# 12th International Symposium on Parameterized and Exact Computation

**IPEC 2017, September 6–8, 2017, Vienna, Austria**

Edited by

Daniel Lokshtanov

Naomi Nishimura

*Editors*

Daniel Lokshtanov
Department of Informatics
University of Bergen
Daniel.Lokshtanov@uib.no

Naomi Nishimura
David R. Cheriton School of Computer Science
University of Waterloo
nishi@uwaterloo.ca

*ACM Classification 1998*
F.1.3 Complexity Measures and Classes, F.2 Analysis of Algorithms and Problem Complexity, G.2 Discrete Mathematics

**ISBN 978-3-95977-051-4**

*Published online and open access by*
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-051-4.

*Publication date*
February, 2018

*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 http://dnb.d-nb.de.



Digital Object Identifier: 10.4230/LIPIcs.IPEC.2017.

## 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.



**ISSN 1868-8969**

**http://www.dagstuhl.de/lipics**

# Contents



12th International Symposium on Parameterized and Exact Computation (IPEC 2017).
Editors: Daniel Lokshtanov and Naomi Nishimura
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

# Preface

This volume contains the papers presented at IPEC 2017: the 12th International Symposium on Parameterized and Exact Computation held during September 6–8, 2017, in Vienna, Austria. IPEC was held together with five other algorithms conferences and a summer school as part of the annual ALGO congress.

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.

In response to the call for papers, 68 papers were submitted. Each submission was reviewed by at least 3 reviewers. The reviews came from the 14 members of the program committee, and from 100 external reviewers contributing 132 external reviews. The program committee held electronic meetings through the EasyChair.

The program committee felt that the median submission quality was very high, and in the end selected 29 of the submissions for presentation at the symposium and for inclusion in this proceedings volume. The Best Paper Award was presented to Radu Curticapean, Holger Dell, Fedor Fomin, Leslie Ann Goldberg and John Lapinskas for the paper *A Fixed-Parameter Perspective on #BIS* and the Excellent Student Paper Award was presented to Bart M. P. Jansen and Astrid Pieterse for the paper *Optimal Data Reduction for Graph Coloring Using Low-Degree Polynomials*.

Previous IPECs

| | |
|---|---|
| 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 |

IPEC invited one plenary speaker to the ALGO meeting, Fabrizio Grandoni, as part of the award ceremony for the 2017 EATCS-IPEC Nerode Prize for outstanding papers in the area of multvariate algorithmics. The award was given by a committee consisting of David Eppstein, Daniel Marx, and Jianer Chen to Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch for their paper *A Measure & Conquer Approach for the Analysis of Exact Algorithms* [Journal of the ACM 65 (5): Article 25, 2009]. We thank Fabrizio for accepting our invitation and for contributing an excellent talk to IPEC 2017.

IPEC also invited Mikołaj Bojańczyk to present a tutorial "On Courcelle's conjecture about recognisable graph classes."

We would like to thank the program committee, together with the external reviewers, for their commitment in the difficult paper selection process. We also thank all the authors who submitted their work for consideration. Finally, we are grateful to the local organizers of ALGO, chaired by Stefan Szeider, for their efforts, which made chairing IPEC an enjoyable experience.

Daniel Lokshtanov and Naomi Nishimura
Bergen and Waterloo, October 2017

12th International Symposium on Parameterized and Exact Computation (IPEC 2017).
Editors: Daniel Lokshtanov and Naomi Nishimura
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

# List of External Reviewers

Faisal Abu-Khzam
Isolde Adler
Akanksha Agrawal
Antonios Antoniadis
Aritra Banik
Rémy Belmonte
Matthias Bentert
Olaf Beyersdorff
Ivona Bezakova
Arindam Biswas
Andreas Björklund
Markus Bläser
Hans L. Bodlaender
Greg Bodwin
Edouard Bonnet
Robert Bredereck
Jonathan Buss
Leizhen Cai
Katrin Casel
Jiehua Chen
Yijia Chen
Fabio Cunial
Konrad Kazimierz Dabrowski
Ronald de Haan
Holger Dell
William E. Devanny
Vida Dujmovic
David Eppstein
Till Fluschnik
Vincent Froese
Kyle Fox
Robert Ganian
Serge Gaspers
Panos Giannopoulos
Petr Golovach
Samuel Haney
Danny Hermelin
Eva-Maria Hols
Ashwin Jacob
Bart M. P. Jansen
Andrzej Kaczmarczyk
Petteri Kaski
Steven Kelk
Eun Jung Kim
Johannes Köbler
Martin Koutecky
Lukasz Kowalik
Sebastian Krinninger
R. Krithika
Alan Kuhnle
Marvin Künnemann
Foram Lakhani
Michael Lampis
Hung Le
Bingkai Lin
Maarten Löffler
Diptapriyo Majumdar
David Manlove
Till Miltzow
Pranabendu Misra
Matthias Mnich
Hendrik Molter

12th International Symposium on Parameterized and Exact Computation (IPEC 2017).
Editors: Daniel Lokshtanov and Naomi Nishimura
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## External Reviewers

N.S. Narayanaswamy

Rian Neogi

André Nichterlein

Joanna Ochremiak

Sebastian Ordyniak

Fahad Panolan

Christophe Paul

Daniel Paulusma

Marcin Pilipczuk

Vinod Reddy

Fernando Sanchez Villaamil

Ignasi Sau

Saket Saurabh

Kevin Schewior

Ildi Schlotter

Pascal Schweitzer

Anil Shukla

Sebastian Siebertz

Matthew Skala

Arkadiusz Socala

Manuel Sorge

Srikanth Srinivasan

Ulrike Stege

Ondrej Suchy

Prafullkumar Tale

Jacobo Torán

Rene van Bevern

Tom van der Zanden

Erik Jan van Leeuwen

Mathias Weller

Marcin Wrochna

Meirav Zehavi

Chihao Zhang

# List of Authors




12th International Symposium on Parameterized and Exact Computation (IPEC 2017).
Editors: Daniel Lokshtanov and Naomi Nishimura
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Vincent T'Kindt

Prafullkumar Tale

Nimrod Talmon

Hisao Tamaki

Jan Arne Telle

Dimitrios M. Thilikos

Jacobo Torán

Tom C. van der Zanden

Mathias Weller

Emo Welzl

Ryan Williams

Gerhard Woeginger

Stefan Woltran

Marcin Wrochna

Anna Zych-Pawlewicz

# On the Parameterized Complexity of Contraction to Generalization of Trees*

**Akanksha Agrawal[1], Saket Saurabh[2], and Prafullkumar Tale[3]**

1 **Department of Informatics, University of Bergen, Bergen, Norway**
akanksha.agrawal@uib.no
2 **Department of Informatics, University of Bergen, Bergen, Norway and The Institute of Mathematical Sciences, HBNI, Chennai, India**
saket@imsc.re.in
3 **The Institute of Mathematical Sciences, HBNI, Chennai, India**
pptale@imsc.res.in

## Abstract

For a family of graphs $\mathcal{F}$, the $\mathcal{F}$-Contraction problem takes as an input a graph $G$ and an integer $k$, and the goal is to decide if there exists $S \subseteq E(G)$ of size at most $k$ such that $G/S$ belongs to $\mathcal{F}$. Here, $G/S$ is the graph obtained from $G$ by contracting all the edges in $S$. Heggernes et al. [*Algorithmica (2014)*] were the first to study edge contraction problems in the realm of Parameterized Complexity. They studied $\mathcal{F}$-Contraction when $\mathcal{F}$ is a simple family of graphs such as trees and paths. In this paper, we study the $\mathcal{F}$-Contraction problem, where $\mathcal{F}$ generalizes the family of trees. In particular, we define this generalization in a "parameterized way". Let $\mathbb{T}_\ell$ be the family of graphs such that each graph in $\mathbb{T}_\ell$ can be made into a tree by deleting at most $\ell$ edges. Thus, the problem we study is $\mathbb{T}_\ell$-Contraction. We design an FPT algorithm for $\mathbb{T}_\ell$-Contraction running in time $\mathcal{O}((2\sqrt{\ell}+2)^{\mathcal{O}(k+\ell)} \cdot n^{\mathcal{O}(1)})$. Furthermore, we show that the problem does not admit a polynomial kernel when parameterized by $k$. Inspired by the negative result for the kernelization, we design a lossy kernel for $\mathbb{T}_\ell$-Contraction of size $\mathcal{O}([k(k+2\ell)]^{(\lceil \frac{\alpha}{\alpha-1} \rceil + 1)})$.

**1998 ACM Subject Classification** G.2.2 Graph Algorithms, I.1.2 Analysis of Algorithms

**Keywords and phrases** Graph Contraction, Fixed Parameter Tractability, Graph Algorithms, Generalization of Trees

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.1

## 1 Introduction

Graph editing problems are one of the central problems in graph theory that have been extensively studied in the realm of Parameterized Complexity. Some of the important graph editing operations are vertex deletion, edge deletion, edge addition, and edge contraction. For a family of graphs $\mathcal{F}$, the $\mathcal{F}$-Editing problem takes as an input a graph $G$ and an integer $k$, and the goal is to decide whether or not we can obtain a graph in $\mathcal{F}$ by applying at most $k$ edit operations on $G$. In fact, the $\mathcal{F}$-Editing problem, where the edit operations are restricted to one of vertex deletion, edge deletion, edge addition, or edge contraction have also received a lot of attention in Parameterized Complexity. When we restrict the operations to only deletion operation (vertex/edge deletion) then the corresponding problem is called $\mathcal{F}$-Vertex

* A full paper containing all the proofs and explanations can be found at https://arxiv.org/abs/1708.00622

12th International Symposium on Parameterized and Exact Computation (IPEC 2017).
Editors: Daniel Lokshtanov and Naomi Nishimura; Article No. 1; pp. 1:1–1:12

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

(Edge) Deletion problem. On the other hand if we only allow edge contraction then the corresponding problem is called $\mathcal{F}$-Contraction. The $\mathcal{F}$-Editing problem generalizes several NP-hard problems such as Vertex Cover, Feedback vertex set, Planar $\mathcal{F}$-Deletion, Interval Vertex Deletion, Chordal Vertex Deletion, Odd cycle transversal, Edge Bipartization, Tree Contraction, Path Contraction, Split Contraction, Clique Contraction, etc. Most of the studies in the Parameterized Complexity or the classical Complexity Theory have been restricted to combination of vertex deletion, edge deletion or edge addition. Only recently, edge contraction as an edit operation has started to gain attention in the realm of Parameterized Complexity. In this paper, we add another family of graphs $\mathcal{F}$ – a parameterized generalization of trees – such that $\mathcal{F}$-Contraction is fixed parameter tractable (FPT). We also explore the problem from the viewpoints of Kernelization Complexity as well as its new avatar the Lossy Kernelization. For more details on Parameterized Complexity we refer to the books of Downey and Fellows [11, 12], Flum and Grohe [13], Niedermeier [21], and Cygan et al. [8].

Our starting point is the result of Heggernes et al. [17] who studied $\mathcal{F}$-Contraction when $\mathcal{F}$ is the family of paths ($\mathbb{P}$) and trees ($\mathbb{T}$). To the best of our knowledge these were the first results concerning Parameterized Complexity of $\mathcal{F}$-Contraction problems. They showed that $\mathbb{P}$-Contraction and $\mathbb{T}$-Contraction are FPT. Furthermore, they showed that $\mathbb{T}$-Contraction does not admit a polynomial kernel. On the other hand $\mathbb{P}$-Contraction admits a polynomial kernel with at most $5k + 3$ vertices (see [18] for an improved bound of $3k + 4$ on the number of vertices). Moreover, $\mathcal{F}$-Contraction is not FPT(unless some unlikely collapse in Parameterized Complexity happens) even for simple family of graphs such as $P_t$-free graphs for some $t \geq 5$, the family of $C_t$-free graphs for some $t \geq 4$ [6, 19], and the family of split graphs [2]. Here, $P_t$ and $C_t$ denotes the path and cycle on $t$ vertices. In light of these mixed answers, two natural questions are:

1. What additional parameter we can associate with $\mathbb{T}$-Contraction such that it admits a polynomial kernel?
2. What additional parameter we can associate with $\mathbb{T}$-Contraction such that an FPT algorithm with combination of these parameterizations leads to an algorithm that generalizes the FPT algorithm on trees?

In our earlier paper (a superset of authors) we addressed the first question [1]. In particular we studied $\mathcal{F}$-Contraction, where $\mathcal{F}$ is the *family of trees with at most $\ell$ leaves* (together with some other problems), and designed a polynomial kernel (hence an FPT algorithm) with $\mathcal{O}(k\ell)$ vertices. This was complimented by a matching kernel lower bound result. In this paper we focus on the second question.

**Our Problem and Results.** To define our problem formally let us define $\mathbb{T}_\ell$ to be the family of graphs such that each graph in $\mathbb{T}_\ell$ can be made into a tree by deleting at most $\ell$ edges. Thus the problem we study will be called $\mathbb{T}_\ell$-Contraction.

| $\mathbb{T}_\ell$-Contraction | **Parameter:** $k$ |
|---|---|
| **Input:** A graph $G$ and an integer $k$. | |
| **Question:** Does there exist $S \subseteq E(G)$ of size at most $k$ such that $G/S \in \mathbb{T}_\ell$? | |

Observe that for $\ell = 0$, $\mathbb{T}_\ell$-Contraction is the usual $\mathbb{T}$-Contraction. We design an FPT algorithm for $\mathbb{T}_\ell$-Contraction running in time $\mathcal{O}((2\sqrt{\ell} + 2)^{\mathcal{O}(k+\ell)} \cdot n^{\mathcal{O}(1)})$. Our algorithm follows the general approach of Heggernes et al. [17] for designing the algorithm for $\mathbb{T}$-Contraction. Also, we show that the problem does not admit a polynomial kernel, when parameterized by $k$, for any (fixed) $\ell \in \mathbb{N}$. Inspired by the negative result on kernelization, we design a lossy kernel for $\mathbb{T}_\ell$-Contraction.

**Related Works.** For several families of graphs $\mathcal{F}$, early papers by Watanabe et al. [22, 23] and Asano and Hirata [3] showed that $\mathcal{F}$-Contraction is NP-complete. From the viewpoint of Parameterized Complexity these problems exhibit properties that are quite different from the problems where the edit operations are restricted to deleting or adding vertices or edges. For instance, deleting $k$ edges from a graph such that the resulting graph is a tree is polynomial time solvable. On the other hand, Asano and Hirata showed that $\mathbb{T}$-Contraction is NP-hard [3]. Furthermore, a well-known result by Cai [5] states that when $\mathcal{F}$ is a hereditary family of graphs with a finite set of forbidden induced subgraphs then the graph modification problem defined by $\mathcal{F}$ and the edit operations restricted to vertex deletion, edge deletion, or edge addition admits an FPT algorithm. Moreover, this result does not hold when the edit operation is edge contraction. Lokshtanov et al. [19] and Cai and Guo [6] independently showed that if $\mathcal{F}$ is either the family of $P_\ell$-free graphs for some $\ell \geq 5$ or the family of $C_\ell$-free graphs for some $\ell \geq 4$ then $\mathcal{F}$-Contraction is W[2]-hard. Golovach et al. [14] proved that if $\mathcal{F}$ is the family of planar graphs then $\mathcal{F}$-Contraction is FPT. Belmonte et al. [4] proved that the problem is FPT for $\mathcal{F}$ being the family of degree constrained graphs like bounded degree, (constant) degenerate and (constant) regular graphs. Moreover, Cai and Guo [6] showed that in case $\mathcal{F}$ is the family of cliques, $\mathcal{F}$-Contraction is solvable in time $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$, while in case $\mathcal{F}$ is the family of chordal graphs, the problem is W[2]-hard. Heggernes et al. [16] developed an FPT algorithm for the case where $\mathcal{F}$ is the family of bipartite graphs (see [15] for a faster algorithm).

## 2 Preliminaries

In this section, we state some basic definitions and introduce terminologies from graph theory and algorithms. We also establish some of the notations that will be used throughout. We denote the set of natural numbers by $\mathbb{N}$. For $k \in \mathbb{N}$, by $[k]$ we denote the set $\{1, 2, \ldots, k\}$.

**Graphs.** We use standard terminologies from the book of Diestel [10] for the graph related terms which are not explicitly defined here. We consider simple graphs. For a graph $G$, by $V(G)$ and $E(G)$ we denote the vertex and edge sets of $G$, respectively. For a vertex $v \in V(G)$, we use $deg_G(v)$ to denote the degree of $v$ in $G$, *i.e.* the number of edges in $G$ that are incident to $v$. For $v \in V(G)$, by $N_G(v)$ we denote the set $\{u \in V(G) \mid vu \in E(G)\}$. We drop the subscript $G$ from $deg_G(v)$ and $N_G(v)$ whenever the context is clear. For a vertex subset $S \subseteq V(G)$, by $G[S]$ we denote the graph with the vertex set $S$ and the edge set as $\{vu \in E(G) \mid v, u \in S\}$. By $G - S$ we denote the graph $G[V(G) \setminus S]$. We say $S, S' \subseteq V(G)$ are *adjacent* if there is $v \in S$ and $v' \in S'$ such that $vv' \in E(G)$. Further, an edge $uv \in E(G)$ is *between* $S$ and $S'$ if $u \in S$ and $v \in S'$. For $E' \subseteq E(G)$, by $G/E'$ we denote the graph obtained from $G$ by contracting the edges in $E'$. For $\ell \in \mathbb{N}$, by $\mathbb{T}_\ell$ we denote the family of graphs from which we can obtain a tree using at most $\ell$ edge deletions. Observe that for any graph $G \in \mathbb{T}_\ell$, we have $|E(G)| \leq |V(G)| - 1 + \ell$. Moreover, for any connected graph $G$, if $|E(G)| \leq |V(G)| - 1 + \ell$ then $G \in \mathbb{T}_\ell$.

A graph $G$ is *contractible* to a graph $H$, if their exists $E' \subseteq E(G)$ such that $G/E'$ is *isomorphic* to $H$. In other words, $G$ is *contractible* to $H$ if there exists a *surjective* function $\varphi : V(G) \to V(H)$ with $W(h) = \{v \in V(G) \mid \varphi(v) = h\}$, for $h \in V(H)$ and the following property holds.

- For all $h, h' \in V(H)$, $hh' \in E(H)$ if and only if $W(h), W(h')$ are *adjacent* in $G$.
- For all $h \in V(H)$, $G[W(h)]$ is *connected*.

Let $\mathcal{W} = \{W(h) \mid h \in V(H)\}$. Observe that $\mathcal{W}$ defines a partition of vertices in $G$. We call $\mathcal{W}$ as an *$H$-witness structure* of $G$. The sets in $\mathcal{W}$ are called *witness sets*. If a *witness set* contains more than one vertex then we will call it a *big* witness-set, otherwise it is a *small* witness set. A graph $G$ is said to be $k$-contractible to a graph $H$ if there exists $E' \subseteq E(G)$ such that $G/E'$ is isomorphic to $H$ and $|E'| \leq k$.

For a subset $S \subseteq V(G)$ and a $k$-coloring $\phi$ of $G$, $S$ is said to be *monochromatic* with respect to $\phi$ if for all $s, s' \in S$, $\phi(s) = \phi(s')$. Observe that $\phi$ partitions $V(G)$ into (at most) $k$ pairwise disjoint sets. A subset $S \subseteq V(G)$ is said to be *monochromatic component* with respect to $\phi$ if $S$ is monochromatic and $G[S]$ is connected.

## 3 FPT Algorithm for $\mathbb{T}_\ell$-Contraction

In this section, we design an FPT algorithm for $\mathbb{T}_\ell$-Contraction. Our algorithm proceeds as follows. We start by applying some simple reduction rules. Then by branching we ensure that the resulting graph is 2-connected. Finally, we give an FPT algorithm running in time $\mathcal{O}((2\sqrt{\ell}+2)^{\mathcal{O}(k+\ell)} \cdot n^{\mathcal{O}(1)})$ on 2-connected graphs. The approach we use for designing the algorithm for the case when the input graph is 2-connected follows the approach of Heggernes et al. [17] for designing an FPT algorithm for contracting to trees. Also, whenever we are dealing with an instance of $\mathbb{T}_\ell$-Contraction we assume that we have an algorithm running in time $\mathcal{O}((2\sqrt{\ell'}+2)^{\mathcal{O}(k+\ell')} \cdot n^{\mathcal{O}(1)})$ for $\mathbb{T}_{\ell'}$-Contraction, for every $\ell' < \ell$. That is, we give family of algorithms inductively for each $\ell' \in \mathbb{N}$, where the algorithm for Tree Contraction by Heggernes et al. forms the base case of our inductive hypothesis.

We start with few observation regarding the graph class $\mathbb{T}_\ell$, which will be useful while designing the algorithm.

▶ **Observation 1.** *For each $T \in \mathbb{T}_\ell$ the following statements hold.*
1. *The chromatic number of $T$ is at most $2\sqrt{\ell}+2$.*
2. *If $T'$ is a graph obtained by subdividing an edge in $T$ then $T' \in \mathbb{T}_\ell$.*
3. *If $T'$ is a graph obtained by contracting an edge in $T$ then $T' \in \mathbb{T}_\ell$.*

Let $(G, k)$ be an instance of $\mathbb{T}_\ell$-Contraction. The measure we use for analysing the running time of our algorithm is $\mu = \mu(G, k) = k$. We start by applying some simple reduction rules.

▶ **Reduction Rule 3.1.** *If $k < 0$ then return that $(G, k)$ is a no instance of $\mathbb{T}_\ell$-Contraction.*

▶ **Reduction Rule 3.2.** *If $k = 0$ and $G \in \mathbb{T}_\ell$ then return that $(G, k)$ is a yes instance of $\mathbb{T}_\ell$-Contraction.*

▶ **Reduction Rule 3.3.** *If $G$ is a disconnected or $k = 0$ and $G \notin \mathbb{T}_\ell$ then return that $(G, k)$ is a no instance.*

We assume that the input graph is 2-connected, and design an algorithm for input restricted to 2-connected graphs. Later, we will show how we can remove this constraint. The key idea behind the algorithm is to use a coloring of $V(G)$ with at most $2\sqrt{\ell}+2$ colors to find a $T$-witness structure (if it exists) of $G$, where $G$ is contractible to $T \in \mathbb{T}_\ell$ using at most $k$ edge contractions (see Observation 1). Moreover, if such a $T$ does not exist then we must correctly conclude that $(G, k)$ is a *no* instance of $\mathbb{T}_\ell$-Contraction. Towards this, we introduce the following notion.

▶ **Definition 2.** Let $G$ be a 2-connected graph, $T$ be a graph in $\mathbb{T}_\ell$, $\mathcal{W}$ be a $T$-witness structure of $G$, and $\phi : V(G) \to [2\sqrt{\ell}+2]$ be a coloring of $V(G)$. Furthermore, let $T_S$ be a

(fixed) spanning tree of $T$, $M = \{t, t' \mid tt' \in E(T) \setminus E(T_S)\} \cup \{t \in V(T) \mid d_T(t) \geq 3\}$, and $B = \{t \in V(T) \mid |W(t)| \geq 2\}$. We say that $\phi$ is $\mathcal{W}$-compatible if the following conditions are satisfied.

1. For all $W \in \mathcal{W}$, and $w, w' \in W$ we have $\phi(w) = \phi(w')$.
2. For all $t, t' \in M \cup B$ such that $tt' \in E(T)$ we have $\phi(W(t)) \neq \phi(W(t'))$.

We refer to the set $M \cup B$ as the set of marked vertices.

Assume that $(G, k)$ is a *yes* instance of $\mathbb{T}_\ell$-Contraction, and $F$ be one of its (inclusion-wise) minimal solution. Furthermore, let $T = G/F$, and $\mathcal{W}$ be the $T$-witness structure of $G$. Suppose we are given $G$ and a $\mathcal{W}$ compatible coloring $\phi : V(G) \to [2\sqrt{\ell} + 2]$ of $G$, but we are neither given $\mathcal{W}$ nor $T$. We will show how we can compute a $T'$ witness structure $\mathcal{W}'$ of $G$ such that $|V(T')| \geq |V(T)|$, where $T' \in \mathbb{T}_\ell$. Informally, we will find such a witness structure by either concluding that none of the edges are part of the solution, some specific set of edges are part of the solution, or finding a star-like structure of the monochromatic components of size at least 2 in $G$, with respect to $\phi$. Towards this, we will employ the algorithm for Connected Vertex Cover (CVC) by Cygan [7].

▶ **Proposition 3** ([7]). CVC *admits an algorithm running in time* $2^k n^{\mathcal{O}(1)}$. *Here,* $k$ *is the size of a solution and* $n$ *is the number of vertices in the input graph.*

We note that we use the algorithm of Cygan [7] instead of the algorithm by Cygan et al. [9], because the latter algorithm is a randomized algorithm. Also, the algorithm given by Proposition 3 can be used to output a solution.

Consider the case when $G$ is $k$-contractable to a graph, say $T \in \mathbb{T}_\ell$, and let $\mathcal{W}$ be a $T$-witness structure of $G$. Furthermore, let $\phi : V(G) \to [2\sqrt{\ell}+2]$ be a $\mathcal{W}$-compatible coloring of $G$, and $\mathcal{X}$ be the set of monochromatic components of $\phi$. We prove some lemmata showing useful properties of $\mathcal{X}$.

▶ **Lemma 4.** *Let* $T'$ *be the graph with* $\mathcal{X}$ *as the* $T'$*-witness structure of* $G$. *Then* $T' \in \mathbb{T}_\ell$ *and* $|V(T')| \leq |V(T)|$.

Next, we proceed to show how we can partition each $X \in \mathcal{X}$ into many smaller witness sets such that either we obtain $\mathcal{W}$ or a $T'$-witness structure of $G$ for some $T' \in \mathbb{T}_\ell$ which has at least as many vertices as $T$. Towards this, we introduce the following notions.

For $X \in \mathcal{X}$, by $\hat{X}$ we denote the set of vertices that have a neighbor outside of $X$, *i.e.* $\hat{X} = N(V(G) \setminus X)$. A *shatter* of $X$ is a partition of $X$ into sets such that one of them is a connected vertex cover $C$ of $G[X]$ containing all the vertices in $\hat{X}$ and all other sets are of size 1. The size of a shatter of $X$ is the of size of $C$. Furthermore, a shatter of $X$ is minimum if there is no other shatter with strictly smaller size.

From Lemma 4 (and Definition 2) it follows that for each $X \in \mathcal{X}$ there is $\mathcal{W}_X \subseteq \mathcal{W}$ such that $X = \cup_{Y \in \mathcal{W}_X} Y$. In the following lemma, we prove some properties of sets in $\mathcal{W}_X$, which will be useful in the algorithm design.

▶ **Lemma 5.** *Consider* $X \in \mathcal{X}$ *with* $|X| \geq 2$, $\mathcal{W}_X \subseteq \mathcal{W}$ *such that* $X = \cup_{Y \in \mathcal{W}_X} Y$, *and all of the following conditions are satisfied.*

- $G[X] = (u, v_1, \ldots, v_q, v)$ *is an induced path, where* $q \in \mathbb{N}$.
- *For each* $i \in [q]$ *we have* $deg(v_i) = 2$.
- *There exists* $X' \in \mathcal{X} \setminus \{X\}$ *such that* $N(u) \cap X' \neq \emptyset$ *and* $N(v) \cap X' \neq \emptyset$.

*Then* $|\mathcal{W}_X| = 1$.

▶ **Lemma 6.** *Consider* $X \in \mathcal{X}$ *with* $|X| \geq 2$, $\mathcal{W}_X$ *such that* $X = \cup_{Y \in \mathcal{W}_X} Y$, *and all the following conditions are satisfied.*

- $G[X] = (v_0, v_1, \ldots, v_q, v)$ *is an induced path, where* $q \in \mathbb{N}$.
- *For each* $i \in [q]$ *we have* $deg(v_i) = 2$.
- *There exists no* $X' \in \mathcal{X}$ *such that* $N(u) \cap X' \neq \emptyset$ *and* $N(v) \cap X' \neq \emptyset$.

*Then* $|\mathcal{W}_X| = |X|$.

Next, we show that each $X \in \mathcal{X}$ for which Lemma 5 and 6 are not applicable must contain exactly one big witness set. Moreover, the unique big witness set (together with other vertices as singleton sets) forms one of its shatters.

▶ **Lemma 7.** *For* $X \in \mathcal{X}$ *with* $|X| \geq 2$*, let* $\mathcal{W}_X \subseteq \mathcal{W}$ *such that* $X = \cup_{Y \in \mathcal{W}_X} Y$*. Furthermore, the set* $X$ *does not satisfy the conditions of Lemma 5 or 6. Then there is exactly one big witness set in* $\mathcal{W}_X$*.*

▶ **Lemma 8.** *Consider* $X \in \mathcal{X}$ *such that* $|X| \geq 2$ *and it contains a big witness set, and it does not satisfy conditions of Lemma 5 or 6. Let* $\mathcal{W}_X \subseteq \mathcal{W}$ *such that* $X = \cup_{Y \in \mathcal{W}_X} Y$*, and* $W^*$ *be the (unique) big witness set in* $X$*. Then* $W^*$ *is a connected vertex cover of* $G[X]$ *and it contains* $\hat{X}$*.*

Using Lemma 6 to Lemma 8 we show how we can replace each $X \in \mathcal{X}$ with the sets of its shatter. Recall that we are given only $G$ and $\phi$, and therefore we know $\mathcal{X}$, but we do not know $\mathcal{W}$. In the Lemma 9, we show how we can find a $T'$-witness structure of $G$ for some $T' \in \mathbb{T}_\ell$, which has at least as many vertices as $T$ (without knowing $\mathcal{W}$).

