specifically, each node is assigned a fixed number of tokens. Then the algorithm proceeds in rounds. In each round, each token assigned to a node is moved to a random out-neighbor. When all rounds are finished, the node having most tokens is added to the feedback vertex set and the process is repeated until the obtained graph is acyclic.

Runner-Up. The team scoring the second rank consists of Gabriel Bathie, Gaétan Berthe, Yoann Coudert-Osmont, David Desobry, Amadeus Reinald and Mathis Rocton from École normale supérieure de Lyon and Université de Lorraine CNRS, Inria, LORIA. After performing data reductions, their algorithm first performs a guess on the reduced instance by leveraging the Sinkhorn-Knopp algorithm. The solution is then improved by pipelining two local search methods. The first local search algorithm is a vertex swapping algorithms, i.e. the algorithms removes a vertex from the current solution and if this creates a cycle it adds a random vertex of the current cycle to the solution. If removing the vertex does not create a cycle, then the solution size has been improved by one. The second local search algorithm uses the fact that a digraph is acyclic if and only if a topological ordering can be computed. The team then shows that a unique feedback vertex set can be created from any topological ordering and thus obtain a local search method by shuffling vertices in the topological ordering.

Third Place. The team Yuming Du, Qingyun Zhang, Junzhou Xu, Shungen Zhang, Chao Liao, Zhihuai Chen, Zhibo Sun, Zhouxing Su, Junwen Ding, Chen Wu, Pinyan Lu and Zhipeng Lv from SMART, School of Computer Science and Technology, Huazhong University of Science & Technology as well as Huawei TCS Lab Shanghai scored the third place. As the other solvers, data reduction rules are applied to first reduce the size of the instance. Afterwards, the authors use a simulated annealing algorithm. To obtain an initial solution the authors first transform the problem into a vertex cover problem and then solve it using a heuristic for this problem. This is done using a time constraint. The time constraint depends on the number of bidirectional edges, i.e. the larger the fraction of bidirectional edges, the more time is assigned to the vertex cover heuristic. The local search used to improve the solution is based on topological orderings of the graph [32].

5 PACE Organization

The program committee of PACE 2022 consisted of Ernestine Großmann, Tobias Heuer, Christian Schulz (chair) and Darren Strash. During the organization of PACE 2022 the Steering Committee was as follows:

- (since 2016) Holger Dell (Goethe University Frankfurt and IT University of Copenhagen)
- (since 2019) Johannes Fichte (Technische Universität Dresden)
- (since 2019) Markus Hecher (Technische Universität Wien)
- (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 2020) Manuel Sorge (Technische Universität Wien)

6 Conclusion and Future Editions of PACE

We thank all the participants for their enthusiasm, strong and interesting contributions. Special thanks go to the participants who also presented at IPEC 2022. We are very happy that this edition attracted many people as well as strong contributions and hope that this will continue for future editions by considering popular problems to the community or even by repeating previously posted problems. As in previous challenges, we provided the public and private instance set in a public data library².

² <https://github.com/PACE-challenge/pacechallenge.org/tree/master/files>

We welcome anyone who is interested to add their name to the mailing list on the PACE website to receive updates and join the discussion. We look forward to the next edition. Detailed information will be posted on the website at pacechallenge.org. Also see the Twitter account³. In particular, plans for PACE 2023 will be posted there.

References

- 1 Networks project, 2017. URL: <http://www.thenetworkcenter.nl>.
- 2 Faisal N. Abu-Khzam, Sebastian Lamm, Matthias Mnich, Alexander Noe, Christian Schulz, and Darren Strash. Recent advances in practical data reduction. *Special Issue of SPP Big Data*, 2022. [arXiv:2012.12594](https://arxiv.org/abs/2012.12594).
- 3 N. Faisal Abu-Khzam, R. Michael Fellows, A. Michael Langston, and Henry W. Suters. Crown structures for vertex cover kernelization. *Theory Comput. Syst.*, 41(3):411–430, 2007. [doi:10.1007/s00224-007-1328-0](https://doi.org/10.1007/s00224-007-1328-0).
- 4 Sebastian Angrick, Ben Bals, Katrin Casel, Sarel Cohen, Niko Hastrich, Theresa Hradilak, Davis Issac, Otto Kißig, Jonas Schmidt, and Leo Wendt. Mount Doom – An Exact Solver for Directed Feedback Vertex Set, June 2022. [doi:10.5281/zenodo.6645235](https://doi.org/10.5281/zenodo.6645235).
- 5 Sebastian Angrick, Ben Bals, Katrin Casel, Sarel Cohen, Niko Hastrich, Theresa Hradilak, Davis Issac, Otto Kißig, Jonas Schmidt, and Leo Wendt. Orodruin — A Heuristic Solver for Directed Feedback Vertex Set, June 2022. [doi:10.5281/zenodo.6645245](https://doi.org/10.5281/zenodo.6645245).
- 6 Andrei Arhire and Paul Diac. `_UAIC_ANDREIARHIRE_` – A Heuristic Solver for the Directed Feedback Vertex Set Problem, June 2022. [doi:10.5281/zenodo.6646187](https://doi.org/10.5281/zenodo.6646187).
- 7 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. [doi:10.1126/science.286.5439.509](https://doi.org/10.1126/science.286.5439.509).
- 8 Timon Behr. Direfever, June 2022. [doi:10.5281/zenodo.6651761](https://doi.org/10.5281/zenodo.6651761).
- 9 Moritz Bergenthal, Jona Dirks, Thorben Freese, Jakob Gahde, and Enna Gerhard. GraPA-JAVA, June 2022. [doi:10.5281/zenodo.6647003](https://doi.org/10.5281/zenodo.6647003).
- 10 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. [doi:10.1007/s00453-020-00777-5](https://doi.org/10.1007/s00453-020-00777-5).
- 11 Mert Biyikli. MertBiyikli/BreakingCycles: Fourth release, June 2022. [doi:10.5281/zenodo.6674065](https://doi.org/10.5281/zenodo.6674065).
- 12 Édouard Bonnet and Florian Sikora. The PACE 2018 parameterized algorithms and computational experiments challenge: The third iteration. In Christophe Paul and Michal Pilipczuk, editors, *13th International Symposium on Parameterized and Exact Computation, IPEC 2018, August 20-24, 2018, Helsinki, Finland*, volume 115 of *LIPICs*, pages 26:1–26:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. [doi:10.4230/LIPICs.IPEC.2018.26](https://doi.org/10.4230/LIPICs.IPEC.2018.26).
- 13 Maria Bresich, Günther Raidl, and Johannes Varga. HyMeHeu-solver – A Hybrid Metaheuristic Solver for the Directed Feedback Vertex Set Problem, June 2022. [doi:10.5281/zenodo.6643236](https://doi.org/10.5281/zenodo.6643236).
- 14 Shaowei Cai, Jinkun Lin, and Chuan Luo. Finding a small vertex cover in massive sparse graphs: Construct, local search, and preprocess. *J. Artif. Intell. Res.*, 59:463–494, 2017. [doi:10.1613/jair.5443](https://doi.org/10.1613/jair.5443).
- 15 Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. NuMVC: An efficient local search algorithm for minimum vertex cover. *J. Artif. Intell. Res.*, 46:687–716, 2013. [doi:10.1613/jair.3907](https://doi.org/10.1613/jair.3907).
- 16 Daniel Castro. Daniel1993/pace-2022: pace-2022, June 2022. [doi:10.5281/zenodo.6634725](https://doi.org/10.5281/zenodo.6634725).
- 17 Jianer Chen, Yang Liu, Songjian Lu, Barry O’Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, 2008. [doi:10.1145/1411509.1411511](https://doi.org/10.1145/1411509.1411511).

³ https://twitter.com/pace_challenge

- 18 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.
- 19 Holger Dell, Henri Froese, Lukas Geis, Jonathan Guthermuth, Anselm Haak, Lars Huth, Frank Kammer, Marius Lotz, Johannes Meintrup, Timo Mertin, Manuel Penschuck, and Lukas Schwarz. BreakingTheCycle, June 2022. This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grants 379157101, ME 2088/5-1 (FOR 2975 — Algorithms, Dynamics, and Information Flow in Networks). doi:10.5281/zenodo.6602946.
- 20 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 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 30:1–30:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.IPEC.2016.30.
- 21 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 parameterized algorithms and computational experiments challenge: The second iteration. 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 30:1–30:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.IPEC.2017.30.
- 22 Dickel, Miskovic, and Uhrmacher. pace-2022-dum, May 2022. doi:10.5281/zenodo.6599645.
- 23 Yuming Du, Qingyun Zhang, Junzhou Xu, Shungen Zhang, Chao Liao, Zhihui Chen, Zhibo Sun, Zhouxing Su, Junwen Ding, Chen Wu, Pinyan Lu, and Zhipeng Lv. Huawei_tcs_dfvs_solver, June 2022. doi:10.5281/zenodo.6638370.
- 24 M. Ayaz Dzulfikar, Johannes Klaus Fichte, and Markus Hecher. The PACE 2019 parameterized algorithms and computational experiments challenge: The fourth iteration (invited paper). 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 25:1–25:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.IPEC.2019.25.
- 25 John D Eblen, Charles A Phillips, Gary L Rogers, and Michael A Langston. The maximum clique enumeration problem: algorithms, applications, and implementations. In *BMC Bioinformatics*, volume 13, page S5, 2012. doi:10.1186/1471-2105-13-S10-S5.
- 26 Paul Erdős and Alfréd Rényi. On Random Graphs I. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959 1959.
- 27 Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998. doi:10.1007/PL00009191.
- 28 Florian. fsikora/pace22: First version for pace, June 2022. doi:10.5281/zenodo.6624196.
- 29 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. *J. ACM*, 66(2), March 2019. doi:10.1145/3284176.
- 30 Daniel Funke, Sebastian Lamm, Ulrich Meyer, Manuel Penschuck, Peter Sanders, Christian Schulz, Darren Strash, and Moritz von Looz. Communication-free massively distributed graph generation. *J. Parallel Distributed Comput.*, 131:200–217, 2019. doi:10.1016/j.jpdc.2019.03.011.
- 31 Bathie Gabriel, Berthe Gaétan, Coudert-Osmont Yoann, Desobry David, Reinald Amadeus, and Rocton Mathis. Dreyfvs, June 2022. doi:10.5281/zenodo.6638217.
- 32 Philippe Galinier, Eunice Adjarath Lemamou, and Mohamed Wassim Bouzidi. Applying local search to the feedback vertex set problem. *J. Heuristics*, 19(5):797–818, 2013. doi:10.1007/s10732-013-9224-z.

- 33 Georges Gardarin and Stefano Spaccapietra. Integrity of data bases: A general lockout algorithm with deadlock avoidance. In G. M. Nijssen, editor, *Modelling in Data Base Management Systems, Proceeding of the IFIP Working Conference on Modelling in Data Base Management Systems, Freudenstadt, Germany, January 5-8, 1976*, pages 395–412. North-Holland, 1976.
- 34 Serge Gaspers and Matthias Mnich. Feedback vertex sets in tournaments. *J. Graph Theory*, 72(1):72–89, 2013. doi:10.1002/jgt.21631.
- 35 E. N. Gilbert. Random graphs. *Ann. Math. Statist.*, 30(4):1141–1144, December 1959. doi:10.1214/aoms/1177706098.
- 36 Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random hyperbolic graphs: Degree sequence and clustering. In *Automata, Languages, and Programming – 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 573–585. Springer, 2012. doi:10.1007/978-3-642-31585-5_51.
- 37 Venkatesan Guruswami, Johan HÅstad, Rajsekar Manokaran, Prasad Raghavendra, and Moses Charikar. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM Journal on Computing*, 40(3):878–914, 2011. doi:10.1137/090756144.
- 38 Venkatesan Guruswami and Euiwoong Lee. Simple proof of hardness of feedback vertex set. *Theory Comput.*, 12(1):1–11, 2016. doi:10.4086/toc.2016.v012a006.
- 39 Ruben Götz. Ruben Bachelor Thesis, June 2022. doi:10.5281/zenodo.6604728.
- 40 Monika Henzinger, Alexander Noe, Christian Schulz, and Darren Strash. Finding all global minimum cuts in practice. In *Proc. ESA 2020*, volume 173 of *Leibniz Int. Proc. Informatics*, pages 59:1–59:20, 2020. doi:10.4230/LIPIcs.ESA.2020.59.
- 41 Demian Hesse, Sebastian Lamm, Christian Schulz, and Darren Strash. WeGotYouCovered: The winning solver from the PACE 2019 challenge, vertex cover track. In H. Martin Bückler, Xiaoye Sherry Li, and Sivasankaran Rajamanickam, editors, *Proceedings of the SIAM Workshop on Combinatorial Scientific Computing, CSC 2020, Seattle, USA, February 11-13, 2020*, pages 1–11. SIAM, 2020. doi:10.1137/1.9781611976229.1.
- 42 A. Itai, A. G. Konheim, and M. Rodeh. A sparse table implementation of priority queues. In S. Even and O. Kariv, editors, *Proceedings of the 8th Colloquium on Automata, Languages and Programming*, volume 115 of *LNCS*, pages 417–431. Springer, 1981. doi:10.1007/3-540-10843-2_34.
- 43 Aman Jain, Sachin Agarwal, Nimish Agrawal, Soumyajit Karmakar, and Srinibas Swain. DRIP: Directed feedback vertex set computation using Reductions and Integer Programming, June 2022. doi:10.5281/zenodo.6618812.
- 44 Aman Jain, Sachin Agarwal, Nimish Agrawal, Soumyajit Karmakar, and Srinibas Swain. FEDRER: Feedback vertex set using Edge Density and REmove Redundant, June 2022. doi:10.5281/zenodo.6618777.
- 45 Arthur B. Kahn. Topological Sorting of Large Networks. *Commun. ACM*, 5(11):558–562, 1962. doi:10.1145/368996.369025.
- 46 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.
- 47 Leon Kellerhals, Tomohiro Koana, André Nichterlein, and Philipp Zschoche. The PACE 2021 parameterized algorithms and computational experiments challenge: Cluster editing. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPIcs*, pages 26:1–26:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.IPEC.2021.26.
- 48 Rafael Kiesel and Andre Schidler. DAGger – An Exact Directed Feedback Vertex Set Solver, June 2022. doi:10.5281/zenodo.6627405.

- 49 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Information Processing Letters*, 114(10):556–560, 2014. doi:10.1016/j.ipl.2014.05.001.
- 50 Viatcheslav Korenwein, André Nichterlein, Rolf Niedermeier, and Philipp Zschoche. Data reduction for maximum matching on real-world graphs: Theory and experiments. In *Proc. ESA 2018*, volume 112 of *Leibniz Int. Proc. Informatics*, pages 53:1–53:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.53.
- 51 Lukasz Kowalik, Marcin Mucha, Wojciech Nadara, Marcin Pilipczuk, Manuel Sorge, and Piotr Wygocki. The PACE 2020 parameterized algorithms and computational experiments challenge: Treedepth. 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 37:1–37:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.37.
- 52 Dmitri V. Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *CoRR*, abs/1006.5169, 2010. arXiv:1006.5169.
- 53 Kenneth Langedal. KennethLangedal/DFVS: pace-2022, June 2022. doi:10.5281/zenodo.6630611.
- 54 Charles E. Leiserson and James B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, 1991. doi:10.1007/BF01759032.
- 55 Mile Lemaic. *Markov-Chain-Based Heuristics for the Feedback Vertex Set Problem for Digraph*. PhD thesis, University of Cologne, 2008. URL: <http://kups.ub.uni-koeln.de/id/eprint/2547>.
- 56 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. <http://snap.stanford.edu/data>, June 2014.
- 57 Orna Lichtenstein and Amir Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In Mary S. Van Deusen, Zvi Galil, and Brian K. Reid, editors, *Conference Record of the Twelfth Annual ACM Symposium on Principles of Programming Languages, New Orleans, Louisiana, USA, January 1985*, pages 97–107. ACM Press, 1985. doi:10.1145/318593.318622.
- 58 Hen-Ming Lin and Jing-Yang Jou. On Computing the Minimum Feedback Vertex Set of a Directed Graph by Contraction Operations. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 19(3):295–307, 2000. doi:10.1109/43.833199.
- 59 Daniel Lokshantov, Pranabendu Misra, Joydeep Mukherjee, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. 2-approximating feedback vertex set in tournaments. *ACM Trans. Algorithms*, 17(2), April 2021. doi:10.1145/3446969.
- 60 Daniel Lokshantov, M. S. Ramanujan, and Saket Saurabh. When recursion is better than iteration: A linear-time algorithm for acyclicity with few error vertices. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '18*, pages 1916–1933, USA, 2018. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611975031.125.
- 61 Daniel Lokshantov, M. S. Ramanujan, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Wannabe bounded treewidth graphs admit a polynomial kernel for DFVS. In Zachary Friggstad, Jörg-Rüdiger Sack, and Mohammad R. Salavatipour, editors, *Algorithms and Data Structures – 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, volume 11646 of *Lecture Notes in Computer Science*, pages 523–537. Springer, 2019. doi:10.1007/978-3-030-24766-9_38.
- 62 Alex Meiburg. Reduction Rules and ILP Are All You Need: Minimal Directed Feedback Vertex Set, 2022. doi:10.48550/ARXIV.2208.01119.
- 63 Alexander Meiburg. PACE 2022 – DFVS-Via-Flattening Solver (DVFS), June 2022. Derived from https://github.com/Timeroot/DVFS_PACE_2022/tree/pace-2022. doi:10.5281/zenodo.6650921.
- 64 Yosuke Mizutani. PACE 2022 – Exact, June 2022. doi:10.5281/zenodo.6604875.

- 65 Mathew Penrose. *Random geometric graphs*. Number 5 in Oxford Studies in Probability. Oxford University Press, 2003.
- 66 Manuel Penschuck, Ulrik Brandes, Michael Hamann, Sebastian Lamm, Ulrich Meyer, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in scalable network generation. *CoRR*, abs/2003.00736, 2020. [arXiv:2003.00736](https://arxiv.org/abs/2003.00736).
- 67 Rick Plachetta and Alexander van der Grinten. SAT-and-reduce for vertex cover: Accelerating branch-and-reduce by SAT solving. In Martin Farach-Colton and Sabine Storandt, editors, *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2021, Virtual Conference, January 10-11, 2021*, pages 169–180. SIAM, 2021. [doi:10.1137/1.9781611976472.13](https://doi.org/10.1137/1.9781611976472.13).
- 68 U. N. Raghavan, R. Albert, and S. Kumara. Near Linear Time Algorithm to Detect Community Structures in Large-Scale Networks. *Physical Review E*, 76(3), 2007. [doi:10.1103/PhysRevE.76.036106](https://doi.org/10.1103/PhysRevE.76.036106).
- 69 Igor Razgon. Computing minimum directed feedback vertex set in $O^*(1.9977^n)$. In Giuseppe F. Italiano, Eugenio Moggi, and Luigi Laura, editors, *Theoretical Computer Science, 10th Italian Conference, ICTCS 2007, Rome, Italy, October 3-5, 2007, Proceedings*, pages 70–81. World Scientific, 2007. [doi:10.1142/9789812770998_0010](https://doi.org/10.1142/9789812770998_0010).
- 70 Ewald Speckenmeyer. On Feedback Problems in Digraphs. In *Graph-Theoretic Concepts in Computer Science, 15th International Workshop, WG '89, Castle Rolduc, The Netherlands, June 14-16, 1989, Proceedings*. Springer, 1989. [doi:10.1007/3-540-52292-1_16](https://doi.org/10.1007/3-540-52292-1_16).
- 71 Leon Stichternath, Ozan Heydt, Kenneth Dietrich, and Philipp Haker. Grapa-Rust, June 2022. [doi:10.5281/zenodo.6603799](https://doi.org/10.5281/zenodo.6603799).
- 72 Sylwester Swat. swacisko/pace-2022: First release of DiVerSeS, a solver for the Directed Feedback Vertex Set problem, June 2022. [doi:10.5281/zenodo.6643144](https://doi.org/10.5281/zenodo.6643144).
- 73 Stefan Tanja. satanja/hex: pace-2022, June 2022. [doi:10.5281/zenodo.6609797](https://doi.org/10.5281/zenodo.6609797).
- 74 Robert Endre Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. [doi:10.1137/0201010](https://doi.org/10.1137/0201010).
- 75 Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. Optil.io: Cloud based platform for solving optimization problems using crowdsourcing approach. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion, CSCW '16 Companion*, pages 433–436, New York, NY, USA, 2016. Association for Computing Machinery. [doi:10.1145/2818052.2869098](https://doi.org/10.1145/2818052.2869098).
- 76 Mingyu Xiao and Hiroshi Nagamochi. An improved exact algorithm for undirected feedback vertex set. *J. Comb. Optim.*, 30(2):214–241, 2015. [doi:10.1007/s10878-014-9737-x](https://doi.org/10.1007/s10878-014-9737-x).
- 77 YuMingDu, QingYunZhang, and ShunGenZhang. pace-2022, June 2022. [doi:10.5281/zenodo.6644409](https://doi.org/10.5281/zenodo.6644409).
- 78 Zhang-qingyun. Zhang-qingyun/pace_2022_HUST_solver: pace_2022_HUST_solver, June 2022. [doi:10.5281/zenodo.6643002](https://doi.org/10.5281/zenodo.6643002).
- 79 Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical Report CMU-CALD-02-107, Carnegie Mellon University, 2002. URL: <http://reports-archive.adm.cs.cmu.edu/anon/cald/CMU-CALD-02-107.pdf>.
- 80 Radovan Červený, Michal Dvořák, Xuan Thang Nguyen, Jan Pokorný, Lucie Procházková, Jaroslav Urban, Václav Blažej, Dušan Knop, Šimon Schierreich, and Ondřej Suchý. G2OAT solver for PACE 2022 (DFVS) exact track, June 2022. [doi:10.5281/zenodo.6637464](https://doi.org/10.5281/zenodo.6637464).
- 81 Radovan Červený, Michal Dvořák, Xuan Thang Nguyen, Jan Pokorný, Lucie Procházková, Jaroslav Urban, Václav Blažej, Dušan Knop, Šimon Schierreich, and Ondřej Suchý. G2OAT solver for PACE 2022 (DFVS) heuristic track, June 2022. [doi:10.5281/zenodo.6637495](https://doi.org/10.5281/zenodo.6637495).