▶ **Lemma 9.** *Given* $\mathcal{X}$*, we can obtain a* $T'$*-witness structure of* $G$ *in time* $2^k n^{\mathcal{O}(1)}$ *time, where* $T' \in \mathbb{T}_\ell$ *and* $|V(T')| \geq |V(T)|$*.*

Now we are ready to present our randomized algorithm for $\mathbb{T}_\ell$-Contraction when input graph is 2-connected.

▶ **Theorem 10.** *There is a Monte Carlo algorithm for solving* $\mathbb{T}_\ell$-Contraction *on 2-connected graphs running in time* $\mathcal{O}((2\sqrt{\ell}+2)^{\mathcal{O}(k+\ell)} \cdot n^{\mathcal{O}(1)})$*, where* $n$ *is the number of vertices in the input graph. It does not return false positive and returns correct answer with probability at least* $1 - 1/e$*.*

**Proof.** Let $(G, k)$ be an instance of $\mathbb{T}_\ell$-Contraction, where $G$ is a 2-connected graph. Furthermore, the Reduction Rules 3.1 and 3.3 are not applicable, otherwise we can correctly decide whether or not $(G, k)$ is a *yes* instance. The algorithm starts by computing a random coloring $\phi : V(G) \to [2\sqrt{\ell}+2]$, by choosing a color for each vertex uniformly and independently at random. Let $\mathcal{X}$ be the set of monochromatic connected components with respect to $\phi$ in $G$. The algorithm applies Lemma 9 in time $2^k n^{\mathcal{O}(1)}$ and tries to compute $T'$ such that $T' \in \mathbb{T}_\ell$ and $G$ is $k$-contractible to $T'$. It runs $(2\sqrt{\ell}+2)^{6k+8\ell}$ many iterations of two steps mentioned above. If for any such iteration it obtains a desired $T'$-witness structure of $G$ then it returns *yes*. If none of the iterations yield *yes* then the algorithm returns *no*. This completes the description of the algorithm.

Observe that the algorithm returns *yes* only if it has found a $T' \in \mathbb{T}_\ell$ such that $G$ is contractible to $T'$ using at most $k$ edge contractions. Therefore, when it outputs *yes*, then indeed $(G, k)$ is a *yes* instance of $\mathbb{T}_\ell$-Contraction. We now argue that if $(G, k)$ is a *yes* instance then using a random coloring the algorithm (correctly) returns the answer with sufficiently high probability. Let $T$ be a graph in $\mathbb{T}_\ell$, such that $G$ is $k$-contractible to $T$, and $\mathcal{W}$ be a $T$-witness structure of $G$. Furthermore, let $T_S$ be a (fixed) spanning tree of $T$, and vertex set $M$, $B$ are set of vertices defined in Definition 2. Let $\psi : V(G) \to [2\sqrt{\ell}+2]$ be a coloring where colors are chosen uniformly at random for each vertex. The total number

of vertices contained in big witness sets of $\mathcal{W}$ is at most $2k$. By our assumption, every leaf is a singleton witness set and it is adjacent to a big witness set. Here, we assume that the number of vertices in $T$ is at least 3, otherwise we can solve the problem in polynomial time. This implies that no leaf is in $M \cup B$. Consider graph $T'$ obtained from $T$ by deleting all the leaves and deleting edges in $E(T_\ell) \setminus E(T_S)$. All the marked vertices of $T_\ell$ and all the paths connecting two marked vertices are also present in $T'$. Notice that $T'$ is tree with at most $k + 2\ell$ leaves. Since the number of vertices of degree three is at most the number of leaves in any tree, there are at most $k + 2\ell$ vertices of degree at least 3. There are at most $k$ vertices in $T$ which are big witness sets and at most $2\ell$ vertices incident to edges in $E(T_\ell) \setminus E(T_S)$. Hence the total number of marked vertices is at most $2k + 4\ell$. Since $T'$ is a tree, there are at most $2k + 4\ell$ vertices which lie on a path between two vertices in $M \cup B$ and are adjacent to one of these. The number of vertices of $G$ which are marked vertices or vertices which are adjacent to it in $T'$ is at most $2(2k+4\ell)+2k$. Therefore, the probability that $\psi$ is compatible with $\mathcal{W}$ is at least $1/(2\sqrt{\ell}+2)^{6k+8\ell}$. Since the algorithm runs $(2\sqrt{\ell}+2)^{6k+8\ell}$ many iterations, probability that none of these colorings which is generated uniformly at random is compatible with $\mathcal{W}$ is at most $(1 - 1/(2\sqrt{\ell}+2)^{6k+8\ell})^{(2\sqrt{\ell}+2)^{6k+8\ell}} < 1/e$. Hence, algorithm returns a solution on positive instances with probability at least $1 - 1/e$. Each iteration takes $2^k \cdot n^{\mathcal{O}(1)}$ time and hence the total running time of the algorithm is $\mathcal{O}((2\sqrt{\ell}+2)^{\mathcal{O}(k+\ell)} \cdot n^{\mathcal{O}(1)})$. ◀

Next, we design reduction rules and a branching rule whose (exhaustive) application will ensure that the instance of $\mathbb{T}_\ell$-Contraction we are dealing with is 2-connected. Either we apply one of these reduction rules or branching rule, or we resolve the instance using the algorithm for $\mathbb{T}_{\ell'}$-Contraction, where $\ell' < \ell$. This together with Theorem 10 gives us an algorithm for $\mathbb{T}_\ell$-Contraction on general graphs.

▶ **Lemma 11.** *If for some $0 \leq \ell' < \ell$, $(G, k)$ is a yes instance of $\mathbb{T}_{\ell'}$-Contraction then return that $(G, k)$ is a yes instance of $\mathbb{T}_\ell$-Contraction.*

Our next reduction rule deals with vertices of degree of 1.

▶ **Reduction Rule 3.4.** *If there is $v \in V(G)$ such that $d(v) = 1$ then delete $v$ from $G$. The resulting instance is $(G - \{v\}, k)$.*

If a connected graph $G$ is not 2-connected graph then there is a cut vertex say, $v$ in $G$. Let $C_1, C_2, \ldots, C_t$ be the components of $G - \{v\}$. Furthermore, let $G_1 = G[V(C_1) \cup \{v\}]$ and $G_2 = G - V(C_1)$. Next, we try to resolve the instance (if possible) using the following lemma.

▶ **Lemma 12.** *If there exists $\ell_1$ and $\ell_2$ with $\ell_1 + \ell_2 = \ell$, where $\ell_1, \ell_2 > 0$, and $k_1$ and $k_2$ with $k_1 + k_2 = k$ such that $(G_1, k_1)$ is a yes instance of $\mathbb{T}_{\ell_1}$-Contraction and $(G_2, k_2)$ is a yes instance of $\mathbb{T}_{\ell_2}$-Contraction then return that $(G, k)$ is a yes instance of $\mathbb{T}_\ell$-Contraction.*

Notice that if Lemma 12 is not applicable then one of $G_1$ or $G_2$ must be contracted to a tree. Let $k_1$ be the smallest integer such that $(G_1, k_1)$ is a *yes* instance of $\mathbb{T}$-Contraction, and $k_2$ be the smallest integer such that $(G_2, k_2)$ is a *yes* instance of $\mathbb{T}$-Contraction. Notice that $k_1$ and $k_2$ can be computed in (deterministic) time $4^k n^{\mathcal{O}(1)}$ using the algorithm for $\mathbb{T}$-Contraction [17]. We next proceed with the following branching rule.

▶ **Branching Rule 3.1.** *We branch depending on which of the graphs among $G_1$ and $G_2$ are contracted to a tree. Therefore, we branch as follows.*

- *Contract $G_1$ to a tree, and the resulting instance is $(G_2, k - k_1)$.*
- *Contract $G_2$ to a tree, and the resulting instance is $(G_1, k - k_2)$.*

Note that the measure strictly decreases in each of the branches of the Branching Rule 3.1 since Reduction Rule 3.4 is not applicable. If we are unable to resolve the instance using Lemma 11 and 12, and Reduction Rules 3.3 and 3.4 and Branching Rule 3.1 are not applicable then the input graph is 2-connected. And, then we resolve the instance using Theorem 10.

▶ **Theorem 13.** *For each $\ell \in \mathbb{N}$, there is a Monte Carlo algorithm for solving $\mathbb{T}_\ell$-*Contraction *with running in time $\mathcal{O}((2\sqrt{\ell}+2)^{\mathcal{O}(k+\ell)} \cdot n^{\mathcal{O}(1)})$. It does not return false positive and returns correct answer with probability at least $1 - 1/e$.*

## 4 Derandomization

In this section, we derandomize the algorithm presented in Section 3. Before proceeding forward we define the following important object of this section.

▶ **Definition 14** (Universal Family)**.** A $(n, k, q)$-universal family is a collection $\mathcal{F}$, of functions from $[n]$ to $[q]$ such that for each $S \subseteq [n]$ of size $k$ and a function $\phi : S \to [q]$, there exists function $f \in \mathcal{F}$ such that $f|_S \equiv \phi$.

Here, $f|_S$ denotes the function $f$ when restricted to the elements of $S$. For $q = 2$, the universal family defined above is called an $(n, k)$-universal set [20]. Hence, $(n, k, q)$-universal family is a generalization of $(n, k)$-universal set. The main result of this section is the following theorem (Theorem 15), which we use to derandomize the algorithm presented in Section 3.

▶ **Theorem 15.** *For any $n, k, q \geq 1$, one can construct an $(n, k, q)$-universal family of size $\mathcal{O}(q^k \cdot k^{\mathcal{O}(k)} \cdot \log n)$ in time $\mathcal{O}(q^k \cdot k^{\mathcal{O}(k)} \cdot n \log n)$.*

Before proceeding to the proof of Theorem 15, we state how we use it to derandomize the algorithm presented in Section 3. Let $(G, k)$ be an instance of $\mathbb{T}_\ell$-Contraction. Assume that $(G, k)$ is a *yes* instance of $\mathbb{T}_\ell$-Contraction, and let $F$ be one of its solution. Furthermore, let $T = G/F$, where $T \in \mathbb{T}_\ell$ and $\mathcal{W}$ be the $T$-witness structure of $G$, and $\phi : V(G) \to [2\sqrt{\ell}+2]$ be a $\mathcal{W}$-compatible coloring of $G$. Recall that our randomized algorithm starts by coloring vertices in $G$ uniformly and independently at random, and then uses this coloring to extract a witness structure out of each color classes. We then argued that any random coloring is "equally good" as that of $\phi$ with sufficiently high probability, which is given by a function of $k$ (and $\ell$). To derandomize this algorithm, we construct a family $\mathcal{F}$ of (coloring) functions from $[n]$ to $[2\sqrt{\ell}+2]$. We argue that one of the colorings in the family that we compute is "equally good" as that of $\phi$. Recall that the number of vertices which we need to be colored in a specific way for a coloring to be $\mathcal{W}$-compatible is bounded by $6k + 8\ell$ (see Definition 2 and Theorem 10). Let $S$ be the set of vertices in $G$ which needs to be colored in a specific way as per the requirements of Definition 2. We can safely assume that $|S| = 6k + 8\ell$. If this is not the case we can add arbitrary vertices in $S$ to ensure this. Notice that any coloring $f$ of $G$ such that $f|_S = \phi|_S$ also satisfies the requirements of Definition 2. Let $\mathcal{F}$ be an $(n, 6k+8\ell, 2\sqrt{\ell}+2)$-universal family constructed using Theorem 15. Instead of using random coloring in the algorithm presented in Section 3, we can iterate over functions in $\mathcal{F}$. Notice that we do not know $S$ but for any such $S$, we are guaranteed to find an appropriate coloring in one of the functions in $\mathcal{F}$, which gives us the desired derandomization of the algorithm.

In rest of the section, we focus on the prove of Theorem 15. Overview of the proof is as follows: Let $S$ be a set of size $k$ in an $n$-sized universe $U$. We first *reduce* this universe $U$ to another universe $U'$ whose size is bounded by $k^2$. We ensure that all elements of $S$ are mapped to different elements of $U'$ during this reduction. Let $Y$ be the range of $S$ in $U'$.

We further partition $U'$ into $\log k$ parts such that $Y$ is almost equally divided among these partition. In other words, each partition contains (roughly) $k/\log k$ many elements of $Y$. For each of these parts, we explicitly store functions which represents all possible $q$-coloring of elements of $Y$ in this partition. Finally, we "pull back" these functions to obtain a coloring of $S$.

▶ **Definition 16** (Splitter [20]). An $(n, k, q)$-splitter $\mathcal{F}$ is a family of functions from $[n]$ to $[q]$ such that for every set $S \subseteq [n]$ of size $k$ there exists a function $f \in \mathcal{F}$ that splits $S$ evenly. That is, for every $1 \leq z, z' \leq q$, $|f^{-1}(z) \cap S|$ and $|f^{-1}(z') \cap S|$ differ by at most 1.

▶ **Lemma 17.** *For every $1 \leq k, q \leq n$ there is a family of $(n, k, q)$-splitter of size $\mathcal{O}(n^{\mathcal{O}(q)})$ which can be constructed in the same time.*

Following is another well known result for construction of splitter when $q = k^2$. We use this result to *reduce* the size of the universe.

▶ **Proposition 18** ([8, 20]). *For any $n, k \geq 1$ one can construct an $(n, k, k^2)$-splitter of size $\mathcal{O}(k^{\mathcal{O}(1)} \log n)$ in time $\mathcal{O}(k^{\mathcal{O}(1)} n \log n)$.*

Next, we look at the $k$-Restriction problem defined by Naor et al. [20]. Before defining the problem, we define some terminologies that will be useful. For a fixed set of alphabets, say $\{1, 2, \ldots, b\}$ and a vector *vector* $V$, which is an ordered collection of alphabets, the length of $V$ is the size of the collection. We represent $n$ length vector $V$ as $(v_1, v_2, \ldots, v_n)$. For a positive integer $i \in [n]$, $V[i]$ denotes the alphabet at the $i^{th}$ position of $V$. Similarly, for an (index) set $S \subseteq [n]$, $V[S]$ denotes the $|S|$ sized vector obtained by taking alphabet at $i^{th}$ position in $V$, for each $i \in S$. In other words, if $S = \{i_1, i_2, \ldots, i_k\}$ for $i_1 < i_2 < \cdots < i_k$, then $V[S] = (V[i_1], V[i_2], \ldots, V[i_k])$. An input to the $k$-Restriction problem is a set $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ called as a $k$-restrictions, where $C_j \subseteq [b]^k$ for $j \in [m]$ and an integer $n$. Here, $[b]^k$ denotes the set of all possible vectors of length $k$ over $[b]$, and $m$ denotes the size of the $k$-restrictions. We say that a collection $\mathcal{V}$ of vectors *obeys* $\mathcal{C}$ if for all $S \subseteq [n]$ which is of size $k$ and for all $C_j \in \mathcal{C}$, there exists $V \in \mathcal{V}$ such that $V[S] \in C_j$. The goal of $k$-Restriction problem is to find a collection $\mathcal{V}$ of as small cardinality as possible, which obeys $\mathcal{C}$. Let $c = \min_{j \in [m]} |C_i|$, and let $T$ be the time needed to check whether or not the vector $V$ is in $C_j$. We next state the result of Naor et al. [20], which will be useful for proving Theorem 15.

▶ **Proposition 19** (Theorem 1 [20]). *For any $k$-Restriction problem with $b \leq n$, there is a deterministic algorithm that outputs a collection obeying $k$-restrictions, which has size at most $(k \log n + \log m)/\log(b^k/(b^k - c))$. Moreover, the algorithm runs in time $\mathcal{O}\big(\frac{b^k}{c}\binom{n}{k} \cdot m \cdot T \cdot n^k\big)$. Here, $b$ is the size of the alphabet set, $m$ is the size of the $k$-restrictions, $n$ is the size of the vectors in the output set, and $c$ is the size of the smallest collection in the $k$-restrictions.*

Notice that a function from $[n]$ to $[q]$ can be seen as an $n$-length vector over the alphabet set $[q]$. Consider the case when each $C_j$ contains exactly one vector of length $k$ over $[q]$, *i.e.* $\mathcal{C} = \{\{C\} \mid C \in [q]^k\}$, $m = q^k$, $c = 1$, and $T = \mathcal{O}(n)$. The output of $k$-Restriction on this input is exactly an $(n, k, q)$-universal family. Therefore, we obtain the following corollary.

▶ **Corollary 20.** *For any $n, k, q \geq 1$, one can construct an $(n, k, q)$-universal family of size $\mathcal{O}(q^k \cdot k \cdot (\log n + \log q))$ in time $\mathcal{O}(q^k \cdot n^{\mathcal{O}(k)})$.*

Notice that we can not directly employ Corollary 20 to construct the desired family, since its running time is $\mathcal{O}(q^k \cdot n^{\mathcal{O}(k)})$. Therefore, we carefully use splitter to construct an $(n, k, q)$-universal family to obtain the desired running time.

*Proof of Theorem 15.* For the sake of clarity in the notations, we assume that $\log k$ and $k/\log k$ are integers. Let $\mathcal{A}$ be a $(n, k, k^2)$-splitter obtained by Proposition 18. Let $\mathcal{B}$ be a $(k^2, k, \log k)$-splitter obtained by Lemma 17. Let $\mathcal{D}$ be a $(k^2, k/\log k, q)$-universal family obtained by Corollary 20. We construct $\mathcal{F}$ as follows. For every function $f_a$ in $\mathcal{A}$, $f_b$ in $\mathcal{B}$, and $\log k$ functions $g_1, g_2, \ldots, g_{\log k}$ in $\mathcal{D}$, we construct a tuple $f = (f_a, f_b, g_1, g_2, \ldots, g_{\log k})$, and add it to $\mathcal{F}$. We note here that $g_1, g_2, \ldots, g_{\log k}$ need not be different functions. For $f \in \mathcal{F}$, we define $f : [n] \to [q]$ as follows. For $x \in [n]$, we have $f(x) = g_r(f_b(f_a(x)))$, where $r = f_b(f_a(x))$.

We first argue about the size of $\mathcal{F}$ and the time needed to construct it. Notice that $|\mathcal{F}| \leq |\mathcal{A}||\mathcal{B}||\mathcal{D}|^{\log k}$. We know $|\mathcal{A}| \leq k^{\mathcal{O}(1)} \log n$, $|\mathcal{B}| \leq \mathcal{O}(k^{\mathcal{O}(\log k)})$ and $|\mathcal{D}| \leq q^{k/\log k} k^{\mathcal{O}(k/\log k)}$ by Proposition 18, Lemma 17, and Corollary 20, respectively. This implies that $|\mathcal{F}| \in \mathcal{O}(q^k \cdot k^{\mathcal{O}(\log k)} \cdot \log n)$. Note that $\mathcal{A}, \mathcal{B}, \mathcal{D}$ can be constructed in time $\mathcal{O}(k^{\mathcal{O}(1)} n \log n)$, $\mathcal{O}(k^{\mathcal{O}(\log k)})$, and $\mathcal{O}(q^k \cdot k^{\mathcal{O}(k/\log k)})$, respectively. This implies that time required to construct $\mathcal{F}$ is bounded by $\mathcal{O}(q^k \cdot k^{\mathcal{O}(k)} \cdot n \log n)$.

It remains to argue that $\mathcal{F}$ has the desired properties. Consider $S \subseteq [n]$ of size $k$ and $\phi : S \to [q]$. We prove that there exists a function $f \in \mathcal{F}$ such that $f|_S \equiv \phi$. By the definition of splitter, there exists $f_a \in \mathcal{A}$ such that $f_a$ evenly splits $S$ (see Definition 16). Since $|S| < k^2$, for every $y \in [k^2]$, $|f_a^{-1}(y) \cap S|$ is either 0 or 1. Let $Y = \{y_1, y_2, \ldots, y_k\}$ be a subset of $[k^2]$ such that $y_1 < y_2 < \cdots < y_k$ and $|f_a^{-1}(y_i) \cap S| = 1$, for all $i \in [k]$. For $j = k/\log k$, we mark every $j^{th}$ element in set $Y$ marking $\log k - 1$ indices altogether. In other words, construct a subset $Y'$ of $Y$ of cardinality $\log k - 1$ such that $Y' = \{y_{1j}, y_{2j}, y_{3j} \ldots, y_{(\log k - 1)j}\}$. We use the set $Y'$ to partition $[k^2]$ in a way that every partition contains almost $k/\log k$ many elements of $Y$. Let $y_0 = 0$ and $y_{(\log k)j} = k^2$ and define set $Y_r = \{y \in Y \mid y_{r-1} < y \leq y_r\}$ for $r \in [\log k]$. Recall that a $\mathcal{B}$ is $(k^2, k, \log k)$-splitter family obtained by Lemma 17. By construction, there exists a function $f_b$ which corresponds to subset $Y'$ of $\log k - 1$ many indices. In other words, there is a function $f_b$ such that $f_b^{-1}(r)$ contains all the elements in $Y_r$, for each $r$ in $[\log k]$. We note that size of $f_b^{-1}(r)$ could be as large as $k^2$. Recall that $\mathcal{D}$ is a $(k^2, k/\log k, q)$-universal family. Therefore, for every $r \in [\log k]$ there exists $g_r \in \mathcal{D}$ such that $g_r|_{Y_r} \equiv \phi|_{Y_r}$. Consider a function $f = (f_a, f_b, g_1, g_2, \ldots, g_{\log k})$ in $\mathcal{F}$ where $f_a, f_b$ and $g_r$ satisfies the property mentioned above. The function $f_a$ is bijective on $S$ and $f(S) = Y$. The function $f_b$ partitions $Y$ into $\log k$ many parts by mapping $Y$ into $Y_1, Y_2, \ldots, Y_{\log k}$. For each $Y_r$ there exists a function $g_r$ which gives the desired coloring of elements in $Y_r$ and hence for the elements in $S$. Since we considering all possible combinations of $f_a, f_b$ and $\log k$ functions in $\mathcal{D}$, there exists a function $f$ such that $f|_S \equiv \phi$, which proves the theorem. ◀

## 5 Non-existence of a Polynomial Kernel for $\mathbb{T}_\ell$-Contraction

In this section, we show that $\mathbb{T}_\ell$-Contraction does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly. We note that $\mathbb{T}$-Contraction (Tree Contraction) does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly [17]. We give a reduction from $\mathbb{T}$-Contraction to $\mathbb{T}_\ell$-Contraction as follows.

Let $(G, k)$ be an instance of $\mathbb{T}$-Contraction. We create an instance $(G', k')$ of $\mathbb{T}_\ell$-Contraction as follows. Initially, we have $G = G'$. Let $v^*$ be an arbitrarily chosen vertex in $V(G)$. For each $i \in [\ell]$, we add a cycle $(v^*, w_1^i, w_2^i, \ldots, w_{k+1}^i)$ on $k+2$ vertices to $G'$, which pairwise intersect at $v^*$, and we set $k' = k$. It is easy to see that $(G, k)$ is a *yes* instance of $\mathbb{T}$-Contraction if and only if $(G', k')$ is a *yes* instance of $\mathbb{T}_\ell$-Contraction.

▶ **Theorem 21.** $\mathbb{T}_\ell$-Contraction *does not admit a polynomial kernel unless* NP $\subseteq$ coNP/poly.

## 6 PSAKS for $\mathbb{T}_\ell$-Contraction

In this section, we design a PSAKS for $\mathbb{T}_\ell$-Contraction, which complements the result that $\mathbb{T}_\ell$-Contraction does not admit a polynomial kernel assuming NP $\not\subseteq$ coNP/poly (Section 5).

Let $(G, k)$ be an instance of $\mathbb{T}_\ell$-Contraction. The algorithm starts by applying Reduction Rules 3.1 to 3.4 (if applicable, in that order). Next, we state the following lemma which will be useful in designing a reduction rule which will be employed for bounding the sizes of induced paths.

▶ **Lemma 22.** *Let $(G, k)$ be an instance of $\mathbb{T}_\ell$-Contraction and $P = (u_0, u_1, \ldots, u_q, u_{q+1})$ be a path in $G$, where $q \geq k + 2$, and for each $i \in [q + 1]$ we have $deg(u_i) = 2$. Then no minimal solution $F$ to $\mathbb{T}_\ell$-Contraction in $(G, k)$ with $|F| \leq k$ contains an edge incident to $V(P) \setminus \{u_0, u_{q+1}\}$.*

▶ **Reduction Rule 6.1.** *If $G$ has a path $P = (u_0, u_1, \ldots, u_q, u_{q+1})$ such that $q > k + 2$ and for all $i \in [q]$, we have $deg(u_i) = 2$. Then contract the edge $u_{q-1}u_q$,* i.e. *the resulting instance is $(G/\{u_{q-1}u_q\}, k)$.*

Note that Reduction Rule 6.1 can be applied in polynomial time by searching for such a path (if it exists) in the subgraph induced on the vertices of degree 2 in $G$.

▶ **Lemma 23.** *Consider an instance $(G, k)$ of $\mathbb{T}_\ell$-Contraction on which Reduction Rule 6.1 is not applicable. If $(G, k)$ is a yes instance of $\mathbb{T}_\ell$-Contraction then $G$ has a connected vertex cover of size at most $2(k + 3)(k + 2\ell)$.*

Before describing the next reduction rule, we define a partition of $V(G)$ into the following sets. Let $H = \{u \in V(G) \mid deg(u) \geq 2(k + 3)(k + 2\ell) + 1\}$, $I = \{v \in V(G) \setminus H \mid N(v) \subseteq H\}$, and $R = V(G) \setminus (H \cup I)$. Vertices $v, u$ are said to be *false twins* if $N(v) = N(u)$. We use Lemma 24 to reduce the number of vertices in $I$ which have many false twins. Let $G$ be $k$-contractible to a graph $T$ in $\mathbb{T}_\ell$ and $\mathcal{W}$ be the $T$-witness structure of $G$.

▶ **Lemma 24.** *Consider sets $X, U \subseteq V(G)$ such that $U$ is an independent set in $G$ and for all $v \in U$ we have $X \subseteq N(v)$. If $|U| \geq k + \ell + 2$ then there is a vertex $t \in V(T)$ such that $X \subseteq W(t)$.*

▶ **Reduction Rule 6.2.** *If there is a vertex $v \in I$ that has at least $k + \ell + 2$ false twins in $I$ then delete $v$,* i.e. *the resulting instance is $(G - \{v\}, k)$.*

For $\alpha > 1$, we let $d = \lceil \frac{\alpha}{\alpha - 1} \rceil$. Next, we state our last reduction rule.

▶ **Reduction Rule 6.3.** *If there are vertices $v_1, v_2, \cdots, v_{k+\ell+2} \in I$ and $h_1, h_2, \cdots, h_d \in H$ such that for all $i \in [k + \ell + 2]$, we have $\{h_1, \ldots, h_d\} \subseteq N(v_i)$ then contract all edges in $\tilde{E} = \{v_1 h_i \mid i \in [d]\}$, and decrease $k$ by $d - 1$. The resulting instance is $(G/\tilde{E}, k - d + 1)$.*

We note that the *lossy-ness* is introduced only in the Reduction Rule 6.3. We have determined that $H' = \{h_1, h_2, \ldots, h_d\}$ need to be in one witness bag but $G[H']$ may not be connected. To simplify the graph, we introduce additional vertex $v_1$ to the bag which contains $H'$. By doing this we are able to contract $H' \cup \{v_1\}$ into a single vertex. In the following lemma, we argue that the number of extra edge contracted in this process is $\alpha$ factor of the optimum solution.

▶ **Lemma 25.** *Let $(G, k)$ be an instance of $\mathbb{T}_\ell$-Contraction where none of the Reduction Rules 6.1 to 6.3 are applicable. If $(G, k)$ is a* yes *of $\mathbb{T}_\ell$-Contraction then $|V(G)| \leq c[k(k + 2\ell)]^{d+1}$, where $c$ is some fixed constant.*

▶ **Theorem 26.** *$\mathbb{T}_\ell$-Contraction admits a strict PSAKS, where the number of vertices is bounded by $c[k(k + 2\ell)]^{(\lceil \frac{\alpha}{\alpha-1} \rceil + 1)}$, where $c$ is some fixed constant.*

## References


1 Akanksha Agrawal, Lawqueen Kanesh, Saket Saurabh, and Prafullkumar Tale. Paths to trees and cacti. In *CIAC*, pages 31–42, 2017.

2 Akanksha Agrawal, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Split contraction: The untold story. In *STACS*, volume 66 of *LIPIcs*, pages 5:1–5:14, 2017.

3 Takao Asano and Tomio Hirata. Edge-Contraction Problems. *Journal of Computer and System Sciences*, 26(2):197–208, 1983.

4 Rémy Belmonte, Petr A. Golovach, Pim Hof, and Daniël Paulusma. Parameterized complexity of three edge contraction problems with degree constraints. *Acta Informatica*, 51(7):473–497, 2014.

5 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

6 Leizhen Cai and Chengwei Guo. Contracting few edges to remove forbidden induced subgraphs. In *IPEC*, pages 97–109, 2013.

7 Marek Cygan. Deterministic parameterized connected vertex cover. In *Scandinavian Workshop on Algorithm Theory*, pages 95–106. Springer, 2012.

8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

9 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS*, pages 150–159, 2011.

10 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

11 Rod G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer-Verlag, 1997.

12 Rod G. Downey and Michael R. Fellows. *Fundamentals of Parameterized complexity*. Springer-Verlag, 2013.

13 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

14 Petr A. Golovach, Pim van 't Hof, and Daniel Paulusma. Obtaining planarity by contracting few edges. *Theoretical Computer Science*, 476:38–46, 2013.

15 Sylvain Guillemot and Dániel Marx. A faster FPT algorithm for bipartite contraction. *Inf. Process. Lett.*, 113(22–24):906–912, 2013.

16 Pinar Heggernes, Pim van 't Hof, Daniel Lokshtanov, and Christophe Paul. Obtaining a bipartite graph by contracting few edges. *SIAM Journal on Discrete Mathematics*, 27(4):2143–2156, 2013.

17 Pinar Heggernes, Pim van 't Hof, Benjamin Lévêque, Daniel Lokshtanov, and Christophe Paul. Contracting graphs to paths and trees. *Algorithmica*, 68(1):109–132, 2014.

18 Wenjun Li, Qilong Feng, Jianer Chen, and Shuai Hu. Improved kernel results for some FPT problems based on simple observations. *Theor. Comput. Sci.*, 657:20–27, 2017.

19 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. On the hardness of eliminating small induced subgraphs by contracting edges. In *IPEC*, pages 243–254, 2013.

20 Moni Naor, Leonard J Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *FOCS*, pages 182–191. IEEE, 1995.

21 Rolf Niedermeier. *Invitation to fixed-parameter algorithms*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006.

22 Toshimasa Watanabe, Tadashi Ae, and Akira Nakamura. On the removal of forbidden graphs by edge-deletion or by edge-contraction. *Discrete Applied Mathematics*, 3(2):151–153, 1981.

23 Toshimasa Watanabe, Tadashi Ae, and Akira Nakamura. On the NP-hardness of edge-deletion and-contraction problems. *Discrete Applied Mathematics*, 6(1):63–78, 1983.

# Finding Small Weight Isomorphisms with Additional Constraints is Fixed-Parameter Tractable*†

**Vikraman Arvind[1], Johannes Köbler[‡2], Sebastian Kuhnert[§3], and Jacobo Torán[¶4]**

1 **Institute of Mathematical Sciences (HBNI), Chennai, India**
arvind@imsc.res.in
2 **Institut für Informatik, Humboldt-Universität zu Berlin, Germany**
koebler@informatik.hu-berlin.de
3 **Institut für Informatik, Humboldt-Universität zu Berlin, Germany**
kuhnert@informatik.hu-berlin.de
4 **Institut für Theoretische Informatik, Universität Ulm, Germany**
toran@uni-ulm.de

## Abstract

Lubiw showed that several variants of Graph Isomorphism are NP-complete, where the solutions are required to satisfy certain additional constraints [12]. One of these, called Isomorphism With Restrictions, is to decide for two given graphs $X_1 = (V, E_1)$ and $X_2 = (V, E_2)$ and a subset $R \subseteq V \times V$ of forbidden pairs whether there is an isomorphism $\pi$ from $X_1$ to $X_2$ such that $i^\pi \neq j$ for all $(i, j) \in R$. We prove that this problem and several of its generalizations are in fact in FPT:

- The problem of deciding whether there is an isomorphism between two graphs that moves $k$ vertices and satisfies Lubiw-style constraints is in FPT, with $k$ and $|R|$ as parameters. The problem remains in FPT even if a CNF of such constraints is allowed. As a consequence of the main result it follows that the problem to decide whether there is an isomorphism that moves exactly $k$ vertices is in FPT. This solves a question left open in [1].
- When the number of moved vertices is unrestricted, finding isomorphisms that satisfy a CNF of Lubiw-style constraints is in $\mathsf{FPT}^{\mathrm{GI}}$.
- Checking if there is an isomorphism between two graphs that has complexity $t$ is also in FPT with $t$ as parameter, where the complexity of a permutation $\pi$ is the Cayley measure defined as the minimum number $t$ such that $\pi$ can be expressed as a product of $t$ transpositions.
- We consider a more general problem in which the vertex set of a graph $X$ is partitioned into Red and Blue, and we are interested in an automorphism that stabilizes Red and Blue and moves exactly $k$ vertices in Blue, where $k$ is the parameter. This problem was introduced in [5], and in [1] we showed that it is W[1]-hard even with color classes of size 4 inside Red. Now, for color classes of size at most 3 inside Red, we show the problem is in FPT.

In the non-parameterized setting, all these problems are NP-complete. Also, they all generalize in several ways the problem to decide whether there is an isomorphism between two graphs that moves at most $k$ vertices, shown to be in FPT by Schweitzer [13].

* Some proofs are omitted from this extended abstract; see https://arxiv.org/abs/1709.10063 for the full version.

† This work was supported by the Alexander von Humboldt Foundation in its research group linkage program.
‡ The author is supported by DFG grant KO 1053/7-2.
§ The author is supported by DFG grant KO 1053/7-2.
¶ The author is supported by DFG grant TO 200/3-2.



12th International Symposium on Parameterized and Exact Computation (IPEC 2017).
Editors: Daniel Lokshtanov and Naomi Nishimura; Article No. 2; pp. 2:1–2:13

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

**1998 ACM Subject Classification** F.1.3 Complexity Measures and Classes, F.2 Analysis of Algorithms and Problem Complexity

**Keywords and phrases** Parameterized algorithms, hypergraph isomorphism, mislabeled graphs

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.2

## 1 Introduction

The Graph Isomorphism problem (GI) consists in deciding whether two given input graphs are isomorphic, i.e., whether there is a bijection between the vertex sets of the two graphs that preserves the adjacency relation. It is an intensively researched algorithmic problem for over four decades, culminating in Babai's recent quasi-polynomial time algorithm [2].

There is also considerable work on the parameterized complexity of GI. For example, already in 1980 it was shown [7] that GI, parameterized by color class size, is fixed-parameter tractable (FPT). It is also known that GI, parameterized by the eigenvalue multiplicity of the input graph, is in FPT [3]. More recently, GI, parameterized by the treewidth of the input graph, is shown to be in FPT [11].

In a different line of research, Lubiw [12] has considered the complexity of GI with additional constraints on the isomorphism. Exploring the connections between GI and the NP-complete problems, Lubiw defined the following version of GI.

Isomorphism With Restrictions: Given two graphs $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$ and a set of forbidden pairs $R \subseteq V_1 \times V_2$, decide whether there is an isomorphism $\pi$ from $X_1$ to $X_2$ such that $i^\pi \neq j$ for all $(i, j) \in R$.

When $X_1 = X_2$, the problem is to check if there is an automorphism that satisfies these restrictions. Lubiw showed that the special case of testing for *fixed-point-free automorphisms* is NP-complete. Klavík et al. recently reexamined Isomorphism With Restrictions [10]. They show that it remains NP-complete when restricted to graph classes for which GI is as hard as for general graphs. Conversely, they show that it can be solved in polynomial time for several graph classes for which the isomorphism problem is known to be solvable in polynomial time by combinatorial algorithms, e.g. planar graphs and bounded treewidth graphs. However, they also show that the problem remains NP-complete for bounded color class graphs, where an efficient group theoretic isomorphism algorithm is known.

A different kind of constrained isomorphism problem was introduced by Schweitzer [13]. The weight (or support size) of a permutation $\pi \in \mathrm{Sym}(V)$ is $|\{i \in V \mid i^\pi \neq i\}|$. Schweitzer showed that the problem of testing if there is an isomorphism $\pi$ of weight at most $k$ between two $n$-vertex input graphs in the same vertex set can be solved in time $k^{\mathcal{O}(k)}\,\mathrm{poly}(n)$. Hence, the problem is in FPT with $k$ as parameter. Schweitzer's algorithm exploits interesting properties of the structure of an isomorphism $\pi$. Based on Lubiw's reductions [12], it is not hard to see that the problem is NP-complete when $k$ is not treated as parameter.

In this paper we consider the problem of finding isomorphisms with additional constraints in the parameterized setting. In our main result we formulate a *graph isomorphism/automorphism problem with additional constraints* that generalizes Lubiw's setting as follows. For a graph $X = (V, E)$, let $\pi \in \mathrm{Aut}(X)$ be an automorphism of $X$. We say that a permutation $\pi \in \mathrm{Sym}(V)$ *satisfies* a formula $F$ over the variables in $\mathrm{Var}(V) = \{x_{u,v} \mid u, v \in V\}$ if $F$ is satisfied by the assignment that has $x_{u,v} = 1$ if and only if $u^\pi = v$. For example, the conjunction $\bigwedge_{u \in V} \neg x_{uu}$ expresses the condition that $\pi$ is fixed-point-free. We define:

Exact-CNF-GI: Given two graphs $X_1 = (V, E_1)$ and $X_2 = (V, E_2)$, a CNF formula $F$ over $\mathrm{Var}(V)$, and $k \in \mathbb{N}$, decide whether there is an isomorphism from $X_1$ to $X_2$ that has

weight exactly $k$ and satisfies $F$. The parameter is $|F| + k$, where $|F|$ is the number of variables used in $F$.

In Section 4, we first give an FPT algorithm for Exact-CNF-GA, the automorphism version of this problem. The algorithm uses an orbit shrinking technique that allows us to transform the input graph into a graph with bounded color classes, preserving the existence of an exact weight $k$ automorphism that satisfies the formula $F$. The bounded color class version is easy to solve using color coding; see Section 3 for details. Building on this, we show that Exact-CNF-GI is also in FPT. In particular, this allows us to efficiently find isomorphisms of weight exactly $k$, a problem left open in [1], and extends Schweitzer's result mentioned above to the *exact* case. In our earlier paper [1] we have shown that the problem of *exact weight $k$ automorphism* is in FPT using a simpler orbit shrinking technique which does not work for exact weight $k$ isomorphisms. In this paper, we use some extra group-theoretic machinery to obtain a more versatile orbit shrinking.

In Section 5, we consider the problem of computing graph isomorphisms of *complexity* exactly $t$: The complexity of a permutation $\pi \in \mathrm{Sym}(V)$ is the minimum number of transpositions whose product is $\pi$. Checking for automorphisms or isomorphisms of complexity exactly $t$ is NP-complete in the non-parameterized setting. We show that the problem is in FPT with $t$ as parameter. Again, the "at most $t$" version of this problem was already shown to be in FPT by Schweitzer [13] as part of his algorithmic strategy to solve the weight at most $k$ problem. Our results in Sections 4 and 5 also hold for hypergraphs when the maximum hyperedge size is taken as additional parameter.

In Section 6, we examine a different restriction on the automorphisms being searched for. Consider graphs $X = (V, E)$ with vertex set partitioned into Red and Blue. The *Colored Graph Automorphism* problem (defined in [5]; we denote it Col-GA), is to check if $X$ has an automorphism that respects the partition and moves exactly $k$ Blue vertices. We showed in [1] that this problem is W[1]-hard. In our hardness proof the orbits of the vertices in the Red part of the graph have size at most 4, while the ones for the Blue vertices have size 2. We show here that this cannot be restricted any further: If the input graph has Red further partitioned into color classes of size at most 3 each, then the problem to test whether there is an automorphism moving exactly $k$ Blue vertices can be solved in FPT (with parameter $k$). The Blue part of the graph remains unconstrained. Observe that Schweitzer's problem [13] coincides with the special case of this problem where there are no Red vertices. This implies that the non-parameterized version of Col-GA is NP-complete (even when $X$ has only Blue vertices). Similarly, finding weight $k$ automorphisms of a hypergraph reduces to Col-GA by taking the incidence graph, where the original vertices become Blue and the vertices for hyperedges are Red; note that this yields another special case, where both Red and Blue induce the empty graph, respectively.

## 2 Preliminaries

We use standard permutation group terminology, see e.g. [4]. Given a permutation $\sigma \in \mathrm{Sym}(V)$, its *support* is $\mathrm{supp}(\sigma) = \{u \in V \mid u^\sigma \neq u\}$ and its *(Hamming) weight* is $|\mathrm{supp}(\sigma)|$. The *complexity* of $\sigma$ (sometimes called its *Cayley weight*) is the minimum number $t$ such that $\sigma$ can be written as the product of $t$ transpositions.

Let $G \leq \mathrm{Sym}(V)$ and $\pi \in \mathrm{Sym}(V)$; this includes the case $\pi = \mathrm{id}$. A permutation $\sigma \in G\pi \setminus \{\mathrm{id}\}$ has *minimal complexity in $G\pi$* if for every way to express $\sigma$ as the product of a minimum number of transpositions $\sigma = \tau_1 \cdots \tau_{\mathrm{compl}(\sigma)}$ and every $i \in \{2, \ldots, \mathrm{compl}(\sigma)\}$ it holds that $\tau_i \cdots \tau_{\mathrm{compl}(\sigma)} \notin G\pi$. The following lemma observes that every element of $G\pi$ can be decomposed into minimal-complexity factors.

▶ **Lemma 2.1** [1, Lemma 2.2]. *Let $G\pi$ be a coset of a permutation group $G$ and let $\sigma \in G\pi \setminus \{\mathrm{id}\}$. Then for some $\ell \geq 1$ there are $\sigma_1, \ldots, \sigma_{\ell-1} \in G$ with minimal complexity in $G$ and $\sigma_\ell \in G\pi$ with minimal complexity in $G\pi$ such that $\sigma = \sigma_1 \cdots \sigma_\ell$ and $\mathrm{supp}(\sigma_i) \subseteq \mathrm{supp}(\sigma)$ for each $i \in \{1, \ldots, \ell\}$.*

An action of a permutation group $G \leq \mathrm{Sym}(V)$ on a set $V'$ is a group homomorphism $h\colon G \to \mathrm{Sym}(V')$; we denote the image of $G$ under $h$ by $G(V')$. For $u \in V$, we denote its stabilizer by $G_u = \{\pi \in G \mid u^\pi = u\}$. For $U \subseteq V$, we denote its pointwise stabilizer by $G_{[U]} = \{\pi \in G \mid \forall u \in U : u^\pi = u\}$ and its setwise stabilizer by $G_{\{U\}} = \{\pi \in G \mid U^\pi = U\}$. For $S \subseteq \mathcal{P}(V)$, we let $G_S = \{\pi \in G \mid \forall U \in S : U^\pi = U\}$.

A *hypergraph* $X = (V, E)$ consists of a vertex set $V$ and a hyperedge set $E \subseteq \mathcal{P}(V)$. Graphs are the special case where $|e| = 2$ for all $e \in E$. The *degree* of a vertex $v \in V$ is $|\{e \in E | v \in e\}|$. A *(vertex) coloring* of $X$ is a partition of $V$ into color classes $\mathcal{C} = (C_1, \ldots, C_m)$. The color classes $\mathcal{C}$ are *$b$-bounded* if $|C_i| \leq b$ for all $i \in [m]$. An *isomorphism* between two hypergraphs $X = (V, E)$ and $X' = (V', E')$ (with color classes $\mathcal{C} = (C_1, \ldots, C_m)$ and $\mathcal{C}' = (C'_1, \ldots, C'_m)$) is a bijection $\pi\colon V \to V'$ such that $E' = \big\{\{\pi(v) | v \in e\} \big| e \in E\big\}$ (and $C'_i = \big\{\pi(v) \big| v \in C_i\big\}$). The isomorphisms from $X$ to $X'$ form a coset that we denote by $\mathrm{Iso}(X, X')$. The automorphisms of a hypergraph $X$ are the isomorphisms from $X$ to itself; they form a group which we denote by $\mathrm{Aut}(X)$.

## 3 Bounded color class size

To show that Exact-CNF-GA for hypergraphs with $b$-bounded color classes can be solved in FPT, we recall our algorithm for exact weight $k$ automorphisms of bounded color class hypergraphs [1] and show how it can be adapted to the additional constraints given by the input formula.

▶ **Definition 3.1.** Let $X = (V, E)$ be a hypergraph with color class set $\mathcal{C} = \{C_1, \ldots, C_m\}$.

**(a)** For a subset $\mathcal{C}' \subseteq \mathcal{C}$, we say that a color-preserving permutation $\pi \in \mathrm{Sym}(V)$ *$\mathcal{C}'$-satisfies* a CNF formula $F$ over $\mathrm{Var}(V)$ if every clause of $F$ contains a literal $x_{u,v}$ or $\neg x_{u,v}$ with $u \in \bigcup \mathcal{C}'$ that is satisfied by $\pi$.

**(b)** For a color-preserving permutation $\pi \in \mathrm{Sym}(V)$, let $\mathcal{C}[\pi] = \{C_i \in \mathcal{C} \mid \exists v \in C_i : v^\pi \neq v\}$ be the subset of color classes that intersect $\mathrm{supp}(\pi)$. For a subset $\mathcal{C}' \subseteq \mathcal{C}[\pi]$, we define the permutation $\pi_{\mathcal{C}'} \in \mathrm{Sym}(V)$ as

$$\pi_{\mathcal{C}'}(v) = \begin{cases} v^\pi, & \text{if } v \in \bigcup \mathcal{C}', \\ v, & \text{if } v \notin \bigcup \mathcal{C}'. \end{cases}$$

Note that $\pi_{\mathcal{C}[\pi]} = \pi$.

**(c)** A color-preserving automorphism $\sigma \neq \mathrm{id}$ of $X$ is said to be *color-class-minimal*, if for every set $\mathcal{C}'$ with $\varnothing \subsetneq \mathcal{C}' \subsetneq \mathcal{C}[\sigma]$, the permutation $\sigma_{\mathcal{C}'}$ is not in $\mathrm{Aut}(X)$.

▶ **Lemma 3.2.** *Let $X = (V, E)$ be a hypergraph with color class set $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$. For $\varnothing \neq \mathcal{C}' \subseteq \mathcal{C}$ and a CNF formula $F$ over $\mathrm{Var}(V)$, the following statements are equivalent:*

- *There is a nontrivial automorphism $\sigma$ of $X$ with $\mathcal{C}[\sigma] = \mathcal{C}'$ that satisfies $F$.*
- *$\mathcal{C}'$ can be partitioned into $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$ and $F$ (seen as a set of clauses) can be partitioned into CNF formulas $F_0, \ldots, F_\ell$ such that $F_0$ is $(\mathcal{C} \setminus \mathcal{C}')$-satisfied by* id *and for each $i \in \{1, \ldots, \ell\}$ there is a color-class-minimal automorphism $\sigma_i$ of $X$ with $\mathcal{C}[\sigma_i] = \mathcal{C}_i$ that $\mathcal{C}_i$-satisfies $F_i$.*

*Moreover, the automorphisms $\sigma$ and $\sigma_i$ can be chosen to satisfy $\sigma_i = \sigma_{\mathcal{C}_i}$ for $1 \leq i \leq \ell$, respectively.*

In [1] an algorithm is presented that, when given a hypergraph $X$ on vertex set $V$ with $b$-bounded color classes and $k \in \mathbb{N}$, computes all color-class-minimal automorphisms of $X$ that have weight exactly $k$ in $\mathcal{O}\big((kb!)^{\mathcal{O}(k^2)} \operatorname{poly}(N)\big)$ time. We use it as a building block for the following algorithm (see line 5).

**Algorithm 1** $\texttt{Color-Exact-CNF-HGA}_b(X, \mathcal{C}, k, F)$

1 **Input**: A hypergraph $X = (V, E)$ with $b$-bounded color classes $\mathcal{C} = \{C_1, \dots, C_m\}$, a parameter $k \in \mathbb{N}$, and a CNF formula $F$ over $\operatorname{Var}(V)$
2 **Output**: A color-preserving automorphism $\sigma$ of $X$ with $|\operatorname{supp}(\sigma)| = k$ that satisfies $F$, or $\perp$ if none exists
3 $A_0 = \{\mathrm{id}\}$
4 **for** $i \in \{1, \dots, k\}$ **do**
5 $\quad A_i \leftarrow \{\sigma \in \operatorname{Aut}(X) \mid \sigma$ is color-class-minimal and has weight $i\}$ *// see [1]*
6 **for** $h \in \mathcal{H}_{\mathcal{C},k}$ **do** *// $\mathcal{H}_{\mathcal{C},k}$ is the perfect family of hash functions $h\colon \mathcal{C} \to [k]$ from [6]*
7 $\quad$ **for** $\ell \in \{1, \dots, k\}, h'\colon [k] \to [\ell]$ **do**
8 $\quad\quad$ **for** $(k_1, \dots, k_\ell) \in \{0, \dots, k\}^\ell$ with $\sum_{i=1}^{\ell} k_i = k$ **do**
9 $\quad\quad\quad$ **for** each partition of the clauses of $F$ into $F_0, \dots, F_\ell$ **do**
10 $\quad\quad\quad\quad$ **if** $\forall i \in \{1, \dots, \ell\} : \exists \sigma_i \in A_{k_i} : \operatorname{supp}(\sigma_i) \subseteq \bigcup (h' \circ h)^{-1}(i)$ **and** $F_i$ is $\mathcal{C}[\sigma_i]$-satisfied by $\sigma_i$, **and** $F_0$ is $(\mathcal{C} \setminus \bigcup_{i=1}^{\ell} \mathcal{C}[\sigma_i])$-satisfied by id **then**
11 $\quad\quad\quad\quad\quad$ **return** $\sigma = \sigma_1 \cdots \sigma_\ell$
12 **return** $\perp$

▶ **Theorem 3.3.** *Given a hypergraph $X = (V, E)$ with $b$-bounded color classes $\mathcal{C}$, a CNF formula $F$ over $\operatorname{Var}(V)$, and $k \in \mathbb{N}$, the algorithm $\texttt{Color-Exact-CNF-HGA}_b(X, \mathcal{C}, k, F)$ computes a color-preserving automorphism $\sigma$ of $X$ of weight $k$ that satisfies $F$ in $(kb!)^{\mathcal{O}(k^2)} k^{\mathcal{O}(|F|)} \operatorname{poly}(N)$ time (where $N$ is the size of $X$), or determines that none exists.*

**Proof.** If the algorithm returns $\sigma = \sigma_1 \cdots \sigma_\ell$, we know $\sigma_i \in A_{k_i}$ and $\operatorname{supp}(\sigma_i) \subseteq \bigcup (h' \circ h)^{-1}(i)$. As these sets are disjoint, we have $|\operatorname{supp}(\sigma)| = \sum_{i=1}^{\ell} |\operatorname{supp}(\sigma_i)| = k$, and Lemma 3.2 implies that $\sigma$ satisfies $F$.

We next show that the algorithm does not return $\perp$ if there is an automorphism $\pi$ of $X$ that has weight $k$ and satisfies $F$. By Lemma 3.2, we can partition $\mathcal{C}[\pi]$ into $\mathcal{C}_1, \dots, \mathcal{C}_\ell$ and the clauses of $F$ into $F_0 \dots, F_\ell$ such that $F_0$ is $(\mathcal{C} \setminus \mathcal{C}[\pi])$-satisfied by id and, for $1 \le i \le \ell$, the permutation $\pi_i = \pi_{\mathcal{C}_i}$ is a color-class-minimal automorphism of $X$ that $\mathcal{C}[\pi_i]$-satisfies $F$. Now consider the iteration of the loop where $h$ is injective on $\mathcal{C}[\pi]$; such an $h$ must exist as it is chosen from a perfect hash family. Now let $h'\colon [k] \to [\ell]$ be a function with $h'\big(h(C)\big) = i$ if $C \in \mathcal{C}[\pi_i]$; such an $h'$ exists because $h$ is injective on $\mathcal{C}[\pi]$. In the loop iterations where $h'$ and the partition of $F$ into $F_0 \dots, F_\ell$ is considered, the condition on line 10 is true (at least) with $\sigma_i = \pi_i$, so the algorithm does not return $\perp$.

Line 5 can be implemented by using the algorithm $\texttt{ColoredAut}_{k,b}(X)$ from [1] which runs in $\mathcal{O}\big((kb!)^{\mathcal{O}(k^2)} \operatorname{poly}(N)\big)$ time, and this also bounds $|A_i|$. As $|\mathcal{C}| \le n$, the perfect hash family $\mathcal{H}_{\mathcal{C},k}$ has size $2^{\mathcal{O}(k)} \log^2 n$, and can also be computed in this time. The inner loops take at most $k^k$, $k^k$ and $(k+1)^{|F|}$ iterations, respectively. Together, this yields a runtime of $(kb!)^{\mathcal{O}(k^2)} k^{\mathcal{O}(|F|)} \operatorname{poly}(N)$. ◀

## 4 Exact weight

In this section, we show that finding isomorphisms that have an exactly prescribed weight and satisfy a CNF formula is fixed parameter tractable. In fact, we show that this is true even for hypergraphs, when the maximum hyperedge size $d$ is taken as additional parameter.

Exact-CNF-HGI: Given two hypergraphs $X_1 = (V, E_1)$ and $X_2 = (V, E_2)$ with hyperedge size bounded by $d$, a CNF formula $F$ over $\mathrm{Var}(V)$, and $k \in \mathbb{N}$, decide whether there is an isomorphism from $X_1$ to $X_2$ of weight $k$ that satisfies $F$. The parameter is $|F| + k + d$.

Our approach is to reduce Exact-CNF-HGI to Exact-CNF-HGA (the analogous problem for automorphisms), which we solve first.

We require some permutation group theory definitions. Let $G \leq \mathrm{Sym}(V)$ be a permutation group. The group $G$ partitions $V$ into orbits: $V = \Omega_1 \cup \Omega_2 \cdots \cup \Omega_r$. On each orbit $\Omega_i$, the group $G$ acts transitively. A subset $\Delta \subseteq \Omega_i$ is a *block* of the group $G$ if for all $\pi \in G$ either $\Delta^\pi = \Delta$ or $\Delta^\pi \cap \Delta = \varnothing$. Clearly, $\Omega_i$ is itself a block, and so are all singleton sets. These are *trivial* blocks. Other blocks are *nontrivial*. If $G$ has no nontrivial blocks it is *primitive*. If $G$ is not primitive, we can partition $\Omega_i$ into blocks $\Omega_i = \Delta_1 \cup \Delta_2 \cup \cdots \cup \Delta_s$, where each $\Delta_j$ is a *maximal nontrivial block*. Then the group $G$ acts primitively on the block system $\{\Delta_1, \Delta_2, \ldots, \Delta_s\}$. In this action, a permutation $\pi \in G$ maps $\Delta_i$ to $\Delta_i^\pi = \{u^\pi \mid u \in \Delta_i\}$.

The following two theorems imply that every primitive group on a sufficiently large set $V$ contains the alternating group $\mathrm{Alt}(V) = \{\pi \in \mathrm{Sym}(G) \mid \mathrm{compl}(\pi) \text{ is even}\}$.

▶ **Theorem 4.1** [4, Theorem 3.3A]. *Suppose $G \leq \mathrm{Sym}(V)$ is a primitive subgroup of $\mathrm{Sym}(V)$. If $G$ contains an element $\pi$ such that $|\mathrm{supp}(\pi)| = 3$ then $G$ contains $\mathrm{Alt}(V)$. If $G$ contains an element $\pi$ such that $|\mathrm{supp}(\pi)| = 2$ then $G = \mathrm{Sym}(V)$.*

▶ **Theorem 4.2** [4, Theorem 3.3D]. *If $G \leq \mathrm{Sym}(V)$ is primitive with $G \notin \{\mathrm{Alt}(V), \mathrm{Sym}(V)\}$ and contains an element $\pi$ such that $|\mathrm{supp}(\pi)| = m$ (for some $m \geq 4$) then $|V| \leq (m-1)^{2m}$.*

The following lemma implies that the alternating group in a large orbit survives fixing vertices in a smaller orbit.

▶ **Lemma 4.3.** *Let $G \leq \mathrm{Sym}(\Omega_1 \cup \Omega_2)$ be a permutation group such that $\Omega_1$ is an orbit of $G$, and $|\Omega_1| \geq 5$. Recall that $G(\Omega_i)$ denotes the image of $G$ under its action on $\Omega_i$. Suppose $G(\Omega_1) \in \{\mathrm{Alt}(\Omega_1), \mathrm{Sym}(\Omega_1)\}$ and $|G(\Omega_1)| > |G(\Omega_2)|$. Then for some subgroup $H$ of $G(\Omega_2)$, the group $G$ contains the product group $\mathrm{Alt}(\Omega_1) \times H$. In particular, the pointwise stabilizer $G_{[\Omega_2]}$ contains the subgroup $\mathrm{Alt}(\Omega_1) \times \{\mathrm{id}\}$.*

The effect of fixing vertices of some orbit on other orbits of the same size depends on how the group relates these orbits to each other.

▶ **Definition 4.4.** Two orbits $\Omega_1$ and $\Omega_2$ of a permutation group $G \leq \mathrm{Sym}(V)$ are *linked* if there is a group isomorphism $\sigma\colon G(\Omega_1) \to G(\Omega_2)$ with $G(\Omega_1 \cup \Omega_2) = \big\{(\varphi, \sigma(\varphi)) \,\big|\, \varphi \in G(\Omega_1)\big\}$. (This happens if and only if both $G(\Omega_1)$ and $G(\Omega_2)$ are isomorphic to $G(\Omega_1 \cup \Omega_2)$.)

We next show that two large orbits where the group action includes the alternating group are (nearly) independent unless they are linked.

▶ **Lemma 4.5.** *Suppose $G \leq \mathrm{Sym}(V)$ where $V = \Omega_1 \cup \Omega_2$ is its orbit partition such that $|\Omega_i| \geq 5$ and $G(\Omega_i) \in \{\mathrm{Alt}(\Omega_i), \mathrm{Sym}(\Omega_i)\}$ for $i = 1, 2$. Then either $\Omega_1$ and $\Omega_2$ are linked in $G$, or $G$ contains $\mathrm{Alt}(\Omega_1) \times \mathrm{Alt}(\Omega_2)$.*

The last ingredient for our algorithm is the observation that when there are two linked orbits where the group action includes the alternating group, fixing a vertex in one orbit is equivalent to fixing some vertex of the other orbit.