PACE Solver Description: DiVerSeS – A Heuristic Solver for the Directed Feedback Vertex Set Problem*

Sylwester Swat  

Institute of Computing Science, Poznań University of Technology, Poland

Abstract

This article briefly describes the most important algorithms and techniques used in the directed feedback vertex set heuristic solver called “DiVerSeS”, submitted to the 7th Parameterized Algorithms and Computational Experiments Challenge (PACE 2022).

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Directed feedback vertex set, heuristic solver, graph algorithms, PACE 2022

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.27

Supplementary Material *Software (Source Code):*

<https://zenodo.org/record/6643144#.YqjL2r9ByV4>

Funding Supported by the Foundation for Polish Science (FNP).

1 Problem description

A feedback vertex set of a directed graph $G = (V, A)$ is a set $X \subset V$ such that the induced graph $G[V \setminus X]$ is acyclic. The solver briefly described here is a heuristic approach to the Directed Feedback Vertex Set (DFVS) problem, where the goal is to find a smallest possible feedback vertex set of a given directed graph. The DFVS problem can be also considered equivalently as a Maximum Directed Acyclic Subgraph problem, where for a given directed graph $G = (V, A)$ the task is to find a largest possible set $Y \subset V$ such that $G[Y]$ is a DAG.

2 Solver description

In this paper we provide a short description of the most important algorithms implemented in solver DiVerSeS. Due to a large variety of used methods, this description does not contain full information about used algorithms and their behaviour in many distinct situations. The workflow of DiVerSeS can be described in the following general steps:

1. Reduce the graph using data reduction rules.
2. Find some initial solution of a reduced graph using fast heuristics.
3. Improve found solution using a variety of heuristic approaches.
4. Lift the solution to create a final DFVS of the original graph.

Before we proceed to further description, let us introduce some notations. By $A(G)$ we denote the set of arcs of a directed graph G . An arc $(a, b) \in A$ is called a *pi-arc* if there is also an arc $(b, a) \in A$. A *pi-graph* of a graph G is a graph $pi(G) = (V, A_{pi})$, where $A_{pi} = \{(a, b) \in A : (a, b) \text{ is a pi-arc}\}$. A *nonpi-graph* of a graph G is a graph

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2022. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.



© Sylwester Swat;

licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 27; pp. 27:1–27:3

Leibniz International Proceedings in Informatics



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

$npi(G) = (V, A_{npi})$, where $A_{npi} = \{(a, b) \in A : (a, b) \notin A(pi(G))\}$. A *superpi-graph* of a graph G is a graph $spi(G) = (V, A_{spi})$, where $A_{spi} = \{(a, b) : (a, b) \in A \text{ or } (b, a) \in A\}$. Node v is called a *pi-node* if all arcs incident to it are pi-arcs.

3 Preprocessing

We use a large number of different data reduction rules. This includes an implementation of almost all data reduction rules known to us from the literature, modifications of some existing methods and a whole collection of new methods used to reduce the graph size or to modify the graph structure in some specific way (e.g. by adding some arcs, what might be seen as a little bit contradictory to the intuitive comprehension of a term 'data reduction'), from which some constitute a generalization of known data-reduction rules for the vertex-cover problem. To the most well known data reduction rules we can include those from [4] and [5]. These include the following:

1. IN0, OUT0 rules: removing nodes with empty in-neighborhood $N^-(v)$ or empty out-neighborhood $N^+(v)$.
2. IN1, OUT1 rules: merging each node v with $|N^-(v)| = 1$ or $|N^+(v)| = 1$ (by merging node v we mean adding to the graph, unless already present, all possible arcs from the set $N^-(v) \times N^+(v)$, then removing v from the graph).
3. PIE rule: removing all arcs (a, b) such that a and b belong to different strongly connected components in graph $npi(G)$.
4. DOME rule: removing from the graph all dominated arcs (see [5] for more details).
5. CORE rule: removing from the graph (and adding to constructed DFVS) all neighbors of a pi-node v whose neighborhood is a clique in $pi(G)$.

These are the most basic (but still very effective in practice) among rules implemented in DiVerSeS. Nevertheless, proper implementation (with a guarantee of best worst-case performance, but also minimizing a constant overhead factor) of some of those rules is not trivial. For example, authors of [5] claim that the CORE rule can be implemented in time $O(|A| + |V| \cdot \log|V|)$. We believe that arguments presented by the authors are either not correct or there is no proper explanation, and we were unable to implement the CORE algorithm with such time complexity (our implementation of the CORE rule works in time $O(|A|^{\frac{3}{2}})$). On the other hand the DOME rule can be implemented in time $O(|A|^{\frac{3}{2}})$ instead of $O(|A| \cdot |V|)$ proposed by the authors, what is a significant improvement for large sparse graphs that contain nodes with high degree.

4 Creating initial solution

There are many algorithms used to create an initial solution of a given graph G and are used depending on the graph characteristics. For example, if the fraction $\frac{|A(pi(G))|}{|A|}$ is high (specified by a parameter), then as a DFVS of G we simply take the vertex cover of $spi(G)$. To find a vertex cover we use NumVC [2] and FastVC [1] algorithms. If the fraction of pi-arcs is not high, then we use other methods. One of them is the *agent-flow* algorithm that works in the following way:

1. Assign a fixed number of tokens to each node.
2. In each of R (some small fixed integer) iterations, for each node and each token assigned to that node, move the token to some out-neighbor of a given node.
3. Add to constructed DFVS a node that contains most tokens after all R iterations and remove that node from the graph.
4. Repeat the procedure until the obtained graph is acyclic.

There are two variations of the agent-flow method: discrete and continuous. In the discrete version we have an integral number of tokens in each node and in each iteration we separately move each token to a random out-neighbor. In a continuous version we have a real-valued number of tokens and in each iteration we distribute the tokens evenly among all out-neighbors.

After a DFVS X of graph G is found, we remove redundant nodes from X (a node $x \in X$ is redundant if $X \setminus \{x\}$ is also a DFVS) using a very efficient algorithm making use of properties of dynamically changing topological order of an induced graph $G[V \setminus X]$.

5 Improvement of solution

When an initial solution is found, we try to improve it using various approaches. Which algorithm is used to improve the solution depends on some characteristics of graph G . For example, if G is a very sparse graph (but not too sparse), we use an efficient improvement of a simulated-annealing-based algorithm (description of the basic algorithm can be found in [3]). If the graph contains a high percentage of pi-arcs, then we use (among others) the following approach:

1. Find an ordering (v_1, \dots, v_N) of V with as few backgoing arcs as possible (an arc $(a, b) \in A$ is a backgoing arc if b precedes a in the ordering).
2. Take as a DFVS a vertex cover of a graph $H = (V, A_H)$, where A_H is the set containing all backgoing arcs.

There are a few ways implemented in DiVerSeS to create an ordering and they usually take into account structure of current DFVS. This way we can improve existing solution instead of just finding another one. It is also worth mentioning here that finding an optimal ordering (for which the number of backgoing arcs is minimum) is equivalent to finding an optimal solution of an instance of the Directed Feedback Arc Set Problem, which is NP-complete (and in certain sense equivalent to the DFVS problem).

6 Availability

The source code of DiVerSeS is freely available and can be found at <https://zenodo.org/record/6643144#.YqjL2r9ByV4>.

References

- 1 Shaowei Cai, Jinkun Lin, and Chuan Luo. Finding a small vertex cover in massive sparse graphs: Construct, local search, and preprocess. *Journal of Artificial Intelligence Research*, 59:463–494, July 2017. doi:10.1613/jair.5443.
- 2 Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. Numvc: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research*, 46, February 2014. doi:10.1613/jair.3907.
- 3 Philippe Galinier, Eunice Lemamou, and Mohamed Bouzidi. Applying local search to the feedback vertex set problem. *Journal of Heuristics*, 19, October 2013. doi:10.1007/s10732-013-9224-z.
- 4 Hanoch Levy and David W Low. A contraction algorithm for finding small cycle cutsets. *Journal of Algorithms*, 9(4):470–493, 1988. doi:10.1016/0196-6774(88)90013-2.
- 5 Hen-Ming Lin and Jing-Yang Jou. On computing the minimum feedback vertex set of a directed graph by contraction operations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(3):295–307, 2000. doi:10.1109/43.833199.

PACE Solver Description: Mount Doom – An Exact Solver for Directed Feedback Vertex Set*

Sebastian Angrick ✉

Hasso Plattner Institut,
Universität Potsdam, Germany

Katrin Casel ✉ 

Hasso Plattner Institut,
Universität Potsdam, Germany

Tobias Friedrich ✉ 

Hasso Plattner Institut,
Universität Potsdam, Germany

Theresa Hradilak ✉

Hasso Plattner Institut,
Universität Potsdam, Germany

Otto Kißig ✉ 

Hasso Plattner Institut,
Universität Potsdam, Germany

Leo Wendt ✉

Hasso Plattner Institut,
Universität Potsdam, Germany

Ben Bals ✉

Hasso Plattner Institut,
Universität Potsdam, Germany

Sarel Cohen ✉ 

The Academic College of Tel Aviv-Yaffo, Israel

Niko Hastrich ✉

Hasso Plattner Institut,
Universität Potsdam, Germany

Davis Issac ✉ 

Hasso Plattner Institut,
Universität Potsdam, Germany

Jonas Schmidt ✉

Hasso Plattner Institut,
Universität Potsdam, Germany

Abstract

In this document we describe the techniques we used and implemented for our submission to the Parameterized Algorithms and Computational Experiments Challenge (PACE) 2022. The given problem is *Directed Feedback Vertex Set* (DFVS), where one is given a directed graph $G = (V, E)$ and wants to find a minimum $S \subseteq V$ such that $G - S$ is acyclic. We approach this problem by first exhaustively applying a set of reduction rules. In order to find a minimum DFVS on the remaining instance, we create and solve a series of Vertex Cover instances.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases directed feedback vertex set, vertex cover, reduction rules

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.28

Supplementary Material

Software (Source Code): <https://doi.org/10.5281/zenodo.6645235>

Software (Source Code): <https://github.com/BenBals/mount-doom/tree/exact>, archived at `swb:1:dir:a8ce8a824241821bdff98f5380594c74d2d6c327`

1 Preliminaries

Let $G = (V, E)$ be a directed graph. The Directed Feedback Vertex Set problem asks us to find a minimum $S \subseteq V$, such that $G - S$ is acyclic.

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2022. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.



Let $u, v \in V$. We define $N^+(v)$ as the outgoing neighbors of v and $N^-(v)$ as the incoming neighbors. We call an edge $uv \in E$ bidirectional if $vu \in E$ as well. Let $PIE \subseteq E$ be the set of all bidirectional edges and let $B \subseteq V$ be the set of all vertices only incident to bidirectional edges. We define the bidirectional neighbors $N(v)$ as those which are incident using bidirectional edges. We call $D \subseteq V$ a diclique, if all $u \in D$ have $D \setminus \{u\} \subseteq N(u)$.

Finally, let $v \in V$ be given. Let $V' = V \setminus \{v\}$ and $E' = (E \cap (V' \times V')) \cup (N^-(v) \times N^+(v))$. We call $G' = (V', E')$ the graph obtained from G by shortcutting v . In light of DFVS, this is equivalent to adding the assumption $v \notin S$.

2 Reduction rules

We apply a number of reductions known from the literature and introduce one rule which to the best of our knowledge is new. The known rules can be found in [5] and we adopt their nomenclature. Additionally, we adapt some reduction rules, that have already been used for the Vertex Cover problem. Before running the reduction rules, all nodes with self loops are collected into the solution and removed from the graph. Isolated nodes are removed as well.

PIE. Consider any edge uv between different strongly connected components in $G - PIE$. Any cycle using this edge must therefore use at least one bidirectional edge, which must be covered itself, so we can safely delete uv .

DOME. We call an edge $uv \in E - PIE$ dominated if all outgoing neighbors of v are also outgoing neighbors of u or if all incoming neighbors of u are also incoming neighbors of v . It is well known [2] that such a dominated edge can safely be deleted.

Improved CORE. A vertex v is a core of a diclique if the graph induced by v and its neighbors is a diclique. Traditionally, one now deletes $N(v)$ from G since if S' is optimal for $G - N(v)$ then $S' \cup N(v)$ is optimal for G [5]. We proceed differently and shortcut the node v if $N^+(v)$ or $N^-(v)$ are dicliques. While this extension is easy to prove, it is, to the best of our knowledge, novel.

SHORTONE. Let v be a node with exactly one incoming edge uv and one outgoing edge vw in $G - PIE$ such that any bidirectional neighbor of v is also a bidirectional neighbor of at least one out of u or w . Then, remove the edges uv and vw and add uw . Call the reduced graph G' . For correctness, take any solution S in G . If $v \notin S$, then S is also a solution for G' . If $v \in S$, assume S is not a solution in G' . Then $u, w \notin S$ since any cycle introduced by the reduction must use uw . Since all bidirectional neighbors of v are also bidirectional neighbors of u or w , those must all be in S . Thus we can simply replace v by u (or w) and obtain a solution of the same size. Also, any solution in G' is always a solution in G .

2.1 Vertex Cover Reductions

Note that if we have a bidirectional edge between u and v , we have to take at least one of u and v into the DFVS. Therefore we can regard $G[PIE]$ as a Vertex Cover subproblem. Any DFVS for G must necessarily be a vertex cover for that subgraph. For that reason some Vertex Cover reduction rules [3] translate well to DFVS.

VC-DOME. Recall, that we denote by $B \subseteq V$ the set of vertices only incident to bidirectional edges. Consider $v \in B$, with $u \in N(v)$ with $N(v) \setminus \{u\} \subseteq N(u)$. Then, we add u to the solution and consider $G - u$.

Degree 2 Fold. Consider $v \in B$ with $N(v) = \{u, w\}$ and $u \in B$. Observe that there is an optimal DFVS D^* with either $D^* \cap \{u, v, w\} = \{v\}$ or $D^* \cap \{u, v, w\} = \{v, w\}$. To reduce the graph, we add a new vertex t to G and connect t in such a way, that $N^+(t) = N^+(w) \cup N(u)$ and $N^-(t) = N^-(w) \cup N(u)$ and we remove u, v and w from the graph. Then we solve the instance on the reduced graph to obtain the solution S . If $t \in S$, the solution to the original instance is $(S \setminus \{t\}) \cup \{u, w\}$. Otherwise, the solution to the original instance is $S \cup \{v\}$.

Funnel Fold. Consider a vertex $v \in B$ with $w \in N(v)$ such that $N(v) \setminus w$ is a clique. Observe that there is an optimal DFVS D^* with either $D^* \cap \{v, w\} = \{v\}$ or $D^* \cap \{v, w\} = \{w\}$. To reduce the instance, we first add $C = N(u) \cap N(w)$ to the solution and remove these vertices from the graph. Now we add edges, such that each $x \in N(v) \setminus C$ is a bidirectional neighbor of every $y \in N(w) \setminus C$. Finally, we remove u, w from the graph. Then we solve the instance on the reduced graph to obtain the solution S . If $N(v) \setminus \{w\} \subseteq S$, the solution to the original instance is $S \cup \{w\}$. Otherwise, the solution to the original instance is $S \cup \{v\}$.

3 Reduction to Vertex Cover

After applying a set of reduction rules exhaustively we solve the remaining instance by repeatedly solving Vertex Cover instances. Initially, we choose the reduction rules Improved CORE and PIE. If this does not solve the instance within five seconds, we proceed by using all reduction rules listed above.

First note that if the remaining graph contains only bidirectional edges, we can easily reduce DFVS to Vertex Cover by turning bidirectional edges into undirected edges. Initially, we find an optimal vertex cover S in $G[PIE]$. If S is a DFVS, S must be optimal. Otherwise, we find a set of vertex-disjoint cycles C in $G - S - PIE$ using a DFS. All cycles in C are not covered by S , so we add a gadget to each cycle to ensure, that in the modified graph, there is an optimal vertex cover, which includes a $v \in S$. Finally, we iterate on the modified graph until the vertex cover is also a DFVS. Note, that this can happen multiple times since our choice of C does not guarantee that all cycles in G are covered.

Let $G = (V, E)$ be an undirected graph and let $S \subseteq V$. Our goal is to find the minimum vertex cover in G that also contains a vertex in S . To achieve this, we add a clique of size $|S|$ to G and connect it one-to-one with S . We call the modified graph G' . Consider any optimal vertex cover C in G' . Then, C contains at least $|S| - 1$ vertices in the new clique. Also, C must cover all edges between V and the clique, so it must contain at least one vertex in S or all vertices in the clique. If C contains all vertices in the clique, we exchange one of these vertices for a vertex in S and obtain an optimal vertex cover C' in G' with $C' \cap S \neq \emptyset$. Thus, $C' \cap V$ is an optimal vertex cover of G that also contains a vertex of S .

Note that the exactness of our solver only relies on the optimality of the final vertex cover. Therefore, when the exact solver takes more than five seconds, we switch to a heuristic solver and verify our solution with an exact solver once we have covered all cycles in G . We do this by solving the final Vertex Cover instance completely using the exact solver. If the heuristic vertex cover was not optimal, we iterate further. Surprisingly, this did not occur on any public instance.

To solve a Vertex Cover instance, we initially reduce it using the kernelization procedure, implemented by the winning solver of the 2019 PACE challenge [4]. When solving this instance heuristically, we use a local-search solver [1] on this kernel. To solve the kernel exactly, we use a branch-and-reduce solver [6], which we augment by implementing better upper bounds using the aforementioned local-search solver.