▶ **Lemma 4.6** [4, Theorem 5.2A]. *Let $n = |V| > 9$. Suppose $G$ is a subgroup of $\mathrm{Alt}(V)$ of index strictly less than $\binom{n}{2}$. Then, for some point $u \in V$, the group $G$ is the pointwise stabilizer subgroup $\mathrm{Alt}(V)_u$.*

▶ **Corollary 4.7.** *Let $\Omega_1$ and $\Omega_2$ be two linked orbits of a permutation group $G \leq \mathrm{Sym}(V)$ with $\mathrm{Alt}(\Omega_1) \leq G(\Omega_1)$ and $|\Omega_1| = |\Omega_2| > 9$. Then for each $u \in \Omega_1$ there is a $v \in \Omega_2$ such that $G_u = G_v$.*

**Algorithm 2** $\texttt{Exact-CNF-HGA}_d(X, k, F)$

```
1  Input: A hypergraph X with hyperedge size bounded by d, a parameter k and a formula F
2  Output: An automorphism σ of X with |supp(σ)| = k that satisfies F, or ⊥ if none exists
3  T ← the vertices of X that are mentioned in F
4  G ← ⟨ {σ ∈ Aut(X) | σ has minimal complexity in Aut(X) and |supp(σ)| ≤ k} ⟩
     // see [1, Algorithm 3]
5  b ← (k/2) · max{(k − 1)^{2k}, |T| + k, 9}
6  while G contains an orbit of size more than b do
7      repeat
8          O ← the set of all G-orbits
9          for Ω ∈ O do
10             B(Ω) ← a maximal block system of Ω in G
11             if ∃Δ ∈ B(Ω) : |Δ| > k/2 or |B(Ω)| > (k − 1)^{2k} ∧ Alt(B(Ω)) ⋠ G(B(Ω)) then
12                 G ← G_{B(Ω)} // the setwise stabilizer of all Δ ∈ B(Ω)
13     until G remains unchanged
14     choose Ω_max ∈ O such that |B(Ω_max)| ≥ |B(Ω)| for all Ω ∈ O
15     if |B(Ω_max)| > max{(k − 1)^{2k}, |T| + k, 9} then
16         H ← G_[T] // the pointwise stabilizer of T
17         Ω_H ← the largest H-orbit that is contained in Ω_max
18         B_H ← {Δ ∈ B(Ω_max) | Δ ⊆ Ω_H}
19         choose Δ ∈ B_H
20         G ← G_{Δ} // the setwise stabilizer of Δ
21 O ← the set of all G-orbits
22 return Color-Exact-CNF-HGA_b(X, O, k, F) // see Algorithm 1
```

▶ **Theorem 4.8.** *Algorithm 2 solves* Exact-CNF-HGA *in time* $\big(d(k^k + |F|)!\big)^{\mathcal{O}(k^2)} \mathrm{poly}(N)$.

**Proof sketch.** Suppose there is some $\pi \in \mathrm{Aut}(X)$ of weight exactly $k$ that satisfies $F$. By Lemma 2.1, the automorphism $\pi$ can be decomposed as a product of minimal-complexity automorphisms of weight at most $k$, which implies $\pi \in G$ after line 4. To show that whenever the algorithm shrinks $G$, some weight $k$ automorphism of $X$ that satisfies $F$ survives, we first consider the shrinking in line 12: If $\Omega$ is an orbit with $|\Delta| > k/2$ for some (and thus all) $\Delta \in \mathcal{B}(\Omega)$, then no block of $\Omega$ is moved by $\pi$. If $|\mathcal{B}(\Omega)| > (k-1)^{2k}$ and $G\big(\mathcal{B}(\Omega)\big)$ does not contain $\mathrm{Alt}\big(\mathcal{B}(\Omega)\big)$, then Theorems 4.1 and 4.2 imply that $\pi$ setwise stabilizes all $\Delta \in \mathcal{B}(\Omega)$ and thus survives the shrinking. The other shrinking of $G$, which occurs in line 20, can only happen if $\mathrm{Alt}\big(\mathcal{B}(\Omega_{\max})\big) \leq G\big(\mathcal{B}(\Omega_{\max})\big)$. Let $\mathcal{T} = \bigcup_{\Omega \in \mathcal{O}} \{\Delta \in \mathcal{B}(\Omega) \mid \Delta \cap T \neq \varnothing\}$ be the set of all blocks with vertices from $T$ and let $R = G_{\mathcal{T}}$ be the setwise stabilizer of these blocks. Note that $H \leq R \leq G$. Using Lemmas 4.3 and 4.5 and Corollary 4.7, it can be shown that a sufficiently large part of $\mathrm{Alt}\big(\mathcal{B}(\Omega_{\max})\big)$ survives in $R$ and also in $H$.

▶ **Claim.** *$\mathcal{B}_H$ is a maximal block system for the orbit $\Omega_H$ in $H$. Moreover, $|\mathcal{B}_H| > k$ and $\mathrm{Alt}(\mathcal{B}_H) \leq H(\mathcal{B}_H)$.*

Building on this, it can be shown that when $G$ and $\Delta$ are as in line 20, then for any $\pi \in G$ of weight $k$ that satisfies $F$, there is a $\pi' \in G_{\{\Delta\}}$ of weight $k$ that satisfies $F$. ◀

We now turn to Exact-CNF-HGI. Given a formula $F$ over $\mathrm{Var}(V)$ and $\psi \in \mathrm{Sym}(V)$, let $\psi(F)$ denote the formula obtained from $F$ by replacing each variable $x_{uv}$ by $x_{u\psi(v)}$.

▶ **Lemma 4.9.** *A product $\sigma = \varphi\pi \in \mathrm{Sym}(V)$ satisfies a formula $F$ over $\mathrm{Var}(V)$ if and only if $\varphi$ satisfies $\pi^{-1}(F)$.*

**Algorithm 3** $\texttt{Exact-CNF-HGI}_d(X_1, X_2, k, F)$

**Input**: Two hypergraphs $X_1$ and $X_2$ on vertex set $V$ with hyperedge size bounded by $d$, a parameter $k \in \mathbb{N}$ and a CNF formula $F$ over $\mathrm{Var}(V)$
**Output**: An isomorphism $\sigma$ from $X_1$ to $X_2$ with $|\mathrm{supp}(\sigma)| = k$ that satisfies $F$, or $\perp$ if none exists
$\pi \leftarrow$ some isomorphism from $X_1$ to $X_2$ with $|\mathrm{supp}(\pi)| \leq k$ // *see [1, Theorem 3.8]*
**for** $U \subseteq \mathrm{supp}(\pi)$ **do** // *we will force $u \notin \mathrm{supp}(\varphi\pi)$ for $u \in U$*
  **for** $M \subseteq \mathrm{supp}(\pi) \setminus U$ **do** // *we will force $u \in \mathrm{supp}(\varphi) \cap \mathrm{supp}(\varphi\pi)$ for $u \in M$*
    $I \leftarrow \mathrm{supp}(\pi) \setminus (U \cup M)$ // *we will force $u \notin \mathrm{supp}(\varphi)$ for $u \in I$*
    $F' \leftarrow \pi^{-1}(F) \wedge \bigwedge_{u \in U} x_{u\pi^{-1}(u)} \wedge \bigwedge_{u \in M} (\neg x_{u\pi^{-1}(u)} \wedge \neg x_{u,u}) \wedge \bigwedge_{u \in I} x_{uu}$
    $k' \leftarrow k - |I| + |U|$
    $\varphi \leftarrow \texttt{Exact-CNF-HGA}_d(X_1, k', F')$ // *see Algorithm 2*
    **if** $\varphi \neq \perp$ **then return** $\sigma = \varphi\pi$
**return** $\perp$

▶ **Theorem 4.10.** *Algorithm 3 solves* Exact-CNF-HGI *in time* $\big(d(k^k + |F|)!\big)^{\mathcal{O}(k^2)} \mathrm{poly}(N)$.

**Proof.** Suppose Algorithm 3 returns a permutation $\sigma = \varphi\pi$. Then $\pi$ is an isomorphism from $X_1$ to $X_2$ and $\varphi$ is an automorphism of $X_1$ that satisfies $F'$ and has weight $k'$. As $\varphi$ satisfies $\pi^{-1}(F)$, Lemma 4.9 implies that $\sigma$ satisfies $F$. The additional literals in $F'$ ensure $\mathrm{supp}(\sigma) = (\mathrm{supp}(\varphi) \setminus U) \cup I$ and thus $\big|\mathrm{supp}(\sigma)\big| = k' - |U| + |I| = k$.

Now suppose there is an isomorphism $\sigma$ from $X_1$ to $X_2$ that satisfies $F$ and has weight $k$. Let $\pi$ be the isomorphism computed on line 3. Then $\varphi = \sigma\pi^{-1}$ is an automorphism of $X_1$; it satisfies $\pi^{-1}(F)$ by Lemma 4.9. In the iteration of the loops where $U = \big\{u \in \mathrm{supp}(\pi) \cap \mathrm{supp}(\varphi) \,\big|\, u^{\varphi\pi} = u\big\}$ and $M = \big(\mathrm{supp}(\pi) \cap \mathrm{supp}(\varphi)\big) \setminus U$, it holds that $\varphi$ has weight $k'$ and satisfies $F'$. Thus $\texttt{Exact-CNF-HGA}_d(X_1, k', F')$ does not return $\perp$.

The isomorphism $\pi$ can be found in $(dk)^{\mathcal{O}(k^2)} \mathrm{poly}(N)$ time [1, Theorem 3.8]. The loops have at most $3^k$ iterations, and $\texttt{Exact-CNF-HGA}_d$ takes $\big(d(k^k + |F|)!\big)^{\mathcal{O}(k^2)} \mathrm{poly}(N)$ time. ◀

## 5 Exact complexity

The complexity of a permutation $\pi \in \mathrm{Sym}(V)$ can be bounded by functions of its weight: $\big|\mathrm{supp}(\pi)\big| - 1 \leq \mathrm{compl}(\pi) \leq 2 \cdot \big|\mathrm{supp}(\pi)\big|$. However, there is no direct functional dependence between these two parameters. And while the algorithms of Sections 3 and 4 can be modified to find isomorphisms of exactly prescribed complexity, we give an independent and more efficient algorithm in this section.

The main ingredient is an analysis of decompositions $\sigma = \sigma_1 \cdots \sigma_\ell$ of $\sigma \in \mathrm{Sym}(V)$ into $\sigma_i \in \mathrm{Sym}(V) \setminus \{\mathrm{id}\}$ (for $1 \leq i \leq \ell$) with $\mathrm{compl}(\sigma) = \sum_{i=1}^{\ell} \mathrm{compl}(\sigma_i)$; we call such decompositions *complexity-additive*. For example, the decomposition into complexity-minimal permutations provided by Lemma 2.1 is complexity-additive.

For a sequence of permutations $\sigma_1, \ldots, \sigma_\ell \in \mathrm{Sym}(V)$ and a coloring $c\colon V \to [k]$, its *colored cycle graph* $\mathrm{CG}_c(\sigma_1, \ldots, \sigma_\ell)$ is the incidence graph between $\bigcup_{i=1}^{\ell} \mathrm{supp}(\sigma_i)$ and the $\sigma_i$-orbits

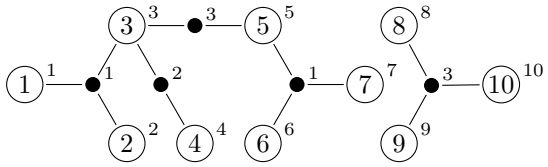



**Figure 1** The colored cycle graph $\mathrm{CG}_{\mathrm{id}}\big((1,2,3)(5,6,7),(3,4),(3,5)(8,9,10)\big)$; the colors are depicted next to the vertices.

of size at least 2, i.e., the cycles of $\sigma_i$, for $1 \leq i \leq \ell$. We call the former *primal vertices* and the latter *cycle-vertices*. Each primal vertex $v \in V$ is colored by $c(v)$, and each cycle-vertex that corresponds to a cycle of $\sigma_i$ is colored by $i$. (Note that a vertex of this graph is a cycle-vertex if and only if it has odd distance to some leaf.) See Figure 1 for an example.

▶ **Lemma 5.1.** *Let $\sigma \in \mathrm{Sym}(V)$, let $\sigma = \sigma_1 \cdots \sigma_\ell$ be a complexity-additive decomposition, and let $c\colon V \to [k]$ be a coloring. Then $\mathrm{CG}_c(\sigma_1, \ldots, \sigma_\ell)$ is a forest.*

A *cycle pattern* $P$ is a colored cycle graph $\mathrm{CG}_c(\sigma_1, \ldots, \sigma_\ell)$ where all primal vertices have different colors. A complexity-additive decomposition $\sigma' = \sigma'_1 \cdots \sigma'_\ell$ of a permutation $\sigma' \in \mathrm{Sym}(V)$ *weakly matches* $P$ if there is a coloring $c'\colon V \to [k]$ and a surjective color-preserving homomorphism $\varphi$ from $\mathrm{CG}_{c'}(\sigma'_1, \ldots, \sigma'_\ell)$ to $P$ where $\varphi(u) = \varphi(v)$ for $u \neq v$ implies that $u$ and $v$ are both primal vertices and belong to different $\sigma'$-orbits.

▶ **Lemma 5.2.** *For any $t \in \mathbb{N}$, there is a set $\mathcal{P}_t$ of $t^{\mathcal{O}(t)}$ cycle patterns such that a permutation $\sigma \in \mathrm{Sym}(V)$ has complexity $t$ if and only if it has a complexity-additive decomposition $\sigma = \sigma_1 \cdots \sigma_\ell$ that weakly matches a pattern in $\mathcal{P}_t$. Moreover, $\mathcal{P}_t$ can be computed in $t^{\mathcal{O}(t)}$ time.*

For a pattern $P$, let $P_i$ denote the subgraph of $P$ induced by the cycle-vertices of color $i$ and their neighbors. A permutation $\sigma \in \mathrm{Sym}(V)$ and a coloring $c\colon V \to [k]$ *realize color $i$ of $P$* if there is an isomorphism $\varphi$ from $\mathrm{CG}_c(\sigma)$ to $P_i$ that preserves colors of primal vertices.

**Algorithm 4** $\texttt{Exact-Complexity-HGI}_d(X, Y, t)$

**Input**: Two hypergraphs $X$ and $Y$ on vertex set $V$ with hyperedge size bounded by $d$, and $t \in \mathbb{N}$
**Output**: An isomorphism $\sigma$ from $X$ to $Y$ with $\mathrm{compl}(\sigma) = t$, or $\perp$ if none exists
$A \leftarrow \big\{\sigma \in \mathrm{Aut}(X) \,\big|\, \sigma \text{ has minimal complexity in } \mathrm{Aut}(X) \text{ and } |\mathrm{supp}(\sigma)| \leq 2t\big\}$
  *// see [1, Algorithm 3]*
**if** $X = Y$ **then** $I \leftarrow A$ **else**
    $I \leftarrow \big\{\sigma \in \mathrm{Iso}(X, Y) \,\big|\, \sigma \text{ has minimal complexity in } \mathrm{Iso}(X, Y) \text{ and } |\mathrm{supp}(\sigma)| \leq 2t\big\}$
      *// see [1, Algorithm 2]*
**for** $P \in \mathcal{P}_t$ **do** *// see Lemma 5.2*
    $k \leftarrow$ the number of primal vertices in $P$
    $\ell \leftarrow$ the number of colors of cycle-vertices in $P$
    **for** $h \in \mathcal{H}_{V,k}$ **do** *// $\mathcal{H}_{V,k}$ is the perfect family of hash functions $h\colon V \to [k]$ from [6]*
        **if** there are $\sigma_1, \ldots, \sigma_{\ell-1} \in A$ and $\sigma_\ell \in I$ s.t. $(\sigma_i, h)$ realize color $i$ of $P$ **then**
            **return** $\sigma = \sigma_1 \cdots \sigma_\ell$
**return** $\perp$

▶ **Theorem 5.3.** *Given two hypergraphs $X$ and $Y$ of hyperedge size at most $d$ and $t \in \mathbb{N}$, the algorithm $\texttt{Exact-Complexity-HGI}_d(X, Y, t)$ finds $\sigma \in \mathrm{Iso}(X, Y)$ with $\mathrm{compl}(\sigma) = t$ (or determines that there is none) in $\mathcal{O}\big((dt)^{\mathcal{O}(t^2)}\,\mathrm{poly}(N)\big)$ time.*

**Proof.** Suppose there is some $\sigma \in \mathrm{Iso}(X,Y)$ with $\mathrm{compl}(\sigma) = t$. Lemma 2.1 gives the complexity-additive decomposition $\sigma = \sigma_1 \cdots \sigma_\ell$ into minimal-complexity permutations $\sigma_1, \ldots, \sigma_{\ell-1} \in \mathrm{Aut}(X)$ and $\sigma_\ell \in \mathrm{Iso}(X,Y)$; all of them have complexity at most $t$. By the correctness of the algorithms from [1], we have $\sigma_1, \ldots, \sigma_{\ell-1} \in A$ and $\sigma_\ell \in I$. As $\mathcal{H}_{V,k}$ is a perfect hash family, it contains some function $h$ whose restriction to $\mathrm{supp}(\sigma)$ is injective. Then $\mathrm{CG}_h(\sigma_1, \ldots, \sigma_\ell)$ is isomorphic to some $P \in \mathcal{P}_t$ by Lemma 5.2. Thus $(\sigma_i, h)$ realize color $i$ of $P$, for $1 \leq i \leq \ell$, so the algorithm does not return $\perp$.

Now suppose that the algorithm returns $\sigma = \sigma_1 \cdots \sigma_\ell$ with $\sigma_1, \ldots, \sigma_{\ell-1} \in A \subseteq \mathrm{Aut}(X)$ and $\sigma_\ell \in I \subseteq \mathrm{Iso}(X,Y)$. This clearly implies $\sigma \in \mathrm{Iso}(X,Y)$. To show $\mathrm{compl}(\sigma) = t$, we observe that the algorithm only returns $\sigma$ if there is a pattern $P \in \mathcal{P}_t$ whose cycle-vertices have $\ell$ colors and which contains $k$ primal vertices such that there is a hash function $h \in \mathcal{H}_{V,k}$ with the property that $\sigma_i$ and $h$ realize color $i$ of $P$, for $i \in [\ell]$. In particular, there is an isomorphism $\varphi_i$ from $\mathrm{CG}_h(\sigma_i)$ to $P_i$ that preserves colors of primal vertices. As the primal vertices of $P$ all have different colors and as $P$ is a forest by Lemma 5.1, it follows that the decomposition $\sigma = \sigma_1 \cdots \sigma_\ell$ is complexity-additive. Now consider the function $\varphi = \bigcup_{i=1}^{\ell} \varphi_i$; it is well-defined, as $v \in \mathrm{supp}(\varphi_i) \cap \mathrm{supp}(\varphi_j)$ implies $\varphi_i(v) = \varphi_j(v)$ because $P$ contains only one primal vertex of color $h(v)$. It is surjective, as every vertex of $P$ occurs in at least one $P_i$. It is a homomorphism from $P_\sigma = \mathrm{CG}_h(\sigma_1, \ldots, \sigma_\ell)$ to $P$, as every edge occurs in the support of one of the isomorphisms $\varphi_i$. Also, $\varphi(u) = \varphi(v)$ for $u \neq v$ implies that $u$ and $v$ are in different connected components of $P_\sigma$, as $P$ is a forest; consequently $u$ and $v$ are in different orbits of $\sigma$. Thus $\sigma = \sigma_1 \cdots \sigma_\ell$ weakly matches $P$. By Lemma 5.2 it follows that $\mathrm{compl}(\sigma) = t$.

It remains to analyze the runtime. The algorithms used to compute $A$ and $I$ each take $\mathcal{O}\big((dt)^{\mathcal{O}(t^2)} \mathrm{poly}(N)\big)$ time [1]. The pattern set $\mathcal{P}_t$ can be computed in $t^{\mathcal{O}(t)}$ time by Lemma 5.2. As $k \leq 2t$, the perfect hash family $\mathcal{H}_{V,k}$ has size $2^{\mathcal{O}(t)} \log^2 n$. As $\ell \leq t$, this gives a total runtime of $\mathcal{O}\big((dt)^{\mathcal{O}(t^2)} \mathrm{poly}(N)\big)$. ◀

## 6 Colored Graph Automorphism

In [1] we showed that the following parameterized version of Graph Automorphism is W[1]-hard. It was first defined in [5] and is a generalization of the problem studied by Schweitzer [13].

Col-GA: Given a graph $X$ with its vertex set partitioned as Red∪Blue, and a parameter $k$, decide if there is a partition-preserving automorphism that moves exactly $k$ Blue vertices.

For an automorphism $\pi \in \mathrm{Aut}(X)$, we will refer to the number of Blue vertices moved by $\pi$ as the Blue weight of $\pi$. In this section, we show that Col-GA is in FPT when restricted to colored graphs where the color classes inside Red have size at most 3.

Given an input instance $X = (V, E)$ with vertex partition $V = \mathrm{Red} \cup \mathrm{Blue}$ such that Red is refined into color classes of size at most 3 each, our algorithm proceeds as follows.

**Step 1: color-refinement.** $X$ already comes with a color classification of vertices (Red and Blue, and within Red color classes of size at most 3 each; within Blue there may be color classes of arbitrary size). The color refinement procedure keeps refining the coloring in steps until no further refinement of the vertex color classes is possible. In a refinement step, if two vertices have identical colors but differently colored neighborhoods (with the multiplicities of colors counted), then these vertices get new different colors.
At the end of this refinement, each color class $C$ induces a regular graph $X[C]$, and each pair $(C, D)$ of color classes induces a semiregular bipartite graph $X[C, D]$.

**Step 2: local complementation.** We complement the graph induced by a color class if this reduces the number of its edges; this does not change the automorphism group of $X$. Similarly, we complement the induced bipartite graph between two color classes if this reduces the number of its edges.

Now each color class within RED induces the empty graph. Similarly, for $b \in \{2,3\}$, the bipartite graph between any two color classes of size $b$ is empty or a perfect matching. (Note that this does not necessarily hold for $b \geq 4$.) Color refinement for graphs of color class size at most 3 has been used in earlier work [8, 9].

Let $C \subseteq$ RED and $D \subseteq$ BLUE be color classes after Step 1. Because of the complementations we have applied, $|C| = 1$ implies that $X[C,D]$ is empty, and if $|C| \in \{2,3\}$ then $X[C,D]$ is either empty or the degree of each $D$-vertex in $X[C,D]$ is 1.

**Step 3: fix vertices that cannot move.** For a color class $C \subseteq$ RED whose elements have more than $k$ BLUE neighbors, give different new colors to each vertex in $C$ (because of Step 2, each non-isolated RED vertex is in a color class with more than one vertex). Afterwards, rerun Steps 1 and 2 so we again have a stable coloring.

Fixing the vertices in $C$ does not lose any automorphism of $X$ that has BLUE-weight at most $k$. Indeed, as every BLUE vertex has at most one neighbor in $C$, any automorphism that moves some $v \in C$ has to move all (more than $k$) BLUE neighbors of $v$.

**Step 4: remove edges in the red part.** We already observed that each color class in RED induces the empty graph. Let $\mathcal{X}$ be the graph whose vertices are the color classes in RED, where two of them are adjacent iff there is a perfect matching between them in $X$. For each $b \in \{1,2,3\}$, the color classes in RED of size $b$ get partitioned into components of $\mathcal{X}$. We consider each connected component $\mathcal{C}$ of $\mathcal{X}$ that consists of more than one color class. Let $X'$ be the subgraph of $X$ induced by vertices in $\bigcup \mathcal{C}$ and their neighbors in BLUE. Because of Step 3, the graph $X'$ has color class size at most $3k$, so we can compute its automorphism group $H = \mathrm{Aut}(X')$ in $2^{\mathcal{O}(k^2)}\,\mathrm{poly}(N)$ time [7]. We distinguish several cases based on the action of $H$ on an arbitrary color class $C \in \mathcal{C}$:

**Case 1:** If $H(C)$ is not transitive, we split the color class $C$ into the orbits of $H(C)$ and start over with Step 1.

**Case 2:** If $H(C) = \mathrm{Sym}(C)$, we drop all vertices in $\left(\bigcup \mathcal{C}\right) \setminus C$ from $X$. And for each color class $D$ within BLUE that has neighbors in at least one $C' \in \mathcal{C}$, we replace the edges between a vertex $u \in D$ and $\bigcup \mathcal{C}$ by the single edge $(u,v)$, where $v$ is the vertex in $C$ that is reachable via the matching edges from the neighbor of $u$ in $C'$.

**Case 3:** If $H(C)$ is generated by a 3-cycle $(v_1 v_2 v_3)$, we first proceed as in Case 2. Additionally, we add directed edges within each color class $D$ within BLUE that now has neighbors in $C$. Let $D_i \subseteq D$ be the neighbors of $v_i$. We add *directed edges* from all vertices in $D_i$ to all vertices in $D_{(i+1) \bmod 3}$ and color these directed edges by $C$.

After this step, there are no edges induced on the RED part of $X$. Moreover, we have not changed the automorphisms on the induced subgraph, so the modified graph $X$ still has the same automorphism group as before.

**Step 5: turn red vertices into hyperedges.** We encode $X$ as a hypergraph $X' = (\text{BLUE} \cup \text{NEW}, E')$ in which each vertex in RED is encoded as a hyperedge on the vertex set BLUE $\cup$ NEW. Let NEW $= \{v_C \mid C \subseteq \text{RED is a color class}\}$. Let $v \in C \subseteq$ RED be any red vertex. We encode $v$ as the hyperedge $e_v = \{v_C\} \cup \{u \in \text{BLUE} \mid (v,u) \in E(X)\}$.

In the hypergraph $X'$ we give distinct colors to each vertex in NEW in order to ensure that each color class $\{v_{C,1}, v_{C,2}, v_{C,3}\}$ in RED is preserved by the automorphisms of $X'$. Clearly, there is a 1-1 correspondence between the color-preserving automorphisms of $X$ and those of $X'$. Note that the hyperedges of $X'$ have size bounded by $k+1$, as each RED vertex in $X$ has at most $k$ BLUE neighbors after Step 3.

**Step 6: bounded hyperedge size automorphism.** We seek a weight $k$ automorphism of $X'$ using the algorithm of [1, Corollary 6.4];[1] this is possible in $d^{\mathcal{O}(k)}2^{\mathcal{O}(k^2)}\,\mathrm{poly}(N)$ time.

This algorithm gives us the following.

▶ **Theorem 6.1.** *The above algorithm solves* Col-GA *when the* Red *part of the input graph is refined into color classes of size at most 3. It runs in* $d^{\mathcal{O}(k)}2^{\mathcal{O}(k^2)}\,\mathrm{poly}(N)$ *time.*

**Acknowledgements.** We thank the anonymous referees for their valuable comments.

## References


1 Vikraman Arvind, Johannes Köbler, Sebastian Kuhnert, and Jacobo Torán. Parameterized complexity of small weight automorphisms. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 7:1–7:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.7.

2 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.

3 László Babai, D. Yu. Grigoryev, and David M. Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 310–324. ACM, 1982. doi:10.1145/800070.802206.

4 John D. Dixon and Brian Mortimer. *Permutation groups*. Springer, 1996. doi:10.1007/978-1-4612-0731-3.

5 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

6 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.

7 Merrick L. Furst, John E. Hopcroft, and Eugene M. Luks. Polynomial-time algorithms for permutation groups. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 36–41. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.34.

8 Neil Immerman and Eric Lander. *Describing Graphs: A First-Order Approach to Graph Canonization*, pages 59–81. Springer, 1990. doi:10.1007/978-1-4612-4478-3_5.

9 Birgit Jenner, Johannes Köbler, Pierre McKenzie, and Jacobo Torán. Completeness results for graph isomorphism. *J. Comput. Syst. Sci.*, 66(3):549–566, 2003. doi:10.1016/S0022-0000(03)00042-4.

10 Pavel Klavík, Dušan Knop, and Peter Zeman. Graph Isomorphism restricted by lists, 2016. URL: https://arxiv.org/abs/1607.03918.

11 Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. *SIAM J. Comput.*, 46(1):161–189, 2017. doi:10.1137/140999980.

12 Anna Lubiw. Some np-complete problems similar to graph isomorphism. *SIAM J. Comput.*, 10(1):11–21, 1981. doi:10.1137/0210002.


[1] There is a caveat that in addition to hyperedges in the graph $X'$[Blue] we also have colored directed edges. However, the algorithm of [1, Corollary 6.4] needs only minor changes to handle this.

**13** Pascal Schweitzer. Isomorphism of (mis)labeled graphs. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, volume 6942 of *Lecture Notes in Computer Science*, pages 370–381. Springer, 2011. doi:10.1007/978-3-642-23719-5_32.