References

- 1 Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. NuMVC: An Efficient Local Search Algorithm for Minimum Vertex Cover. *Journal of Artificial Intelligence Research*, 46:687–716, 2013.
- 2 Reinhard Diestel. Graph Theory 3rd ed. *Graduate Texts in Mathematics*, 173, 2005.
- 3 Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A Measure & Conquer Approach for the Analysis of Exact Algorithms. *Journal of the ACM*, 56(5):25:1–25:32, 2009.
- 4 Demian Hesse, Sebastian Lamm, Christian Schulz, and Darren Strash. WeGotYouCovered: The Winning Solver from the PACE 2019 Challenge, Vertex Cover Track. In *Proceedings of the SIAM Workshop on Combinatorial Scientific Computing (CSC)*, pages 1–11, 2020.
- 5 Mile Lemaic. *Markov-Chain-Based Heuristics for the Feedback Vertex Set Problem for Digraphs*. PhD thesis, Universität zu Köln, 2008.
- 6 Rick Plachetta and Alexander van der Grinten. SAT-and-Reduce for Vertex Cover: Accelerating Branch-and-Reduce by SAT Solving. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 169–180, 2021.

PACE Solver Description: Hust-Solver – A Heuristic Algorithm of Directed Feedback Vertex Set Problem*

YuMing Du ✉

School of Computer Science and Technology, Huazhong University of Science & Technology, China

QingYun Zhang ✉

School of Computer Science and Technology, Huazhong University of Science & Technology, China

JunZhou Xu ✉

Huawei TCS Lab Shanghai, China

ShunGen Zhang ✉

School of Computer Science and Technology, Huazhong University of Science & Technology, China

Chao Liao ✉

Huawei TCS Lab Shanghai, China

ZhiHuai Chen ✉

Huawei TCS Lab Shanghai, China

ZhiBo Sun ✉

School of Computer Science and Technology, Huazhong University of Science & Technology, China

ZhouXing Su ✉

School of Computer Science and Technology, Huazhong University of Science & Technology, China

JunWen Ding ✉

School of Computer Science and Technology, Huazhong University of Science & Technology, China

Chen Wu ✉

Huawei TCS Lab Shanghai, China

PinYan Lu ✉

Huawei TCS Lab Shanghai, China

ZhiPeng Lv ✉

School of Computer Science and Technology, Huazhong University of Science & Technology, China

Abstract

A directed graph is formed by vertices and arcs from one vertex to another. The feedback vertex set problem (FVSP) consists in making a given directed graph acyclic by removing as few vertices as possible. In this write-up, we outline the core techniques used in the heuristic feedback vertex set algorithm, submitted to the heuristic track of the 2022 PACE challenge.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases directed feedback vertex set, local search, simulated annealing, set covering

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.29

Supplementary Material *Software (Source Code)*: <https://doi.org/10.5281/zenodo.6644409>

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2022. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.



© YuMing Du, QingYun Zhang, JunZhou Xu, ShunGen Zhang, Chao Liao, ZhiHuai Chen, ZhiBo Sun, ZhouXing Su, JunWen Ding, Chen Wu, PinYan Lu, and ZhiPeng Lv;
licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 29; pp. 29:1–29:3

Leibniz International Proceedings in Informatics



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

1 Problem Description

Let $G = (V, E)$ be a directed graph, where V is the vertex set and $E \subseteq V \times V$ is the edge set. The feedback vertex set problem is to find a minimum subset $X \subseteq V$ such that, when all vertices of X and their adjacent edges are deleted from G , the remainder subset $G' \subseteq G$ is acyclic. The subset X is a feedback vertex set of the graph G . In the challenge of PACE 2022, for heuristic algorithms, the process was limited in 10 minutes.

2 Reduction Rules

Reduction Rules play an important role in solving FVSP because it improves the efficiency of algorithm. A contraction (reduction) operation reduces the original graph while it preserves the information necessary for finding the minimum feedback set. We adopt eight reduction rules to eliminate some vertices and edges, and deduce that some vertices must be included in the optimal solution. Five contraction operations have been proposed in Levy and Low (1988) [4]. More recently, three new contraction operations have been presented in Lin and Jou (1999) [5]. Our implementation adopted the directed graph reduction method in [5]. The reductions simplifies the graph efficiently and does not take too long.

3 Simulated Annealing

In order to tackle FVSP, we propose a simulated annealing algorithm which is based on the local search framework. The earliest idea of simulated annealing algorithm (SA) was proposed by N. Metropolis et al. [1] in 1953. It generates an initial solution, than iteratively improves the incumbent solution by a local search procedure. At each iteration of the algorithm, it first evaluates the neighborhood of the current solution and determines whether to perform the current neighborhood move by ΔT . If the current solution improves the best solution found so far, then the best solution is updated with the current. Finally, when the specified termination condition is met, the algorithm terminates and returns the best solution.

3.1 Initialization

This procedure is divided into two stages to generate an initial solution. In the first stage, the original problem is transformed into a vertex cover problem [2, 6] and then solves the problem by executing a heuristic algorithm under limited time. The execution time of the heuristic algorithm is determined according to the ratio r_b of bidirectional edges. The greater the value of r_b , the greater the execution time.

$$r_b = \frac{2 * \sum_{i,j \in V} (e_{ij} * e_{ji})}{|E|} \quad (1)$$

In the second stage, the descent heuristic algorithm is used to further optimize the solution. Finally, the initial solution X_0 of the problem is obtained.

3.2 Neighborhood Structure and Evaluation

A directed graph with no directed cycles is named a directed acyclic graph (DAG). Every DAG has a topological ordering, i.e. an ordering of its vertices such that the starting-point of every arc occurs earlier in the ordering than the endpoint of the arc. Conversely, the existence of a topological ordering in a graph proves that this graph is acyclic.

In order to solve FVSP, we adopt a neighborhood move based on add and remove operations [3]. Specifically, an add operation consists of inserting a new vertex $j \notin X$ at some particular position into the sequence and, at the same time, a remove operation used to remove the vertices that would now violate the precedence constraint. Based on the incumbent solution X , performing the operations produces a new neighborhood solution X' .

To obtain the best neighboring solution and improve the incumbent solution X , our solver uses the insert policies that in-coming and out-coming neighbors. Specifically, a vertex v can be inserted in the sequence at only two different positions, which are after its numbered in-coming neighbors, or just before its numbered out-going neighbors.

However, the simulated annealing algorithm used in [3] tends to end prematurely, so after reaching the local optimum, we increase the temperature of simulated annealing appropriately, and randomly delete some vertices in the topology sequence in order to jump out of the local optimum, so that the algorithm can rerun. The algorithm terminates until the limit time is reached. In addition, in the later period of the simulated annealing run, a cache acceleration method is also applied.

In particular, we lazily transform the FVSP to the set covering problem when the number of cycles is not too large, and use the set covering algorithm to further optimize current solution X .

3.3 Data structure

To efficiently update topological sequences, we used a data structure that can perform insert, delete and compare the order of two vertices in constant amortized time.

In our implementation, a method of labeling vertices is used. Each vertex in the sequence has a label that determines their order in the topological sequence. The labels are sparse enough, which is achieved by continuously dividing sufficiently large intervals, so that there will be spare labels between every two adjacent vertices in the topological sequence. When a vertex is inserted, it is assigned a label that is half the sum of the preceding and succeeding labels in topological order. If the spare label does not exist, the label of related vertices in the topology sequence needs to be adjusted.

References

- 1 Dimitris Bertsimas and John Tsitsiklis. Simulated annealing. *Statistical science*, 8(1):10–15, 1993.
- 2 Yuning Chen and Jin-Kao Hao. Dynamic thresholding search for minimum vertex cover in massive sparse graphs. *Eng. Appl. Artif. Intell.*, 82:76–84, 2019.
- 3 Philippe Galinier, Eunice Adjarath Lemamou, and Mohamed Wassim Bouzidi. Applying local search to the feedback vertex set problem. *J. Heuristics*, 19(5):797–818, 2013.
- 4 Hanoch Levy and David W. Low. A contraction algorithm for finding small cycle cutsets. *J. Algorithms*, 9(4):470–493, 1988.
- 5 Hen-Ming Lin and Jing-Yang Jou. Computing minimum feedback vertex sets by contraction operations and its applications on CAD. In *Proceedings of the IEEE International Conference On Computer Design, VLSI in Computers and Processors, ICCD '99, Austin, Texas, USA, October 10-13, 1999*, page 364. IEEE Computer Society, 1999.
- 6 Changsheng Quan and Ping Guo. A local search method based on edge age strategy for minimum vertex cover problem in massive graphs. *Expert Syst. Appl.*, 182:115185, 2021.

PACE Solver Description: GraPA-JAVA*

Moritz Bergenthal  

Universität Bremen, Germany

Thorben Freese 

Universität Bremen, Germany

Enna Gerhard  

Universität Bremen, Germany

Sebastian Siebertz  

Universität Bremen, Germany

Jona Dirks 

Universität Bremen, Germany

Jakob Gahde 

Universität Bremen, Germany

Mario Grobler  

Universität Bremen, Germany

Abstract

We present an exact solver for the Directed Feedback Vertex Set Problem (DFVS), submitted for the exact track of the Parameterized Algorithms and Computational Experiments challenge (PACE) in 2022. The solver heavily relies on data reduction (known from the literature and new reduction rules). The instances are then further processed by integer linear programming approaches. We implemented the algorithm in the scope of a student project at the University of Bremen.

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

Keywords and phrases complexity theory, parameterized complexity, linear programming, java, directed feedback vertex set, PACE 2022

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.30

Supplementary Material

Software (Source Code Release): <https://doi.org/10.5281/zenodo.6647003>

Software (Public Git Repository): <https://gitlab.informatik.uni-bremen.de/grapa/java/fptg-library>, archived at `swh:1:dir:fd00e212eda2d4eab270dad83923224551db839`

Software (Public Git Repository): <https://gitlab.informatik.uni-bremen.de/grapa/java/max-cliqueenumeration>, archived at `swh:1:dir:a4cde68f2e0de0e5b873eb5a02dc975c4a37fbb1`

Software (Public Git Repository): <https://gitlab.informatik.uni-bremen.de/grapa/java/pace-2022-dfvs-solver>, archived at `swh:1:dir:a80285858dd46e917d4f0c89fd13948177c6a048`

Preliminaries

We use standard notation for directed graphs as in [2], with $u, v \in V(G)$ being two vertices, and $(u, v) \in E(G)$ being an edge from vertex u to vertex v . We use the term *directed edge* for any edge $(u, v) \in E(G)$ if $(v, u) \notin E(G)$ and *undirected edge* (or 2-cycle) otherwise.

1 Solver overview

On a high level, our algorithm works as follows. It uses three steps to find a minimum DFVS for an input graph G :

1. Parse the input and apply the most basic data reduction rules.
2. Apply advanced data reduction rules iteratively (details are presented in Section 2).
3. Solve the remaining instance, possibly iteratively (details are presented in Section 3).

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2022. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.



■ **Table 1** Data reduction rules.

Name	Note	Source
1. Delete Loops	Implicitly applied	Folklore, rule 1 of [6]
2. Remove connected to $n - 1$		Special case of rule 5
3. In-/out-degree 0	Included in CRR	Rule 3 of [6], rule 5 of [1]
4. In-/out-degree 1	Included in CRR	Rule 4 of [6], rule 6 of [1]
5. Dominating 2-cycle	Superset of both predecessors and successors	Adapted from rule 6 of [4]
6. Strongly connected components	Previous rules inbetween	One step of [5] at a time
7. Delete unnecessary edges	Partially included in CRR	Special case of rule 10
8. Contract isolated paths of length 3 (lines)	Possible when not creating new induced cycles	Adapted from rule 7 of [4]
9. Contract degree three	Possible when not creating new induced cycles	Adapted from special case of rule 8 of [4]
10. Delete non-induced edges		New
We only need to consider induced edges. For every edge, we start a BFS and do not visit a vertex if a cycle is closed. For this, we track disallowed predecessors that get reduced if an alternative path exists. It is not exhaustive, as there might be cases where two alternative paths would be closed.		
11. Pick dominating vertices on 2-cycles		New
If a vertex dominates all predecessors/successors of the other vertex on an undirected edge with undirected edges, we can include it in the solution.		
12. Crown reduction	Special case of connected to all in foot	Adapted from rule 3 of [4]
13. Three hitting set	Only applies in Section 3.3	New
If all cycles running through a vertex are at most 3-cycles, we can replace them with a hitting set gadget. If all vertices of the internal clique are selected, we are allowed to push one outwards, hitting the cycle		
14. Any hitting set	Only applies in Section 3.3	New
Applied to a cycle that is isolated except for at most one edge, in that case create a gadget as in rule 13		

2 Data reduction rules

We apply the data reduction rules presented in Table 1. The rules are applied iteratively. Each rule is applied exhaustively. After a successful rule application, we return to the first rule.

We apply commonly known rules for DFVS (Rules 1, 3, 4, 6). The most local rules are applied recursively in a rule that we call Combined Recursive Reduction (CRR). As DFVS is a generalization of the Vertex Cover problem, we were able to adapt several rules designed for Vertex Cover (Rules 2, 5, 8, 9, 12). Additionally, we have found several rules generalizing known rules from the literature, as well as several completely new rules not known from the literature. Many of these rules rely on the observation that we only need to hit induced cycles when solving a DFVS instance, as all other cycles will be hit in that case as well. The correctness of all new rules will be presented in a companion paper.

3 Exact solving

After applying the data reduction rules, depending on the relative number of undirected edges, we employ different solving strategies. If less than half of the edges are undirected edges, we immediately resort to the iterative addition of cycles as a constraint for a hitting set ILP formulation, explained in Section 3.1.

In mixed or largely undirected graphs, we resort to an ILP formulation that models the problem as finding a topological order (Section 3.2) with additional hints to improve the internal lower bounds of the solver. ILPs are solved with SCIP. If the ILP solver does not terminate within twelve minutes, the attempt will be terminated. Instead, we compute an exact solution to the Vertex Cover problem. If this is not a solution to DFVS, we do not return a solution. Details are presented in Section 3.3.

3.1 Iterative cycle hitting set ILP

Over time, we generate a set \mathcal{K} of induced cycles. We initialize \mathcal{K} with all induced cycles of length 2, 3 and 4, and add longer disjoint cycles that are packed greedily until no further disjoint cycles remain. We then solve the ILP (Algorithm 1) and interpret the chosen variables as vertices to remove from the graph. If no cycle remains, we have obtained the optimal solution, otherwise, we greedily compute a new cycle packing on the remainder of the graph and add all these cycles to \mathcal{K} .

■ **Algorithm 1** Hitting set ILP formulation.

```

foreach  $v_i \in V(G)$  add variable  $x(v_i) = x_i \in X$  with constraint  $x_i \in [0, 1]$ 
foreach cycle  $K \in \mathcal{K}$  add constraint  $\sum_{v_i \in K} x(v_i) \geq 1$ 
minimize  $\sum_{x_i \in X} x_i$ 

```

3.2 Topological order ILP

A graph is acyclic if and only if its vertices can be ordered as v_1, \dots, v_n such that every edge (v_i, v_j) satisfies $i < j$. We use this observation for the following ILP formulation (Algorithm 2). For the undirected subcomponents of the graph, we compute a set of maximum cliques \mathcal{C} using [3]. This external solver does not always find all 2-cliques, so we compute all 2-cycles and remove them if they are somewhere included in a clique. We create a partial order on all vertices adjacent to directed edges, denoted as \mathcal{E} . The constraints can be interpreted as $o(s) < o(t) \vee x(s) \vee x(t)$: edges must be part of a DAG over the order or either vertex chosen for the solution. We furthermore compute a hint of short cycles \mathcal{K} as in Section 3.1. The vertices chosen can directly be interpreted as a result for DFVS.

3.3 Underlying Vertex Cover

We take the undirected part of the graph and compute a minimum vertex cover using [7]. If no cycles are remaining, we return the solution. Otherwise, we apply the hitting set gadget reduction rules (Rules 13 and 14) and solve the resulting instance again. At this point, we could turn towards iterative solving as in Section 3.1 but did not implement this as the above approach was already sufficient on all of the test instances.

■ **Algorithm 2** Linear ordered ILP formulation.

```

foreach  $v_i \in V(G)$  add
  | variable  $x(v_i) = x_i \in X$  with constraint  $x_i \in [0, 1]$ 
  | variable  $o(v_i) = y_i \in Y$  with constraint  $y_i \in [0, n]$ 
foreach clique  $C \in \mathcal{C}$  add constraint  $\sum_{v_i \in C} x(v_i) \geq |C| - 1$ 
foreach edge  $(s, t) \in \mathcal{E}$  add constraint  $o(t) - o(s) + x(t) \cdot (n + 1) + x(s) \cdot (n + 1) \geq 1$ 
foreach cycle  $K \in \mathcal{K}$  add constraint  $\sum_{v_i \in K} x(v_i) \geq 1$ 
minimize  $\sum_{x_i \in X} x_i$ 

```

References

- 1 Benjamin Bergougnoux, Eduard Eiben, Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Towards a Polynomial Kernel for Directed Feedback Vertex Set. In Kim G. Larsen, Hans L. Bodlaender, and Jean-Francois Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.MFCS.2017.36.
- 2 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2016. doi:10.1007/978-3-319-21275-3.
- 3 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM J. Exp. Algorithmics*, 18, November 2013. doi:10.1145/2543629.
- 4 Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. What is known about vertex cover kernelization? In *Adventures Between Lower Bounds and Higher Altitudes*, pages 330–356, 2018. doi:10.1007/978-3-319-98355-4_19.
- 5 Lisa K. Fleischer, Bruce Hendrickson, and Ali Pinar. On identifying strongly connected components in parallel. In José Rolim, editor, *Parallel and Distributed Processing*, pages 505–511, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. doi:10.1007/3-540-45591-4_68.
- 6 Rudolf Fleischer, Xi Wu, and Liwei Yuan. Experimental study of fpt algorithms for the directed feedback vertex set problem. In Amos Fiat and Peter Sanders, editors, *Algorithms – ESA 2009*, pages 611–622, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. doi:10.1007/978-3-642-04128-0_55.
- 7 Demian Hesse, Sebastian Lamm, Christian Schulz, and Darren Strash. WeGotYouCovered: The winning solver from the pace 2019 implementation challenge, vertex cover track. *ArXiv*, abs/1908.06795, 2019. arXiv:1908.06795.

PACE Solver Description: DreyFVS*

Gabriel Bathie ✉

École Normale Supérieure de Lyon, France

Gaétan Berthe ✉

École Normale Supérieure de Lyon, France

Yoann Coudert–Osmont ✉

Université de Lorraine, CNRS, Inria, LORIA, Nancy, France

David Desobry ✉

Université de Lorraine, CNRS, Inria, LORIA, Nancy, France

Amadeus Reinald ✉ 

École Normale Supérieure de Lyon, France

Mathis Rocton ✉ 

École Normale Supérieure de Lyon, France

Abstract

We describe DreyFVS, a heuristic for DIRECTED FEEDBACK VERTEX SET submitted to the 2022 edition of Parameterized Algorithms and Computational Experiments Challenge. The Directed Feedback Vertex Set problem asks to remove a minimal number of vertices from a digraph such that the resulting digraph is acyclic. Our algorithm first performs a guess on a reduced instance by leveraging the Sinkhorn-Knopp algorithm, to then improve this solution by pipelining two local search methods.

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

Keywords and phrases Directed Feedback Vertex Set, Heuristic, Sinkhorn algorithm, Local search

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.31

Supplementary Material *Software (Source Code):* <https://doi.org/10.5281/zenodo.6638217>

1 Introduction

In the following, a directed graph, or digraph $G = (V, E)$ is a set of vertices V , and a set of arcs E , consisting of ordered pairs of vertices. For a vertex $v \in V(G)$, we denote its out-neighbourhood by $N^+(v) = \{u \in V(G) : (v, u) \in E\}$, and its in-neighbourhood by $N^-(v) = \{u \in V(G) : (u, v) \in E\}$. A directed cycle is a sequence (u_1, \dots, u_k, u_1) such that $(u_i, u_{i+1[k]}) \in E(G)$. A directed acyclic graph, DAG for short, is a digraph containing no cycles. Given X a set of vertices, we let $G[X]$ be the graph induced by X . A strongly connected component of G is a vertex set X such that there is a path between any pair of vertices in $G[X]$. We say that (u, v) is a *digon* if it is a (directed) cycle of length two. We say that a set $C \subseteq V$ is a d -clique whenever $|C| = d$ and each pair $(c, c') \in C$ forms a digon. The DIRECTED FEEDBACK VERTEX SET problem takes as input $G = (V, E)$ and asks for a set $X \subseteq V$ such that $G[V \setminus X]$ contains no directed cycle. We consider the minimization version of the problem, asking for X to be of minimal size.