# Parameterized Complexity of Finding a Spanning Tree with Minimum Reload Cost Diameter*†

**Julien Baste[1], Didem Gözüpek[2], Christophe Paul[3], Ignasi Sau[4], Mordechai Shalom[5], and Dimitrios M. Thilikos[6]**

1 **Université de Montpellier, LIRMM, Montpellier, France**
baste@lirmm.fr
2 **Department of Computer Engineering, Gebze Technical University, Kocaeli, Turkey**
didem.gozupek@gtu.edu.tr
3 **AlGCo project team, CNRS, LIRMM, France**
paul@lirmm.fr
4 **Departamento de Matemática, Universidade Federal do Ceará, Fortaleza, Brazil and AlGCo project team, CNRS, LIRMM, France**
sau@lirmm.fr
5 **TelHai College, Upper Galilee, Israel and Department of Industrial Engineering, Boğaziçi University, Istanbul, Turkey**
cmshalom@telhai.ac.il
6 **Department of Mathematics, National and Kapodistrian University of Athens, Greece and AlGCo project team, CNRS, LIRMM, France**
sedthilk@thilikos.info

## Abstract

We study the minimum diameter spanning tree problem under the reload cost model (Diameter-Tree for short) introduced by Wirth and Steffan (2001). In this problem, given an undirected edge-colored graph $G$, reload costs on a path arise at a node where the path uses consecutive edges of different colors. The objective is to find a spanning tree of $G$ of minimum diameter with respect to the reload costs. We initiate a systematic study of the parameterized complexity of the Diameter-Tree problem by considering the following parameters: the cost of a solution, and the treewidth and the maximum degree $\Delta$ of the input graph. We prove that Diameter-Tree is para-NP-hard for any combination of two of these three parameters, and that it is FPT parameterized by the three of them. We also prove that the problem can be solved in polynomial time on cactus graphs. This result is somehow surprising since we prove Diameter-Tree to be NP-hard on graphs of treewidth two, which is best possible as the problem can be trivially solved on forests. When the reload costs satisfy the triangle inequality, Wirth and Steffan (2001) proved that the problem can be solved in polynomial time on graphs with $\Delta = 3$, and Galbiati (2008) proved that it is NP-hard if $\Delta = 4$. Our results show, in particular, that without the requirement of the triangle inequality, the problem is NP-hard if $\Delta = 3$, which is also best possible. Finally, in the case where the reload costs are polynomially bounded by the size of the input graph, we prove that Diameter-Tree is in XP and W[1]-hard parameterized by the treewidth plus $\Delta$.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, C.2.1 Network Architecture and Design, G.2.2 Graph Theory

* This work has been supported by the bilateral research program of CNRS and TUBITAK under grant no.114E731, PASTA project of Université de Montpellier, TUBITAK 2221 programme, and by project DEMOGRAPH (ANR-16-CE40-0028).

† A full version of this article is permanently available at https://arxiv.org/abs/1703.01686.



12th International Symposium on Parameterized and Exact Computation (IPEC 2017).
Editors: Daniel Lokshtanov and Naomi Nishimura; Article No. 3; pp. 3:1–3:12

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

**Keywords and phrases** reload cost problems, minimum diameter spanning tree, parameterized complexity, FPT algorithm, treewidth, dynamic programming

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.3

## 1 Introduction

Numerous network optimization problems can be modeled by edge-colored graphs. Wirth and Steffan introduced in [28] the concept of *reload cost*, which refers to the cost that arises in an edge-colored graph while traversing a vertex via two consecutive edges of different colors. The value of the reload cost depends on the colors of the traversed edges. Although the reload cost concept has many important applications in telecommunication networks, transportation networks, and energy distribution networks, it has surprisingly received attention only recently.

In heterogeneous communication networks, routing requires switching among different technologies such as cables, fibers, and satellite links. Due to data conversion between incompatible subnetworks, this switching causes high costs, largely outweighing the cost of routing the packets within each subnetwork. The recently popular concept of vertical handover [9], which allows a mobile user to have undisrupted connection during transitioning between different technologies such as 3G (third generation) and wireless local area network (WLAN), constitutes another application area of the reload cost concept. Even within the same technology, switching between different service providers incurs switching costs. Another paradigm that has received significant attention in the wireless networks research community is *cognitive radio networks* (CRN), a.k.a. *dynamic spectrum access networks*. Unlike traditional wireless technologies, CRNs operate across a wide frequency range in the spectrum and frequently requires frequency switching; therefore, the frequency switching cost is indispensable and of paramount importance. Many works in the CRNs literature focused on this frequency switching cost from an application point of view (for instance, see [3, 19, 4, 5, 11, 1, 26]) by analyzing its various aspects such as delay and energy consumption. Operating in a wide range of frequencies is indeed a property of not only CRNs but also other 5G technologies. Hence, applications of the reload cost concept in communication networks continuously increase. In particular, the energy consumption aspect of this switching cost is especially important in the recently active research area of green networks, which aim to tackle the increasing energy consumption of information and communication technologies [6, 8].

The concept of reload cost also finds applications in other networks such as transportation networks and energy distribution networks. For instance, a cargo transportation network uses different means of transportation. The loading and unloading of cargo at junction points is costly and this cost may even outweigh the cost of carrying the cargo from one point to another [12]. In energy distribution networks, reload costs can model the energy losses that occur at the interfaces while transferring energy from one type of carrier to another [12].

Recent works in the literature focused on numerous problems related to the reload cost concept: the minimum reload cost cycle cover problem [14], the problems of finding a path, trail or walk with minimum total reload cost between two given vertices [17], the problem of finding a spanning tree that minimizes the sum of reload costs of all paths between all pairs of vertices [15], various path, tour, and flow problems related to reload costs [2], the minimum changeover cost arborescence problem [13, 22, 20, 18], and problems related to finding a proper edge coloring of the graph so that the total reload cost is minimized [21].

The work in [28], which introduced the concept of reload cost, focused on the following problem, called Minimum Reload Cost Diameter Spanning Tree (Diameter-Tree for short), and which is the one we study in this paper: given an undirected graph $G = (V, E)$ with a (non necessarily proper) edge-coloring $\chi : E(G) \to X$ and a reload cost function $c : X^2 \to \mathbb{N}_0$, find a spanning tree of $G$ with minimum diameter with respect to the reload costs (see Section 2 for the formal definitions).

This problem has important applications in communication networks, since forming a spanning tree is crucial for broadcasting control traffic such as route update messages. For instance, in a multi-hop cognitive radio network where a frequency is assigned to each wireless link depending on availabilities of spectrum bands, delay-aware broadcasting of control traffic necessitates the forming of a spanning tree by taking the delay arising from frequency switching at every node into account. Cognitive nodes send various control information messages to each other over this spanning tree. A spanning tree with minimum reload cost diameter in this setting corresponds to a spanning tree in which the maximum frequency switching delay between any two nodes on the tree is minimized. Since control information is crucial and needs to be sent to all other nodes in a timely manner, ensuring that the maximum delay is minimum is vital in a cognitive radio network.

Wirth and Steffan [28] proved that Diameter-Tree is inapproximable within a factor better than 3 (in particular, it is NP-hard), even on graphs with maximum degree 5. They also provided a polynomial-time exact algorithm for the special case where the maximum degree is 3 and the reload costs satisfy the triangle inequality. Galbiati [12] showed stronger hardness results for this problem, by proving that even on graphs with maximum degree 4, the problem cannot be approximated within a factor better than 2 if the reload costs do not satisfy the triangle inequality, and cannot be approximated within any factor better than 5/3 if the reload costs satisfy the triangle inequality. The complexity of Diameter-Tree (in the general case) on graphs with maximum degree 3 was left open.

**Our results.** In this article we initiate a systematic study of the complexity of the Diameter-Tree problem, with special emphasis on its parameterized complexity for several choices of the parameters. Namely, we consider any combination of the parameters $k$ (the cost of a solution), tw (the treewidth of the input graph), and $\Delta$ (the maximum degree of the input graph). We would like to note that these parameters have practical importance in communication networks. Indeed, besides the natural parameter $k$, whose relevance is clear, many networks that model real-life situations appear to have small treewidth [24]. On the other hand, the degree of a node in a network is related to its number of transceivers, which are costly devices in many different types of networks such as optical networks [25]. For this reason, in practice the maximum degree of a network usually takes small values.

Before elaborating on our results, a summary of them can be found in Table 1.

We first prove, by a reduction from 3-Sat, that Diameter-Tree is NP-hard on outerplanar graphs (which have treewidth at most 2) with only one vertex of degree greater than 3, even with three different costs that satisfy the triangle inequality, and $k = 9$. Note that, in the case where the costs satisfy the triangle inequality, having only one vertex of degree greater than 3 is best possible, as if all vertices have degree at most 3, the problem can be solved in polynomial time [28]. Note also that the bound on the treewidth is best possible as well, since the problem is trivially solvable on graphs of treewidth 1, i.e., on forests.

Toward investigating the border of tractability of the problem with respect to treewidth, we exhibit a polynomial-time algorithm on a relevant subclass of the graphs of treewidth at most 2: cactus graphs. This algorithm is quite involved and, in a nutshell, processes in

**Table 1** Summary of our results, where $k, \mathsf{tw}, \Delta$ denote the cost of the solution, the treewidth, and the maximum degree of the input graph, respectively. NPh stands for NP-hard. The symbol '✓' denotes that the result above still holds for polynomial costs.

| Problem | Parameterized complexity with parameter | | | | Polynomial cases |
|---|---|---|---|---|---|
| | $k + \mathsf{tw}$ | $k + \Delta$ | $\mathsf{tw} + \Delta$ | $k + \mathsf{tw} + \Delta$ | |
| Diameter-Tree | NPh for $k = 9, \mathsf{tw} = 2$ (Thm 1) | NPh for $k = 0, \Delta = 3$ (Thm 2) | NPh for $\mathsf{tw} = 3, \Delta = 3$ (Thm 4) | FPT (Thm 7) | in P on cacti (Thm 6) |
| Diameter-Tree with poly costs | ✓ | ✓ | XP (Thm 7) W[1]-hard (Thm 9) | ✓ | ✓ |

a bottom-up manner the *block tree* of the given cactus graph, and uses at each step of the processing an algorithm that solves 2-Sat as a subroutine.

Back to hardness results, we also prove, by a reduction from a restricted version of 3-Sat, that Diameter-Tree is NP-hard on graphs with $\Delta \leq 3$, even with only two different costs, $k = 0$, and a bounded number of colors. In particular, this settles the complexity of the problem on graphs with $\Delta \leq 3$ in the general case where the triangle inequality is not necessarily satisfied, which had been left open in previous work [28, 12]. Note that $\Delta \leq 3$ is best possible, as Diameter-Tree can be easily solved on graphs with $\Delta \leq 2$.

As our last NP-hardness reduction, we prove, by a reduction from Partition, that the Diameter-Tree problem is NP-hard on planar graphs with $\mathsf{tw} \leq 3$ and $\Delta \leq 3$.

The above hardness results imply that the Diameter-Tree problem is para-NP-hard for any combination of two of the three parameters $k$, tw, and $\Delta$. On the positive side, we show that Diameter-Tree is FPT parameterized by the three of them, by using a (highly nontrivial) dynamic programming algorithm on a tree decomposition of the input graph.

Since our para-NP-hardness reduction with parameter $\mathsf{tw} + \Delta$ is from Partition, which is a typical example of a *weakly* NP-complete problem [16], a natural question is whether Diameter-Tree, with parameter $\mathsf{tw} + \Delta$, is para-NP-hard, XP, W[1]-hard, or FPT when the reload costs are *polynomially bounded* by the size of the input graph. We manage to answer this question completely: we show that in this case the problem is in XP (hence *not* para-NP-hard) and W[1]-hard parameterized by $\mathsf{tw} + \Delta$. The W[1]-hardness reduction is from the Unary Bin Packing problem parameterized by the number of bins, proved to be W[1]-hard by Jansen *et al.* [23].

Altogether, our results provide an accurate picture of the (parameterized) complexity of the Diameter-Tree problem.

**Further research.** In the hardness result of Theorem 4, the bound $\Delta \leq 3$ is tight, but the bound $\mathsf{tw} \leq 3$ might be improved to $\mathsf{tw} \leq 2$. A relevant question is whether the problem admits polynomial kernels parameterized by $k + \mathsf{tw} + \Delta$ (recall that it is FPT by Theorem 7). Theorem 9 motivates the following question: when all reload costs are bounded by a *constant*, is the Diameter-Tree problem FPT parameterized by $\mathsf{tw} + \Delta$? It also makes sense to consider the *color-degree* as a parameter (cf. [20]). Finally, we may consider other relevant width parameters, such as pathwidth (note that the hardness results of Theorems 1, 4, and 9 also hold for pathwidth), cliquewidth, treedepth, or tree-cutwidth.

**Organization of the paper.** We start in Section 2 with some preliminaries about the Diameter-Tree problem. Basic definitions about graphs, parameterized complexity, and tree decompositions can be found in the full version. In Section 3 we provide the para-NP-hardness results, and in Section 4 we present the polynomial-time algorithm on cactus graphs and the FPT algorithm on general graphs parameterized by $k + \mathsf{tw} + \Delta$. In Section 5 we focus on the case where the reload costs are polynomially bounded. Due to lack of space, the proof of the results marked with '[⋆]' can be found in the full version.

## 2 Reload costs and definition of the problem

For reload costs, we follow the notation and terminology defined by [28]. We consider edge-colored graphs $G = (V, E)$, where the colors are taken from a finite set $X$ and the coloring function is $\chi : E(G) \to X$. The reload costs are given by a nonnegative function $c : X^2 \to \mathbb{N}_0$, which we assume for simplicity to be symmetric. The cost of traversing two incident edges $e_1$, $e_2$ is $c(e_1, e_2) := c(\chi(e_1), \chi(e_2))$. By definition, reload costs at the endpoints of a path equal zero. Consequently, the reload cost of a path with one edge also equals zero. The *reload cost* of a path $P$ of length $\ell \geq 2$ with edges $e_1, e_2, \ldots, e_\ell$ is defined as $c(P) := \sum_{i=2}^{\ell} c(e_{i-1}, e_i)$. The induced reload cost distance function is given by $\mathsf{dist}^c_G(u, v) = \min\{c(P) \mid P \text{ is a path from } u \text{ to } v \text{ in } G\}$. The *diameter* of a tree $T$ is $\mathsf{diam}(T) := \max_{u,v \in V} \mathsf{dist}^c_T(u, v)$, where for notational convenience we assume that the edge-coloring function $\chi$ and the reload cost function $c$ are clear from the context.

The problem we study in this paper can be formally defined as follows:

Minimum Reload Cost Diameter Spanning Tree (Diameter-Tree)
**Input:** A graph $G = (V, E)$ with an edge-coloring $\chi$ and a reload cost function $c$.
**Output:** A spanning tree $T$ of $G$ minimizing $\mathsf{diam}(T)$.

If for every three distinct edges $e_1, e_2, e_3$ of $G$ incident to the same node, it holds that $c(e_1, e_3) \leq c(e_1, e_2) + c(e_2, e_3)$, we say that the reload cost function $c$ satisfies the *triangle inequality*. This assumption is sometimes used in practical applications [28].

Throughout the paper, we let $n$, $\Delta$, and $\mathsf{tw}$ denote the number of vertices, the maximum degree, and the treewidth of the input graph, respectively. When we consider the (parameterized) decision version of the Diameter-Tree problem, we also let $k$ denote the desired cost of a solution.

## 3 Para-NP-hardness results

We start with the para-NP-hardness result with parameter $k + \mathsf{tw}$.

▶ **Theorem 1.** *The* Diameter-Tree *problem is* NP*-hard on outerplanar graphs with only one vertex of degree greater than 3, even with three different costs that satisfy the triangle inequality, and* $k = 9$*. Since outerplanar graphs have treewidth at most 2, in particular,* Diameter-Tree *is* para-NP*-hard parameterized by* tw *and* $k$*.*

**Proof.** We present a simple reduction from 3-Sat. Given a formula $\varphi$ with $n$ variables and $m$ clauses, we create an instance $(G, \chi, c)$ of Diameter-Tree as follows. We may assume that there is no clause in $\varphi$ that contains a literal and its negation. The graph $G$ contains a distinguished vertex $r$ and, for each clause $c_j = (\ell_1 \vee \ell_2 \vee \ell_3)$, we add a clause gadget $C_j$ consisting of three vertices $v^j_{\ell_1}, v^j_{\ell_2}, v^j_{\ell_3}$ and five edges $\{r, v^j_{\ell_1}\}$, $\{r, v^j_{\ell_2}\}$, $\{r, v^j_{\ell_3}\}$, $\{v^j_{\ell_1}, v^j_{\ell_2}\}$, and $\{v^j_{\ell_2}, v^j_{\ell_3}\}$. This completes the construction of $G$. Note that $G$ does not depend on the

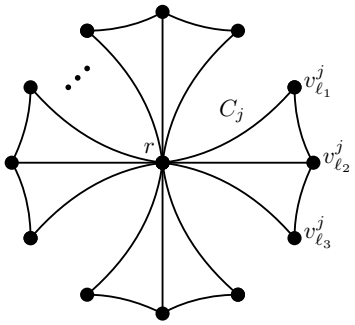



**Figure 1** Example of the graph $G$ built in the reduction of Theorem 1.

formula $\varphi$ except for the number of clause gadgets, and that it is an outerplanar graph with only one vertex of degree greater than 3; see Figure 1 for an illustration.

Let us now define the coloring $\chi$ and the cost function $c$. For simplicity, we associate a distinct color with each edge of $G$, and thus, with slight abuse of notation, it is enough to describe the cost function $c$ for every pair of incident edges of $G$, as we consider symmetric cost functions. We will use just three different costs: 1, 5 and 10. We set

$$c(e_1, e_2) = \begin{cases} 10 & \text{if } e_1 = \{r, v^{j_1}_{\ell_{i_1}}\}, e_2 = \{r, v^{j_2}_{\ell_{i_2}}\} \text{ and } \ell_{i_1} = \overline{\ell_{i_2}}\ , \\ 5 & \text{if } e_1 = \{r, v^{j_1}_{\ell_{i_1}}\}, e_2 = \{r, v^{j_2}_{\ell_{i_2}}\} \text{ and } \ell_{i_1} \neq \overline{\ell_{i_2}}\ , \text{ and} \\ 1 & \text{otherwise.} \end{cases}$$

Note that this cost function satisfies the triangle inequality since the reload costs between edges incident to $r$ are 5 and 10, and the reload costs between edges incident to other vertices are 1.

We claim that $\varphi$ is satisfiable if and only if $G$ contains a spanning tree with diameter at most 9. Since $r$ is a cut vertex and every clause gadget is a connected component of $G - r$, in every spanning tree, the vertices of $C_j$ together with $r$ induce a tree with four vertices. Moreover the reload cost associated with a path from $r$ to a leaf of this tree is always at most 2. Therefore, the diameter of any spanning tree is at most 4 plus the maximum reload cost incurred at $r$ by a path of $T$.

Assume first that $\varphi$ is satisfiable, fix a satisfying assignment $\psi$ of $\varphi$, and let us construct a spanning tree $T$ of $G$ with diameter at most 9. For each clause $c_j$, the tree $T^j$ is the tree spanning $C_j$ and containing the edge between $r$ and an arbitrarily chosen literal of $c_j$ that is set to true by $\psi$. $T$ is the union of all the trees $T_j$ constructed in this way. The reload cost incurred at $r$ by any path of $T$ traversing it is at most 5, since we never choose a literal and its negation. Therefore, it holds that $\mathsf{diam}(T) \leq 9$.

Conversely, let $T$ be a spanning tree of $G$ with $\mathsf{diam}(T) \leq 9$. Then, the reload cost incurred at $r$ by any path traversing it is at most 5 since otherwise $\mathsf{diam}(T) \geq 10$. For every $j \in [m]$, let $T_j$ be the subtree of $T$ induced by $C_j$ and let $\{r, v^j_{\ell_{i_j}}\}$ be one of the edges incident to $r$ in $T_j$. We note that for any pair of clauses $c_{j_1}, c_{j_2}$ we have $\ell_{i_{j_1}} \neq \overline{\ell_{i_{j_2}}}$, since otherwise a path using these two edges would incur a cost of 10 at $r$. The variable in the literal $\ell_{i_j}$ is set by $\psi$ so that $\ell_{i_j}$ is true. All the other variables are set to an arbitrary value by $\psi$. Note that $\psi$ is well-defined, since we never encounter a literal and its negation during the assignment process. It follows that $\psi$ is a satisfying assignment of $\varphi$. ◀

We proceed with the para-NP-hardness result with parameter $k + \Delta$.

▶ **Theorem 2.** *The* Diameter-Tree *problem is* NP*-hard on graphs with* $\Delta \leq 3$*, even with two different costs,* $k = 0$*, and a bounded number of colors. In particular, it is* para-NP*-hard parameterized by* $k$ *and* $\Delta$*.*

**Proof.** We present a reduction from the restriction of 3-Sat to formulas where each variable occurs in at most three clauses; this problem was proved to be NP-complete by Tovey [27]. It is worth mentioning that one needs to allow for clauses of size two or three, as if all clauses have size exactly three, then it turns out that all instances are satisfiable [27].

We may assume that each variable occurs at least once positively and at least once negatively, as otherwise we may set such a variable $x$ to the value that satisfies all clauses in which it appears, and delete $x$ together with those clauses from the formula. We may also assume that each variable occurs *exactly* three times in the given formula $\varphi$. Indeed, let $x$ be a variable occurring exactly two times in the formula. We create a new variable $y$ and we add to $\varphi$ two clauses $(x \vee y)$ and $(y \vee \overline{y})$. Let $\varphi'$ be the new formula. Clearly $\varphi$ and $\varphi'$ are equivalent, and both $x$ and $y$ occur three times in $\varphi'$. Applying these operations exhaustively clearly results in an equivalent formula where each variable occurs exactly three times. Summarizing, we may assume the following property:

✠ *Each variable occurs exactly three times in the given formula* $\varphi$ *of* 3-Sat*. Moreover, each variable occurs at least once positively and at least once negatively in* $\varphi$*.*

Given a formula $\varphi$ with $n$ variables and $m$ clauses, we create an instance $(G, \chi, c)$ of Diameter-Tree with $\Delta(G) \leq 3$ as follows. Let the variables in $\varphi$ be $x_1, \ldots, x_n$. For every $i \in [n]$, we add to $G$ a *variable gadget* consisting of five vertices $u_i, v_i, p_i, r_i, n_i$ and five edges $\{u_i, v_i\}, \{v_i, p_i\}, \{p_i, r_i\}, \{r_i, n_i\}$, and $\{n_i, v_i\}$. For every $i \in [n-1]$, we add the edge $\{u_i, u_{i+1}\}$. For every $j \in [m]$, the clause gadget in $G$ consists of a single vertex $c_j$. We now proceed to explain how we connect the variable and the clause gadgets. For each variable $x_i$, we connect vertex $p_i$ (resp. $n_i$) to one of the vertices corresponding to a clause of $\varphi$ in which $x_i$ appears positively (resp. negatively). Finally, we connect vertex $r_i$ to the remaining clause in which $x_i$ appears (positively or negatively). Note that these connections are well-defined because of property ✠. This completes the construction of $G$, and note that it indeed holds that $\Delta(G) \leq 3$; see Figure 2(a) for an example of the construction of $G$ for a specific satisfiable formula $\varphi$ with $n = 4$ and $m = 5$.

Let us now define the coloring $\chi$ and the cost function $c$. We use nine colors $1, 2, \ldots, 9$ associated with the edges of $G$ as follows. For $i \in [n]$, we set $\chi(\{p_i, r_i\}) = 1$ and $\chi(\{r_i, n_i\}) = 2$, and all edges incident to $u_i$ or $v_i$ have color 3. Finally, for $j \in [m]$, we color the edges containing $c_j$ with colors in $\{4, 5, 6, 7, 8, 9\}$, so that incident edges get different colors, and edges corresponding to positive (resp. negative) occurrences get colors in $\{4, 5, 6\}$ (resp. $\{7, 8, 9\}$); note that such a coloring always exists as each clause contains at most three variables; see Figure 2(b). We will use only two costs, namely 0 and 1, and recall that we consider only symmetric cost functions. We set $c(1, 2) = 1$, $c(1, i) = 1$ for every $i \in \{4, 5, 6\}$, $c(2, i) = 1$ for every $i \in \{7, 8, 9\}$, and $c(i, j) = 1$ for every distinct $4 \leq i, j \leq 9$. All other costs are set to 0. The following claim concludes the proof.

▶ **Claim 3.** [⋆] *$\varphi$ is satisfiable if and only if $G$ contains a spanning tree with diameter 0.* ◀

Note that in the above reduction the cost function $c$ does *not* satisfy the triangle inequality at vertices $p_i$ or $n_i$ for $i \in [n]$, and recall that this is unavoidable since otherwise the problem would be polynomial [28].

Finally, we present the para-NP-hardness result with parameter $\mathsf{tw} + \Delta$.

▶ **Theorem 4.** *The* Diameter-Tree *problem is* NP*-hard on planar graphs with* $\mathsf{tw} \leq 3$ *and* $\Delta \leq 3$*. In particular, it is* para-NP*-hard parameterized by* tw *and* $\Delta$*.*

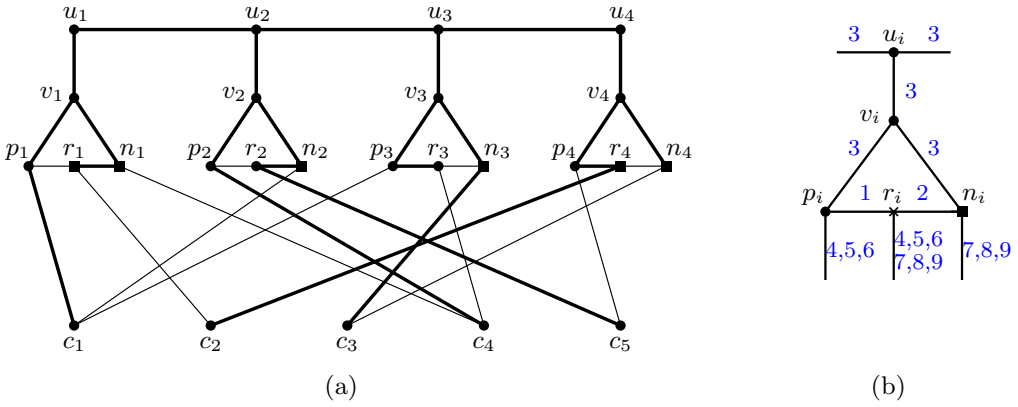



**Figure 2** (a) Graph $G$ described in the reduction of Theorem 2 for the formula $\varphi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_4}) \wedge (\overline{x_3} \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_2 \vee x_4)$. The vertices $p_i, r_i, n_i$ corresponding to positive (resp. negative) occurrences are depicted with circles (resp. squares). An assignment satisfying $\varphi$ is given by $x_1 = x_2 = 1$ and $x_3 = x_4 = 0$, and a solution spanning tree $T$ with diameter 0 is emphasized with thicker edges. (b) The (possible) colors associated with each edge of $G$ are depicted in blue.

**Proof.** We present a reduction from the Partition problem, which is a typical example of a *weakly* NP-complete problem [16]. An instance of Partition is a multiset $S = \{a_1, a_2, \ldots, a_n\}$ of $n$ positive integers, and the objective is to decide whether $S$ can be partitioned into two subsets $S_1$ and $S_2$ such that $\sum_{x \in S_1} x = \sum_{x \in S_2} x = \frac{B}{2}$ where $B = \sum_{x \in S} x$.

Given an instance $S = \{a_1, a_2, \ldots, a_n\}$ of Partition, we create an instance $(G, \chi, c)$ of Diameter-Tree as follows. The graph $G$ contains a vertex $r$, called the root, and for every integer $a_i$ where $i \in [n]$, we add to $G$ six vertices $u_i, u'_i, m_i, m'_i, d_i, d'_i$ and seven edges $\{u_i, u'_i\}$, $\{m_i, m'_i\}$, $\{d_i, d'_i\}$, $\{u_i, m_i\}$, $\{u'_i, m'_i\}$, $\{m_i, d_i\}$, and $\{m'_i, d'_i\}$. We denote by $H_i$ the subgraph induced by these six vertices and seven edges. We add the edges $\{r, u_1\}, \{r, d_1\}$ and, for $i \in [n-1]$, we add the edges $\{u'_i, u_{i+1}\}$ and $\{d'_i, d_{i+1}\}$. Let $G'$ be the graph constructed so far. We then define $G$ to be the graph obtained from two disjoint copies of $G'$ by adding an edge between both roots. Note that $G$ is a planar graph with $\Delta(G) = 3$ and $\mathsf{tw}(G) = 3$. (The claimed bound on the treewidth can be easily seen by building a *path* decomposition of $G$ with consecutive bags of the form $\{u'_{i-1}, d'_{i-1}, u_i, d_i\}, \{u_i, d_i, m_i, u'_i\}, \{d_i, m_i, u'_i, m'_i\}, \{d_i, u'_i, m'_i, d'_i\}, \ldots$.)

Let us now define the coloring $\chi$ and the cost function $c$. Again, for simplicity, we associate a distinct color with each edge of $G$, and thus it is enough to describe the cost function $c$ for every pair of incident edges of $G$. We define the costs for one of the copies of $G'$, and the same costs apply to the other copy. For every edge $e$ being either $\{u'_i, u_{i+1}\}$ or $\{d'_i, d_{i+1}\}$, for $1 \leq i \leq n-1$, we set $c(e, e') = 0$ for each of the four edges $e'$ incident with $e$. For every edge $e = \{m_i, m'_i\}$, for $1 \leq i \leq n$, we set $c(\{u_i, m_i\}, e) = c(\{d_i, m_i\}, e) = a_i$ and $c(e, \{m'_i, u'_i\}) = c(e, \{m'_i, d'_i\}) = 0$. All costs associated with the two edges containing $r$ in one of the copies $G'$ are set to 0. For $e = \{r_1, r_2\}$, where $r_1$ and $r_2$ are the roots of the two copies of $G'$, we set $c(e, e') = 0$ for each of the four edges $e'$ incident to $e$. The cost associated with any other pair of edges of $G$ is equal to $B + 1$; see Figure 3 for an illustration, where (some of) the reload costs are depicted in blue, and a typical solution spanning tree of $G$ is drawn with thicker edges. The following claim concludes the proof.