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2022. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.



© Gabriel Bathie, Gaétan Berthe, Yoann Coudert–Osmont, David Desobry, Amadeus Reinald, and Mathis Rocton;

licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 31; pp. 31:1–31:4



Leibniz International Proceedings in Informatics

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

Our algorithm first applies reduction rules to obtain a smaller instance, guesses a first solution using a Sinkhorn-Knopp algorithm, then uses two heuristics to improve the solution in a local manner. A C++ implementation is available on a public git repository [1].

In Section 2, we present the steps needed to guess our first solution. We first describe our reduction rules 2.1, based on those described by Lin and Jou [4], and provide an extension to their CORE rule. Then, we give an overview of the Sinkhorn-Knopp method in 2.2. In Section 3, we describe our two local search heuristics.

2 Guessing a solution

2.1 Reductions

At any step of the reduction, one can deal with each strongly connected component independently, as solutions to the instance are exactly unions of solutions to each strongly connected component. The main reductions rules applied in our algorithm are those developed in [4], namely LOOP, IN0, IN1, OUT0, OUT1, PIE, CORE and DOME. Given a vertex $v \in V(G)$, a bypass of v consists in the deletion of v , followed by the addition of arcs (u, w) for any $u \in N^-(v), w \in N^+(v)$. The goal of the bypass operation is to reduce instances where v can always be replaced by another vertex in the solution. Towards motivating our extension of the CORE rule, we first describe rule IN1, note that the OUT1 rule is symmetric when v admits a single out-neighbour.

► **Reduction Rule 1 (IN1).** *If a vertex $v \in V(G)$ admits a single in-neighbour, bypass v .*

Indeed, the in-neighbour of v is part of any cycle containing v . A solution using v in the original graph yields a solution of the same size using its in-neighbour in the reduced graph. Thus, a solution in the reduced graph is also a solution in the original one.

In [4], the CORE rule looks to bypass a vertex v that is a CORE of some d -clique, meaning that it forms a d -clique with all its (in or out) neighbours. We extend this rule by applying it whenever its in-neighbours or its out-neighbours form a d -clique.

► **Reduction Rule 2 (CORE').** *If a vertex $v \in V(G)$ is such that $N^-(v)$ or $N^+(v)$ forms a d -clique, bypass v .*

Indeed, considering the case where $N^-(v)$ forms a d -clique, any solution to DFVS must contain at least one vertex per digon in $N^-(v)$, thus at least $d - 1$ vertices of $N^-(v)$ in total. Then, by the same arguments as rule IN1, taking the remaining in-neighbour of v in a solution is at least as good as taking v . The case is symmetrical when $N^+(v)$ forms a d -clique, using the OUT1 rule.

2.2 The Sinkhorn–Knopp Algorithm

Given a graph G , we construct our first solution X by the heuristic of Shook and Beichl [5]. We call a set of vertex disjoint cycles a disjoint cycle union, or DCU for short. Given an $n \times n$ matrix A , the permanent of A is defined as $\text{perm}(A) = \sum_{\sigma \in \mathfrak{S}_n} \prod_{i=1}^n a_{i\sigma(i)}$. The idea of the heuristic is to iteratively add vertices v that are contained in many DCUs to the solution X . While computing all DCUs is not efficient, a first observation is that the permanent of A allows us to count the number of spanning DCUs in G , where a spanning DCU is a DCU covering all vertices of the graph. To lift the spanning constraint and count all DCUs, it is possible to take the permanent of the matrix $M = A + I$. If we denote by M_{ij} the matrix M with row i and column j removed, we define the m -balance matrix $\text{m-bal}(M)$ as in [2]:

$$\text{m-bal}(M)_{ij} = \frac{m_{ij} \cdot \text{perm}(M_{ij})}{\text{perm}(M)},$$

Then, $\text{m-bal}(M)_{vv}$ is the fraction of DCUs that do not contain v . Thus, small values lead us to consider the addition of v to the solution. Computing $\text{m-bal}(M)$ is still hard, nevertheless [2] provides a way to approximate $\text{m-bal}(M)$. This is done by computing the Sinkhorn balance matrix $\text{s-bal}(M)$ through the Sinkhorn-Knopp algorithm [6] applied on M . Sinkhorn-Knopp is an iterative algorithm where each iteration has a $\mathcal{O}(|E|)$ complexity. We decided to apply $\log |V|$ iterations leading to a $\mathcal{O}(|E| \log |V|)$ complexity for the computation of $\text{s-bal}(M)$. We can now describe our iterative construction of solution X . As long as graph G is not empty, we compute $\text{s-bal}(M)$ and add the vertex v minimising $\text{s-bal}(M)_{vv}$ to X , then, we apply our reduction rules to the remaining graph $G \setminus \{v\}$ and re-iterate.

3 Local search

The solution obtained in the previous section is further improved using two greedy local search heuristics sequentially. In the following, G refers to a strongly connected component of the reduced instance, and X to our current solution for DFVS.

3.1 Vertex Swapping

The first local operation consists of a simple vertex swap. Namely, we remove a vertex from the current solution X , and if a cycle is created we attempt to replace it with another vertex of the cycle at random. We iterate through every vertex v of X . If $G \setminus X \cup \{v\}$ is acyclic, we remove v from X such that the solution size decreases by 1. Otherwise, we consider a cycle C in $G \setminus X \cup \{v\}$, and choose some other vertex $w \in C \setminus \{v\}$, replacing v with w in X if $G \setminus X \cup \{v\} \setminus \{w\}$ is acyclic.

3.2 Vertex Ordering Perturbation

Recall that a digraph is acyclic if and only if its set of vertices admits a topological ordering, that is, an order π over V such that for every arc (u, v) , $\pi(u) < \pi(v)$. Our idea, similar to that of [3], is to define a unique DFVS X_π from any ordering π of V , and then swap the positions of random pairs of vertices in π , aiming to reduce the size of the solution X_π .

We first show how to construct X_π from π . Start with $X_\pi^0 = \emptyset$, and then define X_π^n inductively as follows:

$$X_\pi^{n+1} = \begin{cases} X_\pi^n & \text{if } N^+(v_\pi^{n+1}) \cap V_\pi^n \subseteq X_\pi^n \\ X_\pi^n \cup \{v_\pi^{n+1}\} & \text{otherwise} \end{cases}$$

Where $V_\pi^n = \{v \in V \mid \pi(v) \leq n\}$ and v_π^n is the vertex such that $\pi(v_\pi^n) = n$. We then set $X_\pi = X_\pi^{|V|}$. Notice that X_π can be computed in linear time using a traversal of G in the order given by π .

We can use this structure to derive a simple local search algorithm, which repeatedly chooses a uniformly random pair of vertices (u, v) and swaps their positions to obtain a new ordering π' such that $\pi'(u) = \pi(v)$, $\pi'(v) = \pi(u)$ and $\pi'(w) = \pi(w)$ for $w \notin \{u, v\}$. We then compute $|X_{\pi'}|$, and if $|X_{\pi'}| \leq |X_\pi|$, we validate the swap and set π to π' , otherwise we cancel the swap and leave π unchanged. In practice, iterating this process quickly reduces the size of the solution X_π on most instances.

Given a solution X obtained after applying the local search of the previous section 3.1, we can compute the initial value of π using a topological sort of the acyclic digraph $G[V \setminus X]$, to which we append a random permutation of X .

References

- 1 Gabriel Bathie, Gaétan Berthe, Yoann Coudert-Osmont, David Desobry, Amadeus Reinald, and Mathis Rocton. DreyFVS, June 2022. doi:10.5281/zenodo.6638217.
- 2 Isabel Beichl and Francis Sullivan. Approximating the permanent via importance sampling with application to the dimer covering problem. *Journal of Computational Physics*, 149(1):128–147, 1999. doi:10.1006/jcph.1998.6149.
- 3 Philippe Galinier, Eunice Lemamou, and Mohamed Wassim Bouzidi. Applying local search to the feedback vertex set problem. *Journal of Heuristics*, 19(5):797–818, October 2013. doi:10.1007/s10732-013-9224-z.
- 4 Hen-Ming Lin and Jing-Yang Jou. On computing the minimum feedback vertex set of a directed graph by contraction operations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(3):295–307, 2000. doi:10.1109/43.833199.
- 5 James Shook and Isabel Beichl. Matrix scaling: A new heuristic for the feedback vertex set problem, June 2014. URL: <https://math.nist.gov/mcsd/Seminars/2014/2014-06-10-Shook-presentation.pdf>.
- 6 Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

PACE Solver Description: DAGER – Cutting out Cycles with MaxSAT*

Rafael Kiesel   

TU Wien, Austria

André Schidler   

TU Wien, Austria

Abstract

We describe the solver DAGER for the Directed Feedback Vertex Set (DFVS) problem, as it was submitted to the exact track of the 2022 PACE Challenge. Our approach first applies a wide range of preprocessing techniques involving both well-known data reductions for DFVS as well as non-trivial adaptations from the vertex cover problem. For the actual solving, we found that using a MaxSAT solver with incremental constraints achieves a good performance.

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

Keywords and phrases Directed Feedback Vertex Set, Data Reductions, Incremental MaxSAT

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.32

Related Version *Preprint of Full Version*: <https://arxiv.org/abs/2211.06109>

Supplementary Material *Software (Source Code)*: <https://github.com/ASchidler/dfvs>

Software (Source Code): <https://doi.org/10.5281/zenodo.6627405>

Funding This work has been supported by the FWF W1255(-N23).

André Schidler: The author has been supported by the FWF (P32441) and the WWTF (ICT19-065).

1 Introduction

This paper describes the solver DAGER¹, which we submitted to the exact track of the 2022 PACE Challenge [9]. This year the challenge was to solve the Directed Feedback Vertex Set (DFVS) problem. Informally, the DFVS problem is, given a directed graph $G = (V, A)$ to find a minimum cardinality subset $D \subseteq V$ such that every directed cycle of G uses at least one vertex from D . The problem is one Karp’s original 21 NP-complete problems [6] and has a wide range of applications such as for argumentation frameworks [4, 3], deadlock detection, program verification and VLSI chip design [10]. It is known that the problem is fixed-parameter tractable [2] in the size of the DFVS.