▶ **Claim 5.** [⋆] *The instance $S$ of* Partition *is a* Yes*-instance if and only if $G$ has a spanning tree with diameter at most $B$.* ◀

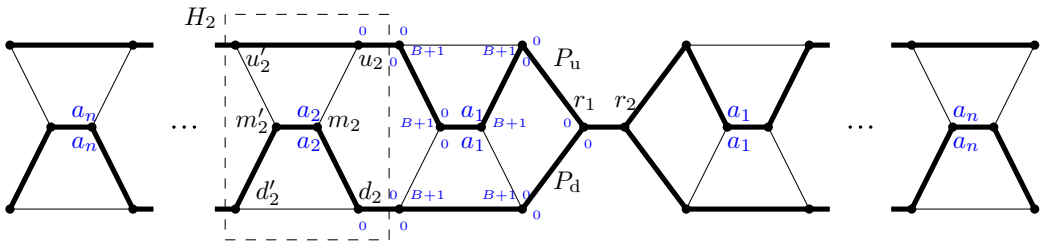



**Figure 3** Graph $G$ built in the reduction of Theorem 4, where the reload costs are depicted in blue at the angle between the two corresponding edges. For better visibility, not all costs and vertex labels are depicted. The typical shape of a solution spanning tree is highlighted with thicker edges.

## 4 Polynomial and FPT algorithms

We start this section by presenting the polynomial-time algorithm to solve the Diameter-Tree problem on cactus graphs, equivalently called *cacti*. We first need some definitions.

A *biconnected component*, or *block*, of a graph is a maximal biconnected induced subgraph of it. The *block tree* of a graph $G$ is a tree $T$ whose nodes are the cut vertices and the blocks of $G$. Every cut vertex is adjacent in $T$ to all the blocks that contain it. Two blocks share at most one vertex. The block tree of a graph is unique and can be computed in polynomial time [10]. A graph is a *cactus* graph if every block of it is either a cycle or a single edge. We term these blocks *cycle blocks* and *edge blocks*, respectively. It is well-known that cacti have treewidth at most 2. Given a forest $F$ and two vertices $x$ and $y$, we define $\texttt{cost}_F(x, y)$ to be $\mathsf{dist}^c_T(x, y)$ if $x$ and $y$ are in the same tree $T$ of $F$ and where $c$ is the given reload cost function, and $\perp$ otherwise. Given a tree $T$ and a vertex $v \in V(T)$, we define the *eccentricity* of $v$ in $T$ to be $\max_{v' \in V(T)} \texttt{cost}_T(v, v')$.

We present a polynomial-time algorithm that solves the decision version of the problem, which we call Diameter-Tree*: the input is an edge-colored graph $G$ and an integer $k$, and the objective is to decide whether the input graph $G$ has a spanning tree with reload cost diameter at most $k$. The algorithm to solve Diameter-Tree* uses dynamic programming on the block tree of the input graph.

As we aim at a *strongly* polynomial-time algorithm to solve Diameter-Tree, we *cannot* afford to solve the decision version for all values of $k$. To overcome this problem, we perform a double *binary search* on the possible solution values and two appropriate eccentricities, resulting (skipping many technical details) in an extra factor of $(\log \mathsf{opt})^2$ in the running time of the algorithm, where $\mathsf{opt}$ is the diameter of a minimum cost spanning tree. This yields a polynomial-time algorithm solving Diameter-Tree in cactus graphs.

Roughly speaking, the algorithm first fixes an arbitrary non-cut vertex $r$ of $G$ and the block $B_r$ that contains it. Then it processes the block tree of $G$ in a bottom-up manner starting from its leaves, proceeding towards $B_r$ while maintaining partial solutions for each block. At each step of the processing, it uses an algorithm that solves an instance of the 2-Sat problem as a subroutine. The intuition behind the instances of 2-Sat created by the algorithm is the following.

Suppose that we are dealing with a cycle block $B$ of the block tree of $G$ (the case of an edge block being easier). Note that any spanning tree of $G$ contains all edges of $B$ except one. Let $G_B$ be the graph processed so far (including $B$). For each potential partial solution $\mathcal{Q}$ in $G_B$, we associate, with each edge $e$ of $B$, a variable that indicates that $e$ is the non-picked edge by the solution in $B$. Now, for any *two* such variables corresponding to intersecting blocks, we add to the formula of 2-Sat essentially two types of clauses: the first set of

clauses, namely $\phi_1$, guarantees that the non-picked edges (corresponding to the variables set to `true` in the eventual assignment) indeed define a spanning tree of $G_B$, while the second one, namely $\phi_2$, forces this solution to have diameter and eccentricity not exceeding the given budget $k$. The fact the $G$ is a cactus allows to prove that these constraints containing only two variables are enough to compute an optimal solution in $G_B$. Full details can be found in the full version.

▶ **Theorem 6.** [⋆] *The* Diameter-Tree *problem can be solved in polynomial time on cacti.*

In the following theorem we prove that the Diameter-Tree problem is FPT on general graphs parameterized by $k$, tw, and $\Delta$. The proof is based on standard, but nontrivial, dynamic programming on graphs of bounded treewidth. It should be mentioned that we can assume that a tree decomposition of the input graph $G$ of width $\mathcal{O}(\mathsf{tw})$ is given together with the input. Indeed, by using for instance the algorithm of Bodlaender *et al.* [7], we can compute in time $2^{\mathcal{O}(\mathsf{tw})} \cdot n$ a tree decomposition of $G$ of width at most $5\mathsf{tw}$. Note that this running time is clearly dominated by the running time stated in Theorem 7.

▶ **Theorem 7.** [⋆] *The* Diameter-Tree *problem can be solved in time* $(k^{\Delta \cdot \mathsf{tw}} \cdot \Delta \cdot \mathsf{tw})^{\mathcal{O}(\mathsf{tw})} \cdot n^{\mathcal{O}(1)}$. *In particular, it is* FPT *parameterized by* $k$, tw, *and* $\Delta$.

## 5 Polynomially bounded costs

So far, we have completely characterized the parameterized complexity of the Diameter-Tree problem for any combination of the three parameters $k$, tw, and $\Delta$. In this section we focus on the special case when the maximum cost value is polynomially bounded by $n$. The following corollary is an immediate consequence of Theorem 7.

▶ **Corollary 8.** *If the maximum cost value is polynomially bounded by* $n$, *the* Diameter-Tree *problem is in* XP *parameterized by* tw *and* $\Delta$.

From Corollary 8, a natural question is whether the Diameter-Tree problem is FPT or W[1]-hard parameterized by tw and $\Delta$, in the case where the maximum cost value is polynomially bounded by $n$. The next theorem provides an answer to this question.

▶ **Theorem 9.** *When the maximum cost value is polynomially bounded by* $n$, *the* Diameter-Tree *problem is* W[1]*-hard parameterized by* tw *and* $\Delta$.

**Proof.** We present a parameterized reduction from the Bin Packing problem parameterized by the number of bins. In Bin Packing, we are given $n$ integer item sizes $a_1, \ldots, a_n$ and an integer capacity $B$, and the objective is to partition the items into a minimum number of bins with capacity $B$. Jansen *et al.* [23] proved that Bin Packing is W[1]-hard parameterized by the number of bins in the solution, even when all item sizes are bounded by a polynomial of the input size. Equivalently, this version of the problem corresponds to the case where the item sizes are given in *unary* encoding; this is why it is called Unary Bin Packing in [23].

Given an instance $(\{a_1, a_2, \ldots, a_n\}, B, k)$ of Unary Bin Packing, where $k$ is the number of bins in the solution and where we can assume that $k \geq 2$, we create an instance $(G, \chi, c)$ of Diameter-Tree as follows. The graph $G$ contains a vertex $r$ and, for $i \in [n]$ and $j \in [k]$, we add to $G$ vertices $v_i, \ell_j^i, r_j^i$ and edges $\{r, \ell_j^1\}$, $\{v_i, \ell_j^i\}$, $\{v_i, r_j^i\}$, and $\{\ell_j^i, r_j^i\}$. Finally, for $i \in [n-1]$ and $j \in [k]$, we add the edge $\{r_j^i, \ell_j^{i+1}\}$. Let $G'$ be the graph constructed so far; see Figure 4 for an illustration.

Similarly to the proof of Theorem 4, we define $G$ to be the graph obtained by taking two disjoint copies of $G'$ and identifying vertex $r$ of both copies. Note that $G$ can be clearly built in

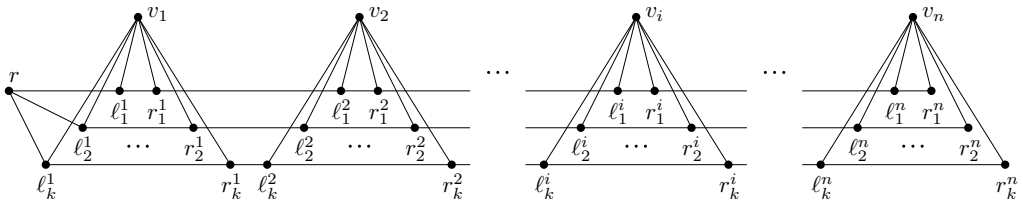



**Figure 4** Graph $G'$ built in the reduction of Theorem 9. The reload costs are not depicted.

polynomial time, and that $\mathsf{tw}(G) \leq k+1$ and $\Delta(G) = 2k$ (since we assume $k \geq 2$). Therefore, $\mathsf{tw}(G)+\Delta(G)$ is indeed bounded by a function of $k$, as required. (Again, the claimed bound on the treewidth can be easily seen by building a *path* decomposition of $G$ with consecutive bags of the form $\{v_i, \ell_1^i, \ell_2^i, \ldots, \ell_k^i, r_1^i\}, \{v_i, \ell_1^i, \ell_2^i, \ldots, \ell_{k-1}^i, r_1^i, r_2^i\}, \{v_i, \ell_1^i, \ell_2^i, \ldots, \ell_{k-2}^i, r_1^i, r_2^i, r_3^i\}, \ldots.)$

Let us now define the coloring $\chi$ and the cost function $c$. Once more, for simplicity, we associate a distinct color with each edge of $G$, and thus it is enough to describe the cost function $c$ for every pair of incident edges of $G$. The cost function is symmetric for both copies of $G'$, so we just focus on one copy. For $i \in [n]$, let $e_1, e_2$ be two distinct edges containing vertex $v_i$. We set $c(e_1, e_2) = 2B + 1$ unless $e_1 = \{v_i, \ell_j^i\}$ and $e_2 = \{v_i, r_j^i\}$ for some $j \in [k]$, in which case we set $c(e_1, e_2) = a_i$. The cost associated with any other pair of edges of $G$ is set to 0. Note that, as $(\{a_1, a_2, \ldots, a_n\}, B, k)$ is an instance of Unary Bin Packing, the reload costs of the instance $(G, \chi, c)$ of Diameter-Tree are polynomially bounded by $|V(G)|$. Again, the following claim concludes the proof.

▶ **Claim 10.** [⋆] *$(\{a_1, a_2, \ldots, a_n\}, B, k)$ is a* Yes*-instance of* Unary Bin Packing *if and only if $G$ has a spanning tree with diameter at most $2B$.* ◀

## References


1 Satyam Agarwal and Swades De. Dynamic spectrum access for energy-constrained cr: single channel versus switched multichannel. *IET Communications*, 10(7):761–769, 2016.

2 E. Amaldi, Giulia Galbiati, and Francesco Maffioli. On minimum reload cost paths, tours, and flows. *Networks*, 57(3):254–260, 2011.

3 Stamatios Arkoulis, Evangelos Anifantis, Vasileios Karyotis, Symeon Papavassiliou, and Nikolaos Mitrou. On the optimal, fair and channel-aware cognitive radio network reconfiguration. *Computer Networks*, 57(8):1739–1757, 2013.

4 Suzan Bayhan and Fatih Alagoz. Scheduling in centralized cognitive radio networks for energy efficiency. *IEEE Transactions on Vehicular Technology*, 62(2):582–595, 2013.

5 Suzan Bayhan, Salim Eryigit, Fatih Alagoz, and Tuna Tugcu. Low complexity uplink schedulers for energy-efficient cognitive radio networks. *IEEE Wireless Communications Letters*, 2(3):363–366, 2013.

6 Aruna Prem Bianzino, Claude Chaudet, Dario Rossi, and Jean-Louis Rougier. A survey of green networking research. *IEEE Communications Surveys & Tutorials*, 14(1):3–20, 2012.

7 Hans L. Bodlaender, Pral Grønras Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016.

8 Abdulkadir Celik and Ahmed E Kamal. Green cooperative spectrum sensing and scheduling in heterogeneous cognitive radio networks. *IEEE Transactions on Cognitive Communications and Networking*, 2(3):238–248, 2016.

9 Claude Desset, Noman Ahmed, and Antoine Dejonghe. Energy savings for wireless terminals through smart vertical handover. In *Proc. of IEEE International Conference on Communications*, pages 1–5, 2009.

**10** Reinhard Diestel. *Graph Theory*, volume 173. Springer-Verlag, 4th edition, 2010.

**11** Salim Eryigit, Suzan Bayhan, and Tuna Tugcu. Channel switching cost aware and energy-efficient cooperative sensing scheduling for cognitive radio networks. In *Proc. of IEEE International Conference on Communications (ICC)*, pages 2633–2638, 2013.

**12** Giulia Galbiati. The complexity of a minimum reload cost diameter problem. *Discrete Applied Mathematics*, 156(18):3494–3497, 2008.

**13** Giulia Galbiati, Stefano Gualandi, and Francesco Maffioli. On minimum changeover cost arborescences. In Panos M. Pardalos and Steffen Rebennack, editors, *Experimental Algorithms - 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proceedings*, volume 6630 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2011. doi:10.1007/978-3-642-20662-7_10.

**14** Giulia Galbiati, Stefano Gualandi, and Francesco Maffioli. On minimum reload cost cycle cover. *Discrete Applied Mathematics*, 164:112–120, 2014.

**15** Ioannis Gamvros, Luis Gouveia, and S Raghavan. Reload cost trees and network design. *Networks*, 59(4):365–379, 2012.

**16** M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.

**17** Laurent Gourvès, Adria Lyra, Carlos Martinhon, and Jérôme Monnot. The minimum reload s-t path, trail and walk problems. *Discrete Applied Mathematics*, 158(13):1404–1417, 2010.

**18** D. Gözüpek, S. Özkan, C. Paul, I. Sau, and M. Shalom. Parameterized complexity of the MINCCA problem on graphs of bounded decomposability. In *Proc. of the 42nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 9941 of *LNCS*, pages 195–206, 2016. Full version available at arXiv:1509.04880.

**19** Didem Gözüpek, Seyed Buhari, and Fatih Alagöz. A spectrum switching delay-aware scheduling algorithm for centralized cognitive radio networks. *IEEE Transactions on Mobile Computing*, 12(7):1270–1280, 2013.

**20** Didem Gözüpek, Hadas Shachnai, Mordechai Shalom, and Shmuel Zaks. Constructing minimum changeover cost arborescenses in bounded treewidth graphs. *Theor. Comput. Sci.*, 621:22–36, 2016. doi:10.1016/j.tcs.2016.01.022.

**21** Didem Gözüpek and Mordechai Shalom. Edge coloring with minimum reload/changeover costs. *Preprint available at* arXiv:1607.06751, 2016.

**22** Didem Gözüpek, Mordechai Shalom, Ariella Voloshin, and Shmuel Zaks. On the complexity of constructing minimum changeover cost arborescences. *Theor. Comput. Sci.*, 540:40–52, 2014. doi:10.1016/j.tcs.2014.03.023.

**23** Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *J. Comput. Syst. Sci.*, 79(1):39–49, 2013. doi:10.1016/j.jcss.2012.04.004.

**24** Finn V. Jensen. *Bayesian Networks and Decision Graphs*. Springer, 2001.

**25** Vijay R Konda and Timothy Y Chow. Algorithm for traffic grooming in optical networks to minimize the number of transceivers. In *Proc. of IEEE Workshop on High Performance Switching and Routing*, pages 218–221, 2001.

**26** Nasser Shami and Mehdi Rasti. A joint multi-channel assignment and power control scheme for energy efficiency in cognitive radio networks. In *Proc. of IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2016.

**27** Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8:85–89, 1984.

**28** Hans-Christoph Wirth and Jan Steffan. Reload cost problems: minimum diameter spanning tree. *Discrete Applied Mathematics*, 113(1):73–85, 2001.

# Optimal Algorithms for Hitting (Topological) Minors on Graphs of Bounded Treewidth*†

**Julien Baste[1], Ignasi Sau[2], and Dimitrios M. Thilikos[3]**

1 **Université de Montpellier, LIRMM, Montpellier, France**
baste@lirmm.fr

2 **AlGCo project-team, CNRS, LIRMM, France and Departamento de Matemática, Universidade Federal do Ceará, Fortaleza, Brazil**
ignasi.sau@lirmm.fr

3 **AlGCo project-team, CNRS, LIRMM, France and Department of Mathematics, National and Kapodistrian University of Athens, Greece**
sedthilk@thilikos.info

## Abstract

For a fixed collection of graphs $\mathcal{F}$, the $\mathcal{F}$-M-Deletion problem consists in, given a graph $G$ and an integer $k$, decide whether there exists $S \subseteq V(G)$ with $|S| \leq k$ such that $G \setminus S$ does not contain any of the graphs in $\mathcal{F}$ as a minor. We are interested in the parameterized complexity of $\mathcal{F}$-M-Deletion when the parameter is the treewidth of $G$, denoted by $\mathbf{tw}$. Our objective is to determine, for a fixed $\mathcal{F}$, the smallest function $f_{\mathcal{F}}$ such that $\mathcal{F}$-M-Deletion can be solved in time $f_{\mathcal{F}}(\mathbf{tw}) \cdot n^{\mathcal{O}(1)}$ on $n$-vertex graphs. Using and enhancing the machinery of boundaried graphs and small sets of representatives introduced by Bodlaender *et al.* [J ACM, 2016], we prove that when all the graphs in $\mathcal{F}$ are connected and at least one of them is planar, then $f_{\mathcal{F}}(w) = 2^{\mathcal{O}(w \cdot \log w)}$. When $\mathcal{F}$ is a singleton containing a clique, a cycle, or a path on $i$ vertices, we prove the following asymptotically tight bounds:

- $f_{\{K_4\}}(w) = 2^{\Theta(w \cdot \log w)}$.
- $f_{\{C_i\}}(w) = 2^{\Theta(w)}$ for every $i \leq 4$, and $f_{\{C_i\}}(w) = 2^{\Theta(w \cdot \log w)}$ for every $i \geq 5$.
- $f_{\{P_i\}}(w) = 2^{\Theta(w)}$ for every $i \leq 4$, and $f_{\{P_i\}}(w) = 2^{\Theta(w \cdot \log w)}$ for every $i \geq 6$.

The lower bounds hold unless the Exponential Time Hypothesis fails, and the superexponential ones are inspired by a reduction of Marcin Pilipczuk [Discrete Appl Math, 2016]. The single-exponential algorithms use, in particular, the rank-based approach introduced by Bodlaender *et al.* [Inform Comput, 2015]. We also consider the version of the problem where the graphs in $\mathcal{F}$ are forbidden as *topological* minors, and prove essentially the same set of results holds.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

**Keywords and phrases** parameterized complexity, graph minors, treewidth, hitting minors, topological minors, dynamic programming, Exponential Time Hypothesis

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.4

* This work has been supported by project DEMOGRAPH (ANR-16-CE40-0028).
† A full version of this article is permanently available at https://arxiv.org/abs/1704.07284.





## 1 Introduction

Let $\mathcal{F}$ be a finite non-empty collection of non-empty graphs. In the $\mathcal{F}$-M-Deletion (resp. $\mathcal{F}$-TM-Deletion) problem, we are given a graph $G$ and an integer $k$, and the objective is to decide whether there exists a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G \setminus S$ does not contain any of the graphs in $\mathcal{F}$ as a minor (resp. topological minor). These problems have a big expressive power, as instantiations of them correspond to several notorious problems. For instance, the cases $\mathcal{F} = \{K_2\}$, $\mathcal{F} = \{K_3\}$, and $\mathcal{F} = \{K_5, K_{3,3}\}$ of $\mathcal{F}$-M-Deletion (or $\mathcal{F}$-TM-Deletion) correspond to Vertex Cover, Feedback Vertex Set, and Vertex Planarization, respectively.

For the sake of readability, we use the notation $\mathcal{F}$-Deletion in statements that apply to *both* $\mathcal{F}$-M-Deletion and $\mathcal{F}$-TM-Deletion. Note that if $\mathcal{F}$ contains a graph with at least one edge, then $\mathcal{F}$-Deletion is NP-hard by the classical result of Lewis and Yannakakis [15].

In this article we are interested in the parameterized complexity of $\mathcal{F}$-Deletion when the parameter is the treewidth of the input graph. Since the property of containing a graph as a (topological) minor can be expressed in Monadic Second Order logic (see [14] for explicit formulas), by Courcelle's theorem [5], $\mathcal{F}$-Deletion can be solved in time $\mathcal{O}^*(f(\mathbf{tw}))$ on graphs with treewidth at most $\mathbf{tw}$, where $f$ is some computable function[1]. Our objective is to determine, for a fixed collection $\mathcal{F}$, which is the *smallest* such function $f$ that one can (asymptotically) hope for, subject to reasonable complexity assumptions.

This line of research has attracted some interest during the last years in the parameterized complexity community. For instance, Vertex Cover is easily solvable in time $\mathcal{O}^*(2^{\mathcal{O}(\mathbf{tw})})$, called *single-exponential*, by standard dynamic-programming techniques, and no algorithm with running time $\mathcal{O}^*(2^{o(\mathbf{tw})})$ exists unless the Exponential Time Hypothesis (ETH)[2] fails [12].

For Feedback Vertex Set, standard dynamic programming techniques give a running time of $\mathcal{O}^*(2^{\mathcal{O}(\mathbf{tw}\cdot\log\mathbf{tw})})$, while the lower bound under the ETH [12] is again $\mathcal{O}^*(2^{o(\mathbf{tw})})$. This gap remained open for a while, until Cygan *et al.* [6] presented an optimal algorithm running in time $\mathcal{O}^*(2^{\mathcal{O}(\mathbf{tw})})$, using the celebrated *Cut&Count* technique. This article triggered several other techniques to obtain single-exponential algorithms for so-called *connectivity problems* on graph of bounded treewidth, mostly based on algebraic tools [2, 8].

Concerning Vertex Planarization, Jansen *et al.* [13] presented an algorithm of time $\mathcal{O}^*(2^{\mathcal{O}(\mathbf{tw}\cdot\log\mathbf{tw})})$ as a crucial subroutine in an FPT algorithm parameterized by $k$. Marcin Pilipczuk [19] proved that this running time is *optimal* under the ETH, by using the framework introduced by Lokshtanov *et al.* [17] for proving superexponential lower bounds.

**Our results.** We present a number of upper and lower bounds for $\mathcal{F}$-Deletion parameterized by treewidth, several of them being tight. Namely, we prove the following results, all the lower bounds holding under the ETH:

1. For every $\mathcal{F}$, $\mathcal{F}$-Deletion can be solved in time $\mathcal{O}^*\left(2^{2^{\mathcal{O}(\mathbf{tw}\cdot\log\mathbf{tw})}}\right)$.
2. For every connected[3] $\mathcal{F}$ containing at least one planar graph (resp. subcubic planar graph), $\mathcal{F}$-M-Deletion (resp. $\mathcal{F}$-TM-Deletion) can be solved in time $\mathcal{O}^*\left(2^{\mathcal{O}(\mathbf{tw}\cdot\log\mathbf{tw})}\right)$.
3. For any connected $\mathcal{F}$, $\mathcal{F}$-Deletion cannot be solved in time $\mathcal{O}^*(2^{o(\mathbf{tw})})$.
4. When $\mathcal{F} = \{K_i\}$, the clique on $i$ vertices, $\{K_i\}$-Deletion cannot be solved in time $\mathcal{O}^*(2^{o(\mathbf{tw}\cdot\log\mathbf{tw})})$ for $i \geq 4$. Note that $\{K_i\}$-Deletion can be solved in time $\mathcal{O}^*(2^{\mathcal{O}(\mathbf{tw})})$ for $i \leq 3$ [6], and that the case $i = 4$ is tight by item **2** above (as $K_4$ is planar).

[1] We use the notation $\mathcal{O}^*(\cdot)$ that suppresses polynomial factors depending on the size of the input graph.
[2] The ETH states that 3-SAT on $n$ variables cannot be solved in time $2^{o(n)}$; see [12] for more details.
[3] A *connected* collection $\mathcal{F}$ is a collection containing only connected graphs.

**Table 1** Summary of our results when $\mathcal{F}$ equals $\{K_i\}$, $\{C_i\}$, or $\{P_i\}$. If only one value '$x$' is written in the table (like '**tw**'), it means that the corresponding problem can be solved in time $\mathcal{O}^*(2^{\mathcal{O}(x)})$, and that this bound is tight. An entry of the form '$x$ (?) $y$' means that the corresponding problem cannot be solved in time $\mathcal{O}^*(2^{o(x)})$ and that it can be solved in time $\mathcal{O}^*(2^{\mathcal{O}(y)})$. We interpret $\{C_2\}$-Deletion as Feedback Vertex Set. Grey cells correspond to known results.

| $\mathcal{F}$ \ $i$ | 2 | 3 | 4 | 5 | $\geq 6$ |
|---|---|---|---|---|---|
| $K_i$ | **tw** | **tw** | $\mathbf{tw} \cdot \log \mathbf{tw}$ | $\mathbf{tw} \cdot \log \mathbf{tw}$ (?) $2^{\mathcal{O}(\mathbf{tw} \cdot \log \mathbf{tw})}$ | $\mathbf{tw} \cdot \log \mathbf{tw}$ (?) $2^{\mathcal{O}(\mathbf{tw} \cdot \log \mathbf{tw})}$ |
| $C_i$ | **tw** | **tw** | **tw** | $\mathbf{tw} \cdot \log \mathbf{tw}$ | $\mathbf{tw} \cdot \log \mathbf{tw}$ |
| $P_i$ | **tw** | **tw** | **tw** | **tw** (?) $\mathbf{tw} \cdot \log \mathbf{tw}$ | $\mathbf{tw} \cdot \log \mathbf{tw}$ |

5. When $\mathcal{F} = \{C_i\}$, the cycle on $i$ vertices, $\{C_i\}$-Deletion can be solved in time $\mathcal{O}^*(2^{\mathcal{O}(\mathbf{tw})})$ for $i \leq 4$, and cannot be solved in time $\mathcal{O}^*(2^{o(\mathbf{tw} \cdot \log \mathbf{tw})})$ for $i \geq 5$. Note that, by items **2** and **3** above, this settles completely the complexity of $\{C_i\}$-Deletion for every $i \geq 3$.
6. When $\mathcal{F} = \{P_i\}$, the path on $i$ vertices, $\{P_i\}$-Deletion can be solved in time $\mathcal{O}^*(2^{\mathcal{O}(\mathbf{tw})})$ for $i \leq 4$, and cannot be solved in time $\mathcal{O}^*(2^{o(\mathbf{tw} \cdot \log \mathbf{tw})})$ for $i \geq 6$. Note that, by items **2** and **3** above, this settles completely the complexity of $\{P_i\}$-Deletion for every $i \geq 2$, except for $i = 5$, where there is still a gap.

The results discussed in the last three items are summarized in Table 1. Note that the cases with $i \leq 3$ were already known [6, 12], except when $\mathcal{F} = \{P_3\}$.

**Our techniques.** The algorithm running in time $\mathcal{O}^*\left(2^{2^{\mathcal{O}(\mathbf{tw} \cdot \log \mathbf{tw})}}\right)$ uses and, in a sense, enhances, the machinery of boundaried graphs, equivalence relations, and representatives originating in the seminal work of Bodlaender *et al.* [3], and which has been subsequently used in [9, 10, 14]. For technical reasons, we use *branch* decompositions instead of tree decompositions, whose associated widths are equivalent from a parametric point of view [20].

In order to obtain the faster algorithm running in time $\mathcal{O}^*\left(2^{\mathcal{O}(\mathbf{tw} \cdot \log \mathbf{tw})}\right)$ when $\mathcal{F}$ is a connected collection containing at least a (subcubic) planar graph, we combine the above ingredients with additional arguments to bound the number and the size of the representatives of the equivalence relation defined by the encoding that we use to construct the partial solutions. Here, the connectivity of $\mathcal{F}$ guarantees that every connected component of a minimum-sized representative intersects its boundary set (cf. the full version). The fact that $\mathcal{F}$ contains a (subcubic) planar graph is essential in order to bound the treewidth of the resulting graph after deleting a partial solution (cf. Lemma 11).

We present these algorithms for the topological minor version and then it is easy to adapt them to the minor version within the claimed running time (cf. Lemma 9).

The single-exponential algorithms when $\mathcal{F} \in \{\{P_3\}, \{P_4\}, \{C_4\}\}$ are ad hoc. Namely, the algorithms for $\{P_3\}$-Deletion and $\{P_4\}$-Deletion use standard (but nontrivial) dynamic programming techniques on graphs of bounded treewidth, exploiting the simple structure of graphs that do not contain $P_3$ or $P_4$ as a minor (or as a subgraph, which in the case of paths is equivalent). The algorithm for $\{C_4\}$-Deletion is more involved, and uses the rank-based approach introduced by Bodlaender *et al.* [2], exploiting again the structure of graphs that do not contain $C_4$ as a minor (cf. Lemma 14). It might seem counterintuitive that this technique works for $C_4$, and stops working for $C_i$ with $i \geq 5$ (see Table 1). A possible reason for that is that the only cycles of a $C_4$-minor-free graph are triangles and each triangle is contained in a bag of a tree decomposition. This property, which is not true anymore for $C_i$-minor-free graphs with $i \geq 5$, permits to keep track of the structure of partial solutions with tables of small size.

As for the lower bounds, the general lower bound of $\mathcal{O}^*(2^{o(\mathbf{tw})})$ for connected collections is based on a simple reduction from Vertex Cover. The superexponential lower bounds, namely $\mathcal{O}^*(2^{o(\mathbf{tw}\cdot\log\mathbf{tw})})$, are strongly based on the ideas presented by Marcin Pilipczuk [19] for Vertex Planarization. We present a general hardness result (cf. Theorem 20) that applies to wide families of connected collections $\mathcal{F}$. Then, our superexponential lower bounds, as well as the result of Marcin Pilipczuk [19] itself, are corollaries of this general result. Combining Theorem 20 with **2**, it easily follows that the running time $\mathcal{O}^*(2^{\mathcal{O}(\mathbf{tw}\cdot\log\mathbf{tw})})$ is tight for a wide family of $\mathcal{F}$, for example, when all graphs in $\mathcal{F}$ are planar and 3-connected.

**Further research.** In order to complete the dichotomy for cliques and paths (see Table 1), it remains to settle the complexity when $\mathcal{F} = \{K_i\}$ with $i \geq 5$ and when $\mathcal{F} = \{P_5\}$. An ultimate goal is to establish the tight complexity of $\mathcal{F}$-Deletion for all collections $\mathcal{F}$, but we are still very far from it. In particular, we do not know whether there exists some $\mathcal{F}$ for which a double-exponential lower bound can be proved, or for which the complexities of $\mathcal{F}$-M-Deletion and $\mathcal{F}$-TM-Deletion differ.

Note that the connectivity of $\mathcal{F}$ was relevant in previous work on the $\mathcal{F}$-M-Deletion problem taking as the parameter the size of the solution [7, 14]. Getting rid of connectivity in both the lower and upper bounds we presented is an interesting avenue. We did not focus on optimizing either the degree of the polynomials involved or the constants involved in our algorithms. Concerning the latter, one could use the framework presented by Lokshtanov *et al.* [16] to prove lower bounds based on the *Strong* Exponential Time Hypothesis.

Finally, let us mention that Bonnet *et al.* [4] recently studied generalized feedback vertex set problems parameterized by treewidth, and obtained independently that excluding $C_4$ plays a fundamental role in the existence of single-exponential algorithms, similarly to our dichotomy for cycles summarized in Table 1.

**Organization of the paper.** In Section 2 we provide some preliminaries. The algorithms based on boundaried graphs are presented in Section 3, and the single-exponential algorithms for hitting paths and cycles are presented in Section 4. The superexponential lower bounds are presented in Section 5. The general lower bound for connected collections and the proofs of all the results marked with '($\star$)' can be found in the full version.

## 2 Preliminaries

In this section we provide some preliminaries to be used in the following sections. We include here only the "non-standard" definitions; the other ones can be found in the full version.

**Block-cut trees.** A connected graph $G$ is *biconnected* if for any $v \in V(G)$, $G \setminus \{v\}$ is connected (notice that $K_2$ is the only biconnected graph that it is not 2-connected). A *block* of a graph $G$ is a maximal biconnected subgraph of $G$. We name $\mathsf{block}(G)$ the set of all blocks of $G$ and we name $\mathsf{cut}(G)$ the set of all cut vertices of $G$. If $G$ is connected, we define the *block-cut tree* of $G$ to be the tree $\mathsf{bct}(G) = (V, E)$ such that $V = \mathsf{block}(G) \cup \mathsf{cut}(G)$ and $E = \{\{B, v\} \mid B \in \mathsf{block}(G), v \in \mathsf{cut}(G) \cap V(B)\}$. Note that $L(\mathsf{bct}(G)) \subseteq \mathsf{block}(G)$. The block-cut tree of a graph can be computed in linear time using depth-first search [11]. Let $\mathcal{F}$ be a set of connected graphs such that for each $H \in \mathcal{F}$, $|V(H)| \geq 2$. Given $H \in \mathcal{F}$ and $B \in L(\mathsf{bct}(H))$, we say that $(H, B)$ is an *essential pair* if for each $H' \in \mathcal{F}$ and each $B' \in L(\mathsf{bct}(H'))$, $|E(B)| \leq |E(B')|$. Given an essential pair $(H, B)$ of $\mathcal{F}$, we define the *first vertex* of $(H, B)$ to be, if it exists, the only cut vertex of $H$ contained in $V(B)$, or an

arbitrarily chosen vertex of $V(B)$ otherwise. We define the *second vertex* of $(H, B)$ to be an arbitrarily chosen vertex of $V(B)$ that is a neighbor in $H[B]$ of the first vertex of $(H, B)$. Note that, given an essential pair $(H, B)$ of $\mathcal{F}$, the first vertex and the second vertex of $(H, B)$ exist and, by definition, are fixed. Moreover, given an essential pair $(H, B)$ of $\mathcal{F}$, we define the *core* of $(H, B)$ to be the graph $H \setminus (V(B) \setminus \{a\})$ where $a$ is the first vertex of $(H, B)$. Note that $a$ is a vertex of the core of $(H, B)$.

**Topological minors and graph separators.** For the statement of our results, we need to consider the class $\mathcal{K}$ containing every connected graph $G$ such that for each $B \in L(\mathsf{bct}(G))$ and for each $r \in \mathbb{N}$, $B \not\preceq_{\mathsf{tm}} K_{2,r}$ (or equivalently, $B \not\preceq_{\mathsf{m}} K_{2,r}$). Let $H$ be a graph. We define the set of graphs $\mathsf{tpm}(H)$ as follows: among all the graphs containing $H$ as a minor, we consider only those that are minimal with respect to the topological minor relation.

▶ **Observation 1.** *There is a function $f_1 : \mathbb{N} \to \mathbb{N}$ such that for every $h$-vertex graph $H$, every graph in $\mathsf{tpm}(H)$ has at most $f_1(h)$ vertices.*

▶ **Observation 2.** *Given two graphs $H$ and $G$, $H$ is a minor of $G$ if and only if some of the graphs in $\mathsf{tpm}(H)$ is a topological minor of $G$.*

Let $G$ be a graph and $S \subseteq V(G)$. Then for each connected component $C$ of $G \setminus S$, we define the *cut-clique* of the triple $(C, G, S)$ to be the graph whose vertex set is $V(C) \cup S$ and whose edge set is $E(G[V(C) \cup S]) \cup \binom{S}{2}$.

▶ **Lemma 3** ($\star$). *Let $i \geq 2$ be an integer, let $H$ be an $i$-connected graph, let $G$ be a graph, and let $S \subseteq V(G)$ such that $|S| \leq i - 1$. If $H$ is a topological minor (resp. a minor) of $G$, then there exists a connected component $G'$ of $G \setminus S$ such that $H$ is a topological minor (resp. a minor) of the cut-clique of $(G', G, S)$.*

▶ **Lemma 4** ($\star$). *Let $G$ be a connected graph, let $v$ be a cut vertex of $G$, and let $V$ be the vertex set of a connected component of $G \setminus \{v\}$. If $H$ is a connected graph such that $H \preceq_{\mathsf{tm}} G$ and for each leaf $B$ of $\mathsf{bct}(H)$, $B \not\preceq_{\mathsf{tm}} G[V \cup \{v\}]$, then $H \preceq_{\mathsf{tm}} G \setminus V$.*

**Graph collections.** Let $\mathcal{F}$ be a collection of graphs. From now on instead of "collection of graphs" we use the shortcut "collection". If $\mathcal{F}$ is a collection that is finite, non-empty, and all its graphs are non-empty, then we say that $\mathcal{F}$ is a *proper collection*. For any proper collection $\mathcal{F}$, we define $\mathsf{size}(\mathcal{F}) = \max\{\{|V(H)| \mid H \in \mathcal{F}\} \cup \{|\mathcal{F}|\}\}$. Note that if the size of $\mathcal{F}$ is bounded, then the size of the graphs in $\mathcal{F}$ is also bounded. We say that $\mathcal{F}$ is a *planar collection* (resp. *planar subcubic collection*) if it is proper and *at least one* of the graphs in $\mathcal{F}$ is planar (resp. planar and subcubic). We say that $\mathcal{F}$ is a *connected collection* if it is proper and *all* the graphs in $\mathcal{F}$ are connected. We say that $\mathcal{F}$ is an *(topological) minor antichain* if no two of its elements are comparable via the (topological) minor relation.

Let $\mathcal{F}$ be a proper collection. We extend the (topological) minor relation to $\mathcal{F}$ such that, given a graph $G$, $\mathcal{F} \preceq_{\mathsf{tm}} G$ (resp. $\mathcal{F} \preceq_{\mathsf{m}} G$) if and only if there exists a graph $H \in \mathcal{F}$ such that $H \preceq_{\mathsf{tm}} G$ (resp. $H \preceq_{\mathsf{m}} G$). We also denote $\mathsf{ex}_{\mathsf{tm}}(\mathcal{F}) = \{G \mid \mathcal{F} \not\preceq_{\mathsf{tm}} G\}$, i.e., $\mathsf{ex}_{\mathsf{tm}}(\mathcal{F})$ is the class of graphs that do not contain any graph in $\mathcal{F}$ as a topological minor. The set $\mathsf{ex}_{\mathsf{m}}(\mathcal{F})$ is defined analogously.

**Definition of the problems.** Let $\mathcal{F}$ be a proper collection. We define the parameter $\mathbf{tm}_{\mathcal{F}}$ as the function that maps graphs to non-negative integers as follows:

$$\mathbf{tm}_{\mathcal{F}}(G) \quad = \quad \min\{|S| \mid S \subseteq V(G) \wedge G \setminus S \in \mathsf{ex}_{\mathsf{tm}}(\mathcal{F})\}. \tag{1}$$

The parameter $\mathbf{m}_{\mathcal{F}}$ is defined analogously. The main objective of this paper is to study the problem of computing the parameters $\mathbf{tm}_{\mathcal{F}}$ and $\mathbf{m}_{\mathcal{F}}$ for graphs of bounded treewidth under several instantiations of the collection $\mathcal{F}$. Note that in both problems, we can always assume that $\mathcal{F}$ is an antichain with respect to the considered relation. Indeed, this is the case because if $\mathcal{F}$ contains two graphs $H_1$ and $H_2$ where $H_1 \preceq_{\mathsf{tm}} H_2$, then $\mathbf{tm}_{\mathcal{F}}(G) = \mathbf{tm}_{\mathcal{F}'}(G)$ where $\mathcal{F}' = \mathcal{F} \setminus \{H_2\}$ (similarly for the minor relation).

Throughout the article, we let $n$ and $\mathbf{tw}$ be the number of vertices and the treewidth of the input graph of the considered problem, respectively. In some proofs, we will also use $w$ to denote the width of a (nice) tree decomposition that is given together with the input graph (which will differ from $\mathbf{tw}$ by at most a factor 5).

## 3 Dynamic programming algorithms for computing $tm_{\mathcal{F}}$

The purpose of this section is to prove the following results.

▶ **Theorem 5.** *If $\mathcal{F}$ is a proper collection, where $d = \mathsf{size}(\mathcal{F})$, then there exists an algorithm that solves $\mathcal{F}$-TM-Deletion in $2^{2^{\mathcal{O}_d(\mathbf{tw}\cdot\log\mathbf{tw})}} \cdot n$ steps.*

▶ **Theorem 6.** *If $\mathcal{F}$ is a connected and planar subcubic collection, where $d = \mathsf{size}(\mathcal{F})$, then there exists an algorithm that solves $\mathcal{F}$-TM-Deletion in $2^{\mathcal{O}_d(\mathbf{tw}\cdot\log\mathbf{tw})} \cdot n$ steps.*

▶ **Theorem 7.** *If $\mathcal{F}$ is a proper collection, where $d = \mathsf{size}(\mathcal{F})$, then there exists an algorithm that solves $\mathcal{F}$-M-Deletion in $2^{2^{\mathcal{O}_d(\mathbf{tw}\cdot\log\mathbf{tw})}} \cdot n$ steps.*

▶ **Theorem 8.** *If $\mathcal{F}$ is a connected and planar collection, where $d = \mathsf{size}(\mathcal{F})$, then there exists an algorithm that solves $\mathcal{F}$-M-Deletion in $2^{\mathcal{O}_d(\mathbf{tw}\cdot\log\mathbf{tw})} \cdot n$ steps.*

The following lemma is a direct consequence of Observation 2.

▶ **Lemma 9.** *Let $\mathcal{F}$ be a proper collection. Then, for every graph $G$, it holds that $\mathbf{m}_{\mathcal{F}}(G) = \mathbf{tm}_{\mathcal{F}'}(G)$ where $\mathcal{F}' = \bigcup_{F\in\mathcal{F}} \mathsf{tpm}(F)$.*

It is easy to see that for every (planar) graph $F$, the set $\mathsf{tpm}(F)$ contains a subcubic (planar) graph. Combining this observation with Lemma 9 and Observation 1, Theorems 7 and 8 follow directly from Theorems 5 and 6, respectively. The rest of this section is dedicated to the proofs of Theorems 5 and 6. For this, we need a number of definitions about boundaried graphs, their equivalence classes, and their branch decompositions. Many of these definitions were introduced in [3,9] (see also [10,14]), and can be found in the full version. We present here only the most fundamental definitions in order to be able to state our results.

**Basic definitions about boundaried graphs.** Let $t \in \mathbb{N}$. A *$t$-boundaried graph* is a triple $\mathbf{G} = (G, R, \lambda)$ where $G$ is a graph, $R \subseteq V(G)$, $|R| = t$, and $\lambda : R \to \mathbb{N}^+$ is an injective function. We call $R$ the *boundary* of $\mathbf{G}$ and we call the vertices of $R$ *the boundary vertices* of $\mathbf{G}$. We also call $G$ *the underlying graph* of $\mathbf{G}$. Moreover, we call $t = |R|$ the *boundary size* of $\mathbf{G}$ and we define the *label set* of $\mathbf{G}$ as $\Lambda(\mathbf{G}) = \lambda(R)$. We also say that $\mathbf{G}$ is a *boundaried graph* if there exists an integer $t$ such that $\mathbf{G}$ is an $t$-boundaried graph. We say that a boundary graph $\mathbf{G}$ is *consecutive* if $\Lambda(\mathbf{G}) = [1, |R|]$. We define $\mathcal{B}^{(t)}$ as the set of all $t$-boundaried graphs.

Let $\mathbf{G}_1 = (G_1, R_1, \lambda_1)$ and $\mathbf{G}_2 = (G_2, R_2, \lambda_2)$ be two $t$-boundaried graphs. We define the *gluing operation* $\oplus$ such that $(G_1, R_1, \lambda_1) \oplus (G_2, R_2, \lambda_2)$ is the graph $G$ obtained by taking the disjoint union of $G_1$ and $G_2$ and then, for each $i \in [1, t]$, identifying the vertex $\psi_{\mathbf{G}_1}^{-1}(i)$ and the vertex $\psi_{\mathbf{G}_2}^{-1}(i)$.

Let $\mathcal{F}$ be a proper collection and let $t$ be a non-negative integer. We define an equivalence relation $\equiv^{(\mathcal{F},t)}$ on $t$-boundaried graphs as follows: Given two $t$-boundaried graphs $\mathbf{G}_1$ and $\mathbf{G}_2$, we write $\mathbf{G}_1 \equiv^{(\mathcal{F},t)} \mathbf{G}_2$ to denote that $\forall \mathbf{G} \in \mathcal{B}^{(t)}$, $\mathcal{F} \preceq_{\mathsf{tm}} \mathbf{G} \oplus \mathbf{G}_1 \iff \mathcal{F} \preceq_{\mathsf{tm}} \mathbf{G} \oplus \mathbf{G}_2$. We set up a *set of representatives* $\mathcal{R}^{(\mathcal{F},t)}$ as a set containing, for each equivalence class $\mathcal{C}$ of $\equiv^{(\mathcal{F},t)}$, some consecutive $t$-boundaried graph in $\mathcal{C}$ with minimum number of edges and no isolated vertices out of its boundary (if there are more than one such graphs, pick one arbitrarily). Given a $t$-boundaried graph $\mathbf{G}$ we denote by $\mathsf{rep}^{(\mathcal{F})}(\mathbf{G})$ the $t$-boundaried graph $\mathbf{B} \in \mathcal{R}^{(\mathcal{F},t)}$ where $\mathbf{B} \equiv^{(\mathcal{F},t)} \mathbf{G}$ and we call $\mathbf{B}$ the *$\mathcal{F}$-representative of* $\mathbf{G}$.

Given $t, r \in \mathbb{N}$, we define $\mathcal{A}^{(t)}_{\mathcal{F},r}$ as the set of all pairwise non-isomorphic boundaried graphs that contain at most $r$ non-boundary vertices, whose label set is a subset of $[1,t]$, and whose underlying graph belongs in $\mathsf{ex}_{\mathsf{tm}}(\mathcal{F})$. Given a $t$-boundaried graph $\mathbf{B}$ and an integer $r \in \mathbb{N}$, we define the $(\mathcal{F},r)$-folio of $\mathbf{B}$, denoted by $\mathsf{folio}(\mathbf{B},\mathcal{F},r)$, as the set containing all boundaried graphs in $\mathcal{A}^{(t)}_{\mathcal{F},r}$ that are topological minors of $\mathbf{B}$.

▶ **Lemma 10** ($\star$). *There exists a function $h_1 : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ such that if $\mathcal{F}$ is a proper collection and $t \in \mathbb{N}$, then $|\mathcal{R}^{(\mathcal{F},t)}| \leq h_1(d,t)$ where $d = \mathsf{size}(\mathcal{F})$. Moreover $h_1(d,t) = 2^{2^{\mathcal{O}_d(t\cdot\log t)}}$.*

▶ **Lemma 11** ($\star$). *There exists a function $\mu : \mathbb{N} \to \mathbb{N}$ such that for every planar subcubic collection $\mathcal{F}$, every graph in $\mathsf{ex}_{\mathsf{tm}}(\mathcal{F})$ has branchwidth at most $y = \mu(d)$ where $d = \mathsf{size}(\mathcal{F})$.*

We already have all the main ingredients to prove Theorem 5; the proof can be found in the full version. In order to prove Theorem 6, we need Lemma 13 below, which should be contrasted with Lemma 10. Its proof, which can be found in the full version, uses, among others, the following result of Baste *et al.* [1] on the number of labeled graphs of bounded treewidth.

▶ **Proposition 12** (Baste *et al.* [1]). *Let $n, y \in \mathbb{N}$. The number of labeled graphs with at most $n$ vertices and branchwidth at most $q$ is $2^{\mathcal{O}_q(n\cdot\log n)}$.*

▶ **Lemma 13** ($\star$). *Let $t \in \mathbb{N}$ and $\mathcal{F}$ be a connected and planar collection, where $d = \mathsf{size}(\mathcal{F})$, and let $\mathcal{R}^{(\mathcal{F},t)}$ be a set of representatives. Then $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_d(t\cdot\log t)}$. Moreover, there exists an algorithm that given $\mathcal{F}$ and $t$, constructs a set of representatives $\mathcal{R}^{(\mathcal{F},t)}$ in $2^{\mathcal{O}_d(t\cdot\log t)}$ steps.*

The proof of Theorem 6 can be found in the full version. The main difference with respect to the proof of Theorem 5 is an improvement on the size of the tables of the dynamic programming algorithm, namely $|\mathcal{P}_e|$, where the fact that $\mathcal{F}$ is a connected and planar subcubic collection is exploited.

## 4 Single-exponential algorithms for hitting paths and cycles

In this section we show that if $\mathcal{F} \in \{\{P_3\},\{P_4\},\{C_4\}\}$, then $\mathcal{F}$-TM-Deletion can also be solved in single-exponential time. It is worth mentioning that the $\{C_i\}$-TM-Deletion problem has been studied in digraphs from a non-parameterized point of view [18].

The algorithms we present for $\{P_3\}$-TM-Deletion and $\{P_4\}$-TM-Deletion use standard dynamic programming techniques, and can be found in the full version. The definition of *nice tree decomposition* can also be found there.

We proceed to use the dynamic programming techniques introduced by Bodlaender *et al.* [2] to obtain a single-exponential algorithm for $\{C_4\}$-TM-Deletion. The algorithm we present solves the decision version of $\{C_4\}$-TM-Deletion: the input is a pair $(G,k)$, where $G$ is a graph and $k$ is an integer, and the output is the boolean value $\mathbf{tm}_{\mathcal{F}}(G) \leq k$.

Given a graph $G$, we denote by $n(G) = |V(G)|$, $m(G) = |E(G)|$, $\mathsf{c}_3(G)$ the number of $C_3$'s that are subgraphs of $G$, and $\mathsf{cc}(G)$ the number of connected components of $G$. We

say that $G$ satisfies the $C_4$-*condition* if $G$ does not contain the diamond as a subgraph and $n(G) - m(G) + \mathsf{c}_3(G) = \mathsf{cc}(G)$. As in the case of $P_3$ and $P_4$, we state in Lemma 14 a structural characterization of the graphs that exclude $C_4$ as a (topological) minor.

▶ **Lemma 14** ($\star$). *Let $G$ be a graph. $C_4 \not\preceq_{\mathsf{tm}} G$ if and only if $G$ satisfies the $C_4$-condition.*

▶ **Lemma 15** ($\star$). *If $G$ is a non-empty graph such that $C_4 \not\preceq_{\mathsf{tm}} G$, then $m(G) \leq \frac{3}{2}(n(G)-1)$.*

We are now going to restate the tools introduced by Bodlaender *et al.* [2] that we need for our purposes. Let $U$ be a set. We define $\Pi(U)$ to be the set of all partitions of $U$. Given two partitions $p$ and $q$ of $U$, we define the coarsening relation $\sqsubseteq$ such that $p \sqsubseteq q$ if for each $S \in q$, there exists $S' \in p$ such that $S \subseteq S'$. $(\Pi(U), \sqsubseteq)$ defines a lattice with minimum element $\{\{U\}\}$ and maximum element $\{\{x\} \mid x \in U\}$. On this lattice, we denote by $\sqcap$ the meet operation and by $\sqcup$ the join operation. Let $p \in \Pi(U)$. For $X \subseteq U$ we denote by $p_{\downarrow X} = \{S \cap X \mid S \in p, S \cap X \neq \emptyset\} \in \Pi(X)$ the partition obtained by removing all elements not in $X$ from $p$, and analogously for $U \subseteq X$ we denote $p_{\uparrow X} = p \cup \{\{x\} \mid x \in X \setminus U\} \in \Pi(X)$ the partition obtained by adding to $p$ a singleton for each element in $X \setminus U$. Given a subset $S$ of $U$, we define the partition $U[S] = \{\{x\} \mid x \in U \setminus S\} \cup \{S\}$. A set of *weighted partitions* is a set $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$. We also define $\mathsf{rmc}(\mathcal{A}) = \{(p, w) \in \mathcal{A} \mid \forall (p', w') \in \mathcal{A} : p' = p \Rightarrow w \leq w'\}$.

We now define some operations on weighted partitions. Let $U$ be a set and $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$.

**Union.** Given $\mathcal{B} \subseteq \Pi(U) \times \mathbb{N}$, we define $\mathcal{A} \uplus \mathcal{B} = \mathsf{rmc}(\mathcal{A} \cup \mathcal{B})$.

**Insert.** Given a set $X$ such that $X \cap U = \emptyset$, we define $\mathsf{ins}(X, \mathcal{A}) = \{(p_{\uparrow U \cup X}, w) \mid (p, w) \in \mathcal{A}\}$.

**Shift.** Given $w' \in \mathbb{N}$, we define $\mathsf{shft}(w', \mathcal{A}) = \{(p, w + w') \mid (p, w) \in \mathcal{A}\}$.

**Glue.** Given a set $S$, we define $\hat{U} = U \cup S$ and $\mathsf{glue}(S, \mathcal{A}) \subseteq \Pi(\hat{U}) \times \mathbb{N}$ as
$\mathsf{glue}(S, \mathcal{A}) = \mathsf{rmc}(\{(\hat{U}[S] \sqcap p_{\uparrow \hat{U}}, w \mid (p, w) \in \mathcal{A}\})$.
Given $w : \hat{U} \times \hat{U} \to \mathcal{N}$, we define $\mathsf{glue}_w(\{u, v\}, \mathcal{A}) = \mathsf{shft}(w(u, v), \mathsf{glue}(\{u, v\}, \mathcal{A}))$.

**Project.** Given $X \subseteq U$, we define $\overline{X} = U \setminus X$ and $\mathsf{proj}(X, \mathcal{A}) \subseteq \Pi(\overline{X}) \times \mathbb{N}$ as
$\mathsf{proj}(X, \mathcal{A}) = \mathsf{rmc}(\{(p_{\downarrow \overline{X}}, w) \mid (p, w) \in \mathcal{A}, \forall e \in X : \forall e' \in \overline{X} : p \sqsubseteq U[ee']\})$.

**Join.** Given a set $U'$, $\mathcal{B} \subseteq \Pi(U) \times \mathbb{N}$, and $\hat{U} = U \cup U'$, we define $\mathsf{join}(\mathcal{A}, \mathcal{B}) \subseteq \Pi(\hat{U}) \times \mathbb{N}$ as
$\mathsf{join}(\mathcal{A}, \mathcal{B}) = \mathsf{rmc}(\{(p_{\uparrow \hat{U}} \sqcap q_{\uparrow \hat{U}}, w_1 + w_2) \mid (p, w_1) \in \mathcal{A}, (q, w_2) \in \mathcal{B}\})$.

▶ **Proposition 16** (Bodlaender *et al.* [2]). *Each of the operations union, insert, shift, glue, and project can be carried out in time $s \cdot |U|^{\mathcal{O}(1)}$, where $s$ is the size of the input of the operation. Given two weighted partitions $\mathcal{A}$ and $\mathcal{B}$, $\mathsf{join}(\mathcal{A}, \mathcal{B})$ can be computed in time $|\mathcal{A}| \cdot |\mathcal{B}| \cdot |U|^{\mathcal{O}(1)}$.*

Given a weighted partition $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$ and a partition $q \in \Pi(U)$, we define $\mathsf{opt}(q, \mathcal{A}) = \min\{w \mid (p, w) \in \mathcal{A}, p \sqcap q = \{U\}\}$. Given two weighted partitions $\mathcal{A}, \mathcal{A}' \subseteq \Pi(U) \times \mathbb{N}$, we say that $\mathcal{A}$ *represents* $\mathcal{A}'$ if for each $q \in \Pi(U)$, $\mathsf{opt}(q, \mathcal{A}) = \mathsf{opt}(q, \mathcal{A}')$. Given a set $Z$ and a function $f : 2^{\Pi(U) \times \mathbb{N}} \times Z \to 2^{\Pi(U) \times \mathbb{N}}$, we say that $f$ *preserves representation* if for each two weighted partitions $\mathcal{A}, \mathcal{A}' \subseteq \Pi(U) \times \mathbb{N}$ and each $z \in Z$, it holds that if $\mathcal{A}'$ represents $\mathcal{A}$ then $f(\mathcal{A}', z)$ represents $f(\mathcal{A}, z)$.

▶ **Proposition 17** (Bodlaender *et al.* [2]). *The union, insert, shift, glue, project, and join operations preserve representation.*

▶ **Theorem 18** (Bodlaender *et al.* [2]). *There exists an algorithm reduce that, given a set of weighted partitions $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$, outputs in time $|\mathcal{A}| \cdot 2^{(\omega-1)|U|} \cdot |U|^{\mathcal{O}(1)}$ a set of weighted partitions $\mathcal{A}' \subseteq \mathcal{A}$ such that $\mathcal{A}'$ represents $\mathcal{A}$ and $|\mathcal{A}'| \leq 2^{|U|}$, where $\omega$ denotes the matrix multiplication exponent.*

We now have all the tools needed to describe our algorithm. This algorithm is based on the one given in [2, Section 3.5] and $E_0 = \{\{v_0, v\} \mid v \in V(G)\}$. The role of $v_0$ is to artificially guarantee the connectivity of the solution graph, so that the machinery of Bodlaender *et al.* [2] can be applied. In the following, for each subgraph $H$ of $G$, for each $Z \subseteq V(H)$, and for each $Z_0 \subseteq E_0 \cap E(H)$, we denote by $H\langle Z, Z_0\rangle$ the graph $\left(Z, Z_0 \cup \left(E(H) \cap \binom{Z \setminus \{v_0\}}{2}\right)\right)$.