The runtime of these parameterized algorithms quickly increases with increasing size of the DFVS, making alternative approaches necessary for solving instances with a large DFVS, like many of the instances in these year’s PACE Challenge. We use an alternative approach to developing a dedicated DFVS algorithm: we express the problem in terms of constraints and use a constraint solver for computing the DFVS. Our implementation uses a propositional satisfiability (SAT) solver as a constraint solver. Unfortunately, the size of a direct encoding in propositional logic is, depending on the encoding, cubic or exponential in the size of the input graph, and therefore prohibitively large. We therefore use a lazy

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2022. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.

¹ Available at <https://github.com/ASchidler/dfvs> and <https://doi.org/10.5281/zenodo.6627405>



© Rafael Kiesel and André Schidler;

licensed under Creative Commons License CC-BY 4.0

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).

Editors: Holger Dell and Jesper Nederlof; Article No. 32; pp. 32:1–32:4

Leibniz International Proceedings in Informatics



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

approach: we use an adapted SAT solver that creates the necessary constraints when they are violated, keeping the total number of constraints low enough for most of the instances in the public instance set. We thereby exploit the following observation by [1] in a similar context. Namely, while there may be exponentially many cycles in a directed graph, in order to prove that we need at least k vertices to cover every cycle, we may need far fewer.

Data reductions are another crucial component of DAGer, as they transform the given graph into a smaller one that is easier to solve. Notably, apart from the well-known standard data reductions for DFVS of [7] and [8], we also managed to non-trivially generalize many data reductions for the Vertex Cover (VC) problem to DFVS.

2 Preliminaries

We denote an undirected graph as $H = (V, E)$ with vertices V and edges E , and a directed graph, or *digraph*, by $G = (V, A)$ with arcs A . Further, we denote an undirected edge between vertices u and v as $\{u, v\}$ and the arc, or directed edge, from u to v as (u, v) .

We can now introduce the central problem addressed in this paper.

► **Definition 1** (Cycle, DFVS). *Given a digraph $G = (V, A)$ a path is a list of vertices v_1, \dots, v_n such that for $i = 1, \dots, n - 1$ there exists an arc $(v_i, v_{i+1}) \in A$. A cycle is a path v_1, \dots, v_n such that $v_1 = v_n$. Furthermore, a cycle is uncovered, if there is no cycle v'_1, \dots, v'_m such that $\{v'_i \mid i = 1, \dots, m\} \subsetneq \{v_i \mid i = 1, \dots, n\}$. A Directed Feedback Vertex Set (DFVS) of G is a set $D \subseteq V$ of minimum cardinality such that every cycle of G contains at least one vertex in D .*

We use the notation $\mathcal{C}(G)$ to refer to the set of all uncovered cycles in G . Since any self-loop (u, u) can be easily preprocessed, by adding u to the DFVS and removing u from the digraph, we will henceforth assume that the input digraph is self-loop-free.

We can represent the problem of finding a DFVS in terms of another problem.

► **Definition 2** (Hitting Set). *Given a set V called the universe and a set $\mathcal{C} = \{C_1, \dots, C_n\}$ with $C_i \subseteq V$ for $1 \leq i \leq n$, the minimum hitting set problem asks for a set $D \subseteq V$ of minimum cardinality such that $D \cap C_i \neq \emptyset$ for all $1 \leq i \leq n$.*

Given a digraph G , a hitting set for V and $\mathcal{C}(G)$ is also a DFVS for G . In the special case where each uncovered cycle has length 2, we can also express DFVS as follows:

► **Definition 3** (Vertex Cover). *Let $H = (V, E)$ be an undirected graph. A minimum vertex cover is a set of vertices $D \subseteq V$, such that D is of minimum cardinality and for each edge $\{u, v\} \in E$ either $u \in D$ or $v \in D$.*

3 Algorithm

Our solver DAGer works via reduction of the DFVS problem to the hitting set problem. Given the set of cycles $\mathcal{C}(G)$, we can easily express that each cycle needs to be *hit* by the DFVS in propositional logic. This propositional encoding has one variable per vertex and the corresponding vertex is in the DFVS if and only if the variable is true. We represent each cycle as a disjunction, as at least one vertex per cycle must be in the DFVS, and connect all cycles by a conjunction. We ensure the minimality of the DFVS by using a MaxSAT solver: besides satisfying the clauses representing the cycles, the MaxSAT solver maximizes the number of satisfied *soft clauses*. We add for each variable the negation as a soft clause, causing the MaxSAT solver to minimize the number of variables set to true.

As the graph might have too many cycles to efficiently enumerate them all, we proceed in a lazy fashion: while computing a hitting set for a subset of the cycles we dynamically check for additional cycles that do not intersect the hitting set. More precisely, DAGER works as follows:

1. Preprocess the graph (See Section 3.1)
2. Identify a set $\mathcal{C}'(G) \subseteq \mathcal{C}(G)$ of uncovered short cycles, we use a maximum length of 8.
3. If $\mathcal{C}'(G) = \mathcal{C}(G)$, perform additional preprocessing not performed in Step 1 (Section 3.1).
4. Find a minimum hitting set D for $\mathcal{C}'(G)$ using a modified MaxSAT solver (See Section 3.2).
5. If an additional cycle is found while solving, add it.

We will briefly discuss our preprocessing and our changes to the MaxSAT solver.

3.1 Preprocessing

From classic DFVS preprocessing we used the data reductions INDICLIQUE, OUTDICLIQUE, DICLIQUE-2 and DICLIQUE-3 from [7], as well as the data reductions PIE and DOME from [8]. Apart from that, we adapted preprocessing techniques from the vertex cover setting to the DFVS problem. Here, we used the data reductions 1,2,3,4,5,6 and 7.2 from [11], as well as data reductions 8 and 10.1 from [5]. Their soundness for DFVS is based on the following observation: Given a directed graph G that only has uncovered cycles of length 2, i.e., each cycle has the form u, v, u and let $G' = (V, E')$ such that $E' = \{(u, v) \mid \text{there is a cycle } u, v, u \text{ in } G\}$. A minimum vertex cover for G' is a minimum DFVS for G .

Clearly, in the above case we can apply all vertex cover data reductions on the undirected graph. Furthermore, if we know all uncovered cycles, we can apply vertex cover preprocessing locally, whenever all vertices involved in the reduction only take part in uncovered cycles of length 2. This case constitutes Step 3. of the algorithm, we described above.

However, it is not always the case that we know all cycles. Therefore, we integrated all the aforementioned vertex cover preprocessing techniques also into the preprocessing in Step 1. It follows from the above reasoning that if all vertices involved in a reduction only have bidirectional edges, then we can apply it. There are additional cases in which we can additionally apply vertex cover preprocessing, however, a detailed analysis of these cases requires a lot of technical details and is therefore left out due to space reasons. The underlying intuition is simple though. We can apply a VC reduction if (1) it would be applicable, in the undirected graph that has an edge $\{u, v\}$ whenever u and v are bidirectionally connected in the original graph and (2) the application of the reduction does not lead to new subset-minimal cycles.

3.2 MaxSAT and Cycle Check

The SAT solver used inside the MaxSAT² solver tries to find satisfying assignment (equivalent to a DFVS) by repeating the following steps, until all variable values have been set:

1. Decide on a variable and value, i.e., decide to add a vertex to the DFVS or leave it in the graph.
2. Propagate values that are implied by the decision, e.g., if all but one vertices in a cycle have been left in the graph, add the remaining vertex to the DFVS.
3. Check if any clause cannot be satisfied, i.e., a cycle where all vertices are left in the graph.
4. If yes, learn a clause to avoid making the same decisions and undo the corresponding decisions.

² We use the solver *EvalMaxSAT* <https://github.com/FlorentAvellaneda/EvalMaxSAT>

We extended the conflict check in Step 3: whenever the SAT solver decides to leave a vertex in the graph, we check if there is a cycle. If there is a cycle, we add the cycle to our encoding and notify the solver of the conflict. This way we do not have to find all the cycles initially and we only add those cycles that are required for finding a DFVS.

References

- 1 Ali Baharev, Hermann Schichl, Arnold Neumaier, and Tobias Achterberg. An exact method for the minimum feedback arc set problem. *Journal of Experimental Algorithmics (JEA)*, 26:1–28, 2021.
- 2 Jianer Chen, Yang Liu, Songjian Lu, Barry O’Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 177–186. ACM, 2008.
- 3 Wolfgang Dvorák, Markus Hecher, Matthias König, André Schidler, Stefan Szeider, and Stefan Woltran. Tractable abstract argumentation via backdoor-treewidth. In *AAAI 2022*, 2022.
- 4 Wolfgang Dvorák, Sebastian Ordyniak, and Stefan Szeider. Augmenting tractable fragments of abstract argumentation. *Artif. Intell.*, 186:157–173, 2012. doi:10.1016/j.artint.2012.03.002.
- 5 Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. What is known about vertex cover kernelization? In Hans-Joachim Böckenhauer, Dennis Komm, and Walter Unger, editors, *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, volume 11011 of *Lecture Notes in Computer Science*, pages 330–356. Springer, 2018. doi:10.1007/978-3-319-98355-4_19.
- 6 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a Symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 7 Mile Lemaic. *Markov-chain-based heuristics for the feedback vertex set problem for digraphs*. PhD thesis, Universität zu Köln, 2008.
- 8 Hen-Ming Lin and Jing-Yang Jou. On computing the minimum feedback vertex set of a directed graph by contraction operations. *IEEE TCAD*, 19(3):295–307, 2000.
- 9 Christian Schulz, Ernestine Großmann, Tobias Heuer, and Darren Strash. Pace 2022. URL: <https://pacechallenge.org/2022/>.
- 10 Abraham Silberschatz, Greg Gagne, and Peter B Galvin. *Operating System Concepts*. JW Wiley, 2018.
- 11 Ulrike Stege and Michael Ralph Fellows. An improved fixed parameter tractable algorithm for vertex cover. *Technical report/Departement Informatik, ETH Zürich*, 318, 1999.