Given a nice tree decomposition of $G$ of width $w$, we define a nice tree decomposition $((T, \mathcal{X}), r, \mathcal{G})$ of $G_0$ of width $w+1$ such that the only empty bags are the root and the leaves and for each $t \in T$, if $X_t \neq \emptyset$ then $v_0 \in X_t$. Note that this can be done in linear time. For each bag $t$, each integers $i$, $j$, and $\ell$, each function $\mathbf{s} : X_t \to \{0,1\}$, each function $\mathbf{s}_0 : \{v_0\} \times \mathbf{s}^{-1}(1) \to \{0,1\}$, and each function $\mathbf{r} : E(G_t\langle \mathbf{s}^{-1}(1), \mathbf{s}_0^{-1}(1)\rangle) \to \{0,1\}$, if $C_4 \not\preceq_{\mathsf{tm}} G_t\langle \mathbf{s}^{-1}(1), \mathbf{s}_0^{-1}(1)\rangle$, we define:

$$
\begin{aligned}
\mathcal{E}_t(p, \mathbf{s}, \mathbf{s}_0, \mathbf{r}, i, j, \ell) \quad = \quad & \{(Z, Z_0) \mid (Z, Z_0) \in 2^{V_t} \times 2^{E_0 \cap E(G_t)} \\
& |Z| = i,\ |E(G_t\langle Z, Z_0\rangle)| = j,\ \mathsf{c}_3(G_t\langle Z, Z_0\rangle) = \ell, \\
& G_t\langle Z, Z_0\rangle \text{ does not contain the diamond as a subgraph}, \\
& Z \cap X_t = \mathbf{s}^{-1}(1),\ Z_0 \cap (X_t \times X_t) = \mathbf{s}_0^{-1}(1),\ v_0 \in X_t \Rightarrow \mathbf{s}(v_0) = 1, \\
& \quad \forall u \in Z \setminus X_t : \text{ either } t \text{ is the root or} \\
& \qquad \exists u' \in \mathbf{s}^{-1}(1) : u \text{ and } u' \text{ are connected in } G_t\langle Z, Z_0\rangle, \\
& \forall v_1, v_2 \in \mathbf{s}^{-1}(1) : p \sqsubseteq V_t[\{v_1, v_2\}] \Leftrightarrow v_1 \text{ and } v_2 \text{ are} \\
& \qquad \text{connected in } G_t\langle Z, Z_0\rangle, \\
& \forall e \in E(G_t\langle Z, Z_0\rangle) \cap \binom{\mathbf{s}^{-1}(1)}{2} : \mathbf{r}(e) = 1 \Leftrightarrow \ e \text{ is an} \\
& \qquad \text{edge of a } C_3 \text{ in } G_t\langle Z, Z_0\rangle\} \\
\mathcal{A}_t(\mathbf{s}, \mathbf{s}_0, \mathbf{r}, i, j, \ell) \quad = \quad & \{p \mid p \in \Pi(\mathbf{s}^{-1}(1)),\ \mathcal{E}_t(p, \mathbf{s}, \mathbf{s}_0, \mathbf{r}, i, j, \ell) \neq \emptyset\}.
\end{aligned}
$$

Otherwise, i.e., if $C_4 \preceq_{\mathsf{tm}} G_t\langle \mathbf{s}^{-1}(1), \mathbf{s}_0^{-1}(1)\rangle$, we define $\mathcal{A}_t(\mathbf{s}, \mathbf{s}_0, \mathbf{r}, i, j, \ell) = \emptyset$.

Note that we do not need to keep track of partial solutions if $C_4 \preceq_{\mathsf{tm}} G_t\langle \mathbf{s}^{-1}(1), \mathbf{s}_0^{-1}(1)\rangle$, as we already know they will not lead to a global solution. Moreover, if $C_4 \not\preceq_{\mathsf{tm}} G_t\langle \mathbf{s}^{-1}(1), \mathbf{s}_0^{-1}(1)\rangle$, then by Lemma 15 it follows that $m(G_t\langle \mathbf{s}^{-1}(1), \mathbf{s}_0^{-1}(1)\rangle) \leq \frac{3}{2}(n(G_t\langle \mathbf{s}^{-1}(1), \mathbf{s}_0^{-1}(1)\rangle) - 1)$.

Using the definition of $\mathcal{A}_r$, Lemma 14, and Lemma 15 we have that $\mathbf{tm}_{\{C_4\}}(G) \leq k$ if and only if for some $i \geq |V(G) \cup \{v_0\}| - k$ and some $j \leq \frac{2}{3}(i-1)$, we have $\mathcal{A}_r(\varnothing, \varnothing, \varnothing, i, j, 1+j-i) \neq \emptyset$. For each $t \in V(T)$, we assume that we have already computed $\mathcal{A}_{t'}$ for each children $t'$ of $t$, and in the full version we show how to compute $\mathcal{A}_t$, distinguishing several cases depending on the type of node $t$. The proof of the following theorem can also be found in the full version.

▶ **Theorem 19** (⋆). $\{C_4\}$-TM-Deletion *can be solved in time* $2^{\mathcal{O}(\mathbf{tw})} \cdot n^7$.

## 5 Superexponential lower bound for specific cases

In this section, we focus on the graph classes $\mathcal{P} = \{P_i \mid i \geq 6\}$ and $\mathcal{K}$, and we show the following theorem. Let us recall that $\mathcal{K}$ is the set containing every connected graph $G$ such that for each leaf $B \in L(\mathsf{bct}(G))$ and $r \in \mathbb{N}$, $B \not\preceq_{\mathsf{tm}} K_{2,r}$ (or $B \not\preceq_{\mathsf{m}} K_{2,r}$, which is equivalent).

▶ **Theorem 20.** *Let $\mathcal{F}$ be a proper collection such that $\mathcal{F} \subseteq \mathcal{P}$ or $\mathcal{F} \subseteq \mathcal{K}$. Unless the ETH fails, neither $\mathcal{F}$-*TM-Deletion *nor $\mathcal{F}$-*M-Deletion *can be solved in time* $2^{o(\mathbf{tw} \log \mathbf{tw})} \cdot n^{\mathcal{O}(1)}$.

In particular, this theorem implies the result of Pilipczuk [19] as a corollary. Indeed, Vertex Planarization corresponds to $\mathcal{F}$-Deletion where $\mathcal{F} = \{K_5, K_{3,3}\}$, and note that $\{K_5, K_{3,3}\} \subseteq \mathcal{K}$. Note also that Theorem 20 also implies the results stated in items **4** and **5** of the introduction, as all these graphs are easily seen to belong in $\mathcal{K}$.

▶ **Corollary 21.** *Unless the ETH fails, for each* $\mathcal{F} \in \{\{C_i\} \mid i \geq 5\} \cup \{\{K_i\} \mid i \geq 4\}$*, neither* $\mathcal{F}$-TM-Deletion *nor* $\mathcal{F}$-M-Deletion *can be solved in time* $2^{o(\mathbf{tw} \log \mathbf{tw})} \cdot n^{\mathcal{O}(1)}$.

In the following we prove Theorem 20 for $\mathcal{F}$-TM-Deletion, and we explain in the full version how to modify the proof to obtain the result for $\mathcal{F}$-M-Deletion. To prove Theorem 20, we reduce from $k \times k$ Permutation Clique ($k \times k$ P. Clique for short), defined by Lokshtanov *et al.* [17]. In this problem, we are given an integer $k$ and a graph $G$ with vertex set $[1, k] \times [1, k]$. The question is whether there is a $k$-clique in G with exactly one element from each row and exactly one element from each column. Lokshtanov *et al.* [17] proved that $k \times k$ P. Clique cannot be solved in time $2^{o(k \log k)}$ unless the ETH fails.

We now present the common part of the construction for both $\mathcal{P}$ and $\mathcal{K}$. Let $\mathcal{F}$ be a proper collection such that $\mathcal{F} \subseteq \mathcal{P}$ or $\mathcal{F} \subseteq \mathcal{K}$. Note that if $\mathcal{F} \subseteq \mathcal{P}$, then $|\mathcal{F}| = 1$. Let us fix $(H, B)$ to be an essential pair of $\mathcal{F}$. We first define some gadgets that generalize the $K_5$*-edge gadget* and the *s-choice gadget* introduced in [19]. Given a graph $G$ and two vertices $x$ and $y$ of $G$, by *introducing an* $H$*-edge gadget* between $x$ and $y$ we mean that we add a copy of $H$ where we identify the first vertex of $(H, B)$ with $y$ and the second vertex of $(H, B)$ with $x$. Using the fact that an $H$-edge gadget between two vertices $x$ and $y$ is a copy of $H$ and that $\{x, y\}$ is a cut set, we have that the $H$-edge gadgets clearly satisfy the following.

▶ **Proposition 22.** *If* $\mathcal{F}$-TM-Deletion *has a solution on* $(G, k)$ *then this solution intersects every* $H$*-edge gadget, and there exists a solution* $S$ *such that for each* $H$*-edge gadget* $A$ *between two vertices* $x$ *and* $y$*,* $V(A) \cap S \subseteq \{x, y\}$ *and* $\{x, y\} \cap S \neq \emptyset$.

In the following, we will always assume that the solution that we take into consideration is a solution satisfying the properties given by Proposition 22. Moreover, we will restrict the solution to contain only vertices of $H$-edge gadgets by setting an appropriate budget to the number of vertices we can remove from the input graph $G$.

Given a graph $G$ and two vertices $x$ and $y$ of $G$, by *introducing a* $B$*-edge gadget* between $x$ and $y$ we mean that we add a copy of $B$ where we identify the first vertex of $(H, B)$ with $y$ and the second vertex of $(H, B)$ with $x$. Given a graph $G$ and three vertices $x$, $y$, and $z$ of $G$, by *introducing a double* $H$*-edge gadget* between $x$ and $z$ through $y$ we mean that we introduce an $H$-edge gadget between $z$ and $y$, and a $B$-edge gadget between $x$ and $y$.

Given a set of $s$ vertices $\{x_i \mid i \in [1, s]\}$, by *introducing an* $H$*-choice gadget connecting* $\{x_i \mid i \in [1, s]\}$, we mean that we add $2s + 2$ vertices $z_i$, $i \in [0, 2s + 1]$, for each $i \in [0, 2s]$, we introduce an $H$-edge gadget between $z_i$ and $z_{i+1}$, and for each $i \in [1, s]$, we introduce a $B$-edge gadget between $x_i$ and $z_{2i-1}$ and another one between $x_i$ and $z_{2i}$. We see the $H$-choice gadget as a graph induced by $\{x_i \mid i \in [1, s]\} \cup \{z_i \mid i \in [0, 2s]\}$, the $B$-edge gadgets, and the $H$-edge gadgets. The following proposition is similar to [19, Lemma 5].

▶ **Proposition 23** ($\star$)**.** *For every* $H$*-choice gadget* $C$ *connecting* $\{x_i \mid i \in [1, s]\}$*, any solution* $S$ *of* $\mathcal{F}$-TM-Deletion *satisfies* $|S \cap V(C)| \geq 2s$*, for every* $i \in [1, s]$ *there exists a solution* $S$ *such that* $x_i \notin S$*, and for every solution* $S$ *with* $|S \cap V(C)| = 2s$*,* $\exists i \in [1, s]$ *such that* $x_i \notin S$.

We now start the description of the general construction. Given an instance $(G, k)$ of $k \times k$ P. Clique, we construct an instance $(G', \ell)$ of $\mathcal{F}$-TM-Deletion, which we call the *general* $H$*-construction* of $(G, k)$. We first introduce $k^2 + 2k$ vertices, namely $\{c_i \mid i \in [1, k]\}$, $\{r_i \mid i \in [1, k]\}$, and $\{t_{i,j} \mid i, j \in [1, k]\}$. For each $i, j \in [1, k]$, we add the edges $\{r_j, t_{i,j}\}$ and $\{t_{i,j}, c_i\}$. For each $j \in [1, k]$, we introduce an $H$-choice gadget connecting $\{t_{i,j} \mid i \in [1, k]\}$. This part of the construction is illustrated in the full version.

We now describe how we encode the edges of $G$ in $G'$. For each $e \in E(G)$, we define the integers $p(e)$, $\gamma(e)$, $q(e)$, and $\delta(e)$ in $[1, k]$, such that $e = \{(p(e), \gamma(e)), (q(e), \delta(e))\}$ with

$p(e) \leq q(e)$. Note that the edges $e$ with $p(e) = q(e)$ are not relevant to our construction and hence we safely forget them. For each $e \in E(G)$, we add to $G'$ three new vertices, $d^\ell_e$, $d^m_e$, and $d^r_e$, and four edges $\{d^\ell_e, c_{p(e)}\}$, $\{d^\ell_e, r_{\gamma(e)}\}$, $\{d^r_e, c_{q(e)}\}$, and $\{d^r_e, r_{\delta(e)}\}$. We introduce a double $H$-edge gadget between $d^\ell_e$ and $d^r_e$ through $d^m_e$. The encoding of an edge $e \in E(G)$ is also illustrated in the full version. For each $1 \leq p < q \leq k$, we define $E(p,q) = \{e \in E(G) \mid (p(e), q(e)) = (p,q)\}$ and we introduce an $H$-choice gadget connecting $\{d^\ell_e \mid e \in E(p,q)\}$.

For each $e \in E(G)$, we increase the size of the requested solution in $G'$ by one, the initial budget being the sum of the budget given by Proposition 23 over all the $H$-choice gadgets introduced in the construction. Because of the double $H$-edge gadget, we need to take in the solution either $d^m_e$ or both $d^\ell_e$ and $d^r_e$. The extra budget given for each edge permits to include $d^m_e$ in the solution. If the $H$-choice gadget connected to $d^\ell_e$ already chooses $d^\ell_e$ to be in the solution, then we can use the extra budget given for the edge $e$ to choose $d^r_e$ instead of $d^m_e$. In the case $d^m_e$ is chosen, in the resulting graph $c_{p(e)}$ remains connected to $r_{\gamma(e)}$ and $c_{q(e)}$ remains connected to $r_{\delta(e)}$. In the following, we consider only a solution $S$ such that either $\{d^\ell_e, d^m_e, d^r_e\} \cap S = \{d^\ell_e, d^r_e\}$ or $\{d^\ell_e, d^m_e, d^r_e\} \cap S = \{d^m_e\}$ for each $e \in E(G)$.

We set $\ell = 3|E(G)| + 2k^2$. By construction, this budget is tight and permits to take only a minimum-size solution in every $H$-choice gadget and one endpoint of each $H$-edge gadget between $d^r_e$ and $d^m_e$, $e \in E(G)$. This concludes the general $H$-construction $(G', \ell)$ of $(G, k)$.

Let us now discuss about the treewidth of $G'$. By deleting $2k$ vertices, namely the vertices $\{c_i \mid i \in [1,k]\}$ and the vertices $\{r_j \mid j \in [1,k]\}$, we obtain a graph where each connected component is an $H$-choice gadget, with eventually some pendant $H$-edge gadgets or double $H$-edge gadgets. As the treewidth of the $H$-choice gadget, the $H$-edge gadget, and the double $H$-choice gadget is linear in $|V(H)|$, we obtain that $\mathbf{tw}(G) = \mathcal{O}_d(k)$ (recall that $d = \mathsf{size}(\mathcal{F})$).

We explain in the full version that, given a permutation $\sigma : [1,k] \to [1,k]$ defining a solution of $k \times k$ P. Clique on $(G,k)$, we can define a so-called *$\sigma$-general $H$-solution* $S$ having *nice* properties. Conversely, given a set $S \subseteq V(G')$ of size at most $3|E(G)| + 2k^2$ satisfying the so-called *permutation property*, we can define (cf. the full version) a unique permutation $\sigma$ that defines a $k$-clique in $G$; we call $\sigma$ the *associated permutation* of $S$.

To conclude the reduction, we deal separately with the cases $\mathcal{F} \subseteq \mathcal{P}$ and $\mathcal{F} \subseteq \mathcal{K}$. For each such $\mathcal{F}$, we assume w.l.o.g. that $\mathcal{F}$ is a topological minor antichain, we fix $(H, B)$ to be an essential pair of $\mathcal{F}$, and given an instance $(G,k)$ of $k \times k$ P. Clique, we start from the general $H$-construction $(G', \ell)$ and add some edges and vertices in order to build an instance $(G'', \ell)$ of $\mathcal{F}$-TM-Deletion. We show that if $k \times k$ P. Clique on $(G,k)$ has a solution $\sigma$, then the $\sigma$-general $H$-solution is a solution of $\mathcal{F}$-TM-Deletion on $(G'', \ell)$. Conversely, we show that if $\mathcal{F}$-TM-Deletion on $(G'', \ell)$ has a solution $S$, then this solution satisfies the permutation property. This allows to prove that the associated permutation $\sigma$ of $S$ is a solution of $k \times k$ P. Clique on $(G,k)$. The details can be found in the full version.

## References


1 Julien Baste, Marc Noy, and Ignasi Sau. On the number of labeled graphs of bounded treewidth. *CoRR*, abs/1604.07273, 2016. To appear in *Proc. of WG 2017*.

2 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015.

3 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (meta) kernelization. *Journal of the ACM*, 63(5):44:1–44:69, 2016.

4 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. *CoRR*, abs/1704.06757, 2017.

5 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

6 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving Connectivity Problems Parameterized by Treewidth in Single Exponential Time. In *Proc. of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 150–159, 2011.

7 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar $\mathcal{F}$-Deletion: Approximation, Kernelization and Optimal FPT Algorithms. In *Proc. of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 470–479, 2012.

8 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *Journal of the ACM*, 63(4):29:1–29:60, 2016.

9 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. In *Proc. of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 503–510, 2010. Full version available at *CoRR*, abs/1606.05689, 2016.

10 Valentin Garnero, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. Explicit linear kernels via dynamic programming. *SIAM Journal on Discrete Mathematics*, 29(4):1864–1894, 2015.

11 John E. Hopcroft and Robert Endre Tarjan. Efficient algorithms for graph manipulation. *Communations of ACM*, 16(6):372–378, 1973.

12 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

13 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In *Proc. of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1802–1811, 2014.

14 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Transactions on Algorithms*, 12(2):21:1–21:41, 2016.

15 J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

16 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.

17 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. In *Proc. of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 760–776, 2011.

18 Doowon Paik, Sudhakar M. Reddy, and Sartaj Sahni. Deleting vertices to bound path length. *IEEE Trans. Computers*, 43(9):1091–1096, 1994. doi:10.1109/12.312117.

19 Marcin Pilipczuk. A tight lower bound for Vertex Planarization on graphs of bounded treewidth. *Discrete Applied Mathematics*, 231:211–216, 2017.

20 Neil Robertson and Paul D. Seymour. Graph minors. X. Obstructions to tree decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991.

# Contraction-Bidimensionality of Geometric Intersection Graphs*

**Julien Baste**[1] **and Dimitrios M. Thilikos**[2]

1 **Université de Montpellier, LIRMM, Montpellier, France**
baste@lirmm.fr

2 **AlGCo project team, CNRS, LIRMM, Montpellier, France and Department of Mathematics, National and Kapodistrian University of Athens, Greece**
sedthilk@thilikos.info

## Abstract

Given a graph $G$, we define $\mathbf{bcg}(G)$ as the minimum $k$ for which $G$ can be contracted to the uniformly triangulated grid $\Gamma_k$. A graph class $\mathcal{G}$ has the SQGC property if every graph $G \in \mathcal{G}$ has treewidth $\mathcal{O}(\mathbf{bcg}(G)^c)$ for some $1 \leq c < 2$. The SQGC property is important for algorithm design as it defines the applicability horizon of a series of meta-algorithmic results, in the framework of bidimensionality theory, related to fast parameterized algorithms, kernelization, and approximation schemes. These results apply to a wide family of problems, namely problems that are *contraction-bidimensional*. Our main combinatorial result reveals a general family of graph classes that satisfy the SQGC property and includes bounded-degree string graphs. This considerably extends the applicability of bidimensionality theory for several intersection graph classes of 2-dimensional geometrical objects.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

**Keywords and phrases** Grid exlusion theorem, Bidimensionality, Geometric intersection graphs, String Graphs

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.5

## 1 Introduction

Treewidth is one of most well-studied parameters in graph algorithms. It serves as a measure of how close a graph is to the topological structure of a tree (see Section 2 for the formal definition). Gavril is the first to introduce the concept in [28] but it obtained its name in the second paper of the Graph Minors series of Robertson and Seymour in [36]. Treewidth has extensively used in graph algorithm design due to the fact that a wide class of intractable problems in graphs becomes tractable when restricted on graphs of bounded treewidth [1, 4, 5]. Before we present some key combinatorial properties of treewidth, we need some definitions.

### 1.1 Graph contractions and minors

Our first aim is the define some parameterized versions of the contraction relation on graphs. Given a non-negative integer $c$, two graphs $H$ and $G$, and a surjection $\sigma : V(G) \to V(H)$ we write $H \leq^c_\sigma G$ if

* This work has been supported by project DEMOGRAPH (ANR-16-CE40-0028).





- for every $x \in V(H)$, the graph $G[\sigma^{-1}(x)]$ is a non-empty graph (i.e., a graph with at least one vertex) of *diameter* at most $c$ and
- for every $x, y \in V(H)$, $\{x, y\} \in E(H) \iff G[\sigma^{-1}(x) \cup \sigma^{-1}(y)]$ is connected.

We say that $H$ is a *$c$-diameter contraction* of $G$ if there exists a surjection $\sigma : V(G) \to V(H)$ such that $H \leq^c_\sigma G$ and we write this $H \leq^c G$. Moreover, if $\sigma$ is such that for every $x \in V(G)$, $|\sigma^{-1}(x)| \leq c' + 1$, then we say that $H$ is a *$c'$-size contraction* of $G$, and we write $H \leq^{(c')} G$.

## 1.2 Combinatorics of treewdith

One of the most celebrated structural results on treewidth is the following:

▶ **Proposition 1.** *There is a function $f : \mathbb{N} \to \mathbb{N}$ such that every graph excluding a $(k \times k)$-grid as a minor has treewidth at most $f(k)$.*

A proof of Proposition 1 appeared for the first time by Robertson and Seymour in [37]. Other proofs, with better bounds to the function $f$, appeared in [38] and later in [17] (see also [31, 33]). Currently, the best bound for $f$ is due to Chuzhoy, who proved in [3] that $f(k) = k^{19} \cdot \log^{\mathcal{O}(1)} k$. On the other hand, it is possible to show that Proposition 1 is not correct when $f(k) = \mathcal{O}(k^2 \cdot \log k)$ (see [41]).

The potential of Proposition 1 on graph algorithms has been capitalized by the *theory of bidimensionality* that was introduced in [9] and has been further developed in [12, 13, 8, 15, 22, 25, 30, 21, 16, 27, 23]. This theory offered general techniques for designing efficient fixed-parameter algorithms and approximation schemes for NP-hard graph problems in broad classes of graphs (see [10, 14, 20, 7, 11]). In order to present the result of this paper we first give a brief presentation of this theory and of its applicability.

## 1.3 Optimization parameters and bidimensionality

A *graph parameter* is a function $\mathbf{p}$ mapping graphs to non-negative integers. We say that $\mathbf{p}$ is a *minimization graph parameter* if $\mathbf{p}(G) = \min\{k \mid \exists S \subseteq V(G) : |S| \leq k \text{ and } \phi(G, S) = \texttt{true}\}$, where $\phi$ is a some predicate on $G$ and $S$. Similarly, we say that $\mathbf{p}$ is a *maximization graph parameter* if in the above definition we replace min by max and $\leq$ by $\geq$ respectivelly. Minimization or maximization parameters are briefly called *optimization parameters*.

Given two graphs $G$ and $H$, if there exists an integer $c$ such that $H \leq^c G$, then we say that $H$ is a *contraction* of $G$, and we write $H \leq G$. Moreover, if there exists a subgraph $G'$ of $G$ such that $H \leq G'$, we say that $H$ is a *minor* of $G$ and we write this $H \preceq G$. A graph parameter $\mathbf{p}$ is *minor-closed* (resp. *contraction-closed*) when $H \preceq G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$ (resp. $H \leq G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$). We can now give the two following definitions:

$\mathbf{p}$ is *minor-bidimensional* if

- $\mathbf{p}$ is minor-closed, and
- $\exists k_0 \in \mathbb{N} : \forall k \geq k_0, \frac{\mathbf{p}(\boxplus_k)}{k^2} \geq \delta$

$\mathbf{p}$ is *contraction-bidimensional* if

- $\mathbf{p}$ is contraction-closed, and
- $\exists k_0 \in \mathbb{N} : \forall k \geq k_0, \frac{\mathbf{p}(\Gamma_k)}{k^2} \geq \delta$

for some $\delta > 0$. In the above definitions, we use $\boxplus_k$ for the $(k \times k)$-grid and $\Gamma_k$ for the uniformly triangulated $(k \times k)$-grid (see Figure 1). If $\mathbf{p}$ is a minimization (resp. maximization) graph parameter, we denote by $\Pi_{\mathbf{p}}$ the problem that, given a graph $G$ and a non-negative integer $k$, asks whether $\mathbf{p}(G) \leq k$ (resp. $\mathbf{p}(G) \geq k$). We say that a problem is *minor/contraction-bidimensional* if it is $\Pi_{\mathbf{p}}$ for some bidimensional optimization parameter $\mathbf{p}$.

A (non exhaustive) list of minor-bidimensional problems is: Vertex Cover, Feedback Vertex Set, Longest Cycle, Longest Path, Cycle Packing, Path Packing, Diamond Hitting Set, Minimum Maximal Matching, Face Cover, and Max Bounded

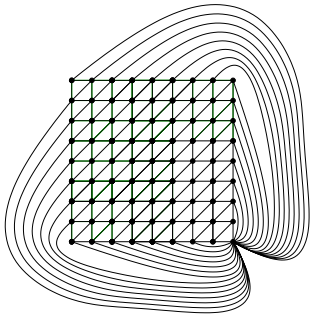


**Figure 1** The graph $\Gamma_9$.

Degree Connected Subgraph. Some problems that are contraction-bidimensional (but not minor-bidimensional) are Connected Vertex Cover, Dominating Set, Connected Dominating Set, Connected Feedback Vertex Set, Induced Matching, Induced Cycle Packing, Cycle Domination, Connected Cycle Domination, $d$-Scattered Set, Induced Path Packing, $r$-Center, connected $r$-Center, Connected Diamond Hitting Set, Unweighted TSP Tour.

## 1.4 Subquadratic grid minor/contraction property

In order to present the meta-algorithmic potential of bidimensionality theory we need to define some *property on graph classes* that defines the horizon of its applicability. Let $\mathcal{G}$ be a graph class. We say that $\mathcal{G}$ has the *subquadratic grid minor property* (SQGM property for short) if there exist a constant $1 \leq c < 2$ such that every graph $G \in \mathcal{G}$ which excludes $\boxplus_t$ as a minor, for some integer $t$, has treewidth $\mathcal{O}(t^c)$. In other words, this property holds for $\mathcal{G}$ if Proposition 1 can be proven for a sub-quadratic $f$ on the graphs of $\mathcal{G}$.

Similarly, we say that $\mathcal{G}$ has the *subquadratic grid contraction property* (SQGC property for short) if there exist a constant $1 \leq c < 2$ such that every graph $G \in \mathcal{G}$ which excludes $\Gamma_t$ as a contraction, for some integer $t$, has treewidth $\mathcal{O}(t^c)$. For brevity we say that $\mathcal{G} \in \texttt{SQGM}(c)$ (resp. $\mathcal{G} \in \texttt{SQGC}(c)$) if $\mathcal{G}$ has the SQGM (resp SQGC) property for $c$. Notice that $\texttt{SQGC}(c) \subseteq \texttt{SQGM}(c)$ for every $1 \leq c < 2$.

## 1.5 Algorithmic implications

The meta-algorithmic consequences of bidimensionality theory are summarised as follows. Let $\mathcal{G} \in \texttt{SQGM}(c)$, for $1 \leq c < 2$, and let $\mathbf{p}$ be a minor-bidimensional-optimization parameter.

**[A]** As it was observed in [9], the problem $\Pi_{\mathbf{p}}$ can be solved in $2^{o(k)} \cdot n^{\mathcal{O}(1)}$ steps on $\mathcal{G}$, given that the computation of $\mathbf{p}$ can be done in $2^{\mathbf{tw}(G)} \cdot n^{\mathcal{O}(1)}$ steps (here $\mathbf{tw}(G)$ is the treewidth of the input graph $G$). This last condition can be implied by a purely meta-algorithmic condition that is based on some variant of *Modal Logic* [35]. There is a wealth of results that yield the last condition for various optimization problems either in classes satisfying the SQGM propety [40, 19, 18, 18, 39] or to general graphs [6, 2, 24].

**[B]** As it was shown in [25] (see also [26]), when the predicate $\phi$ can be expressed in Counting Monadic Second Order Logic (CMSOL) and $\mathbf{p}$ satisfies some additional combinatorial property called *separability*, then the problem $\Pi_{\mathbf{p}}$ admits a *linear kernel*, that is a polynomial-time algorithm that transforms $(G, k)$ to an equivalent instance $(G', k')$ of $\Pi_{\mathbf{p}}$ where $G'$ has size $\mathcal{O}(k)$ and $k' \leq k$.

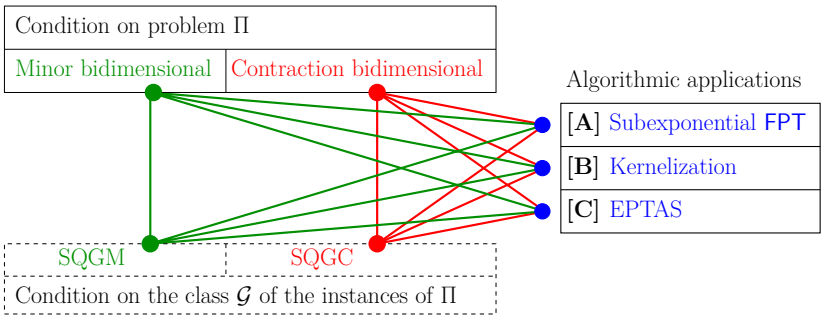



**Figure 2** The applicability of bidimensionality theory.

**[C]** It was proved in [22], that the problem of computing $\mathbf{p}(G)$ for $G \in \mathcal{G}$ admits a *Efficient Polynomial Approximation Scheme* (EPTAS) — that is an $\epsilon$-approximation algorithm running in $f(\frac{1}{\epsilon}) \cdot n^{\mathcal{O}(1)}$ steps — given that $\mathcal{G}$ is hereditary and $\mathbf{p}$ satisfies the separability property and some reducibility property (related to CMSOL expresibility).

All above results have their counterparts for *contraction-bidimensional* problems with the difference that one should instead demand that $\mathcal{G} \in \texttt{SQG}\mathbf{C}(c)$. Clearly, the applicability of all above results is delimited by the SQGM/SQGC property. This is schematically depicted in Figure 2, where the green-triangles triangles indicate the applicability of minor-bidimensionality and the red triangle indicate the applicability of contraction-bidimensionality. The aforementioned $\Omega(k^2 \cdot \log k)$ lower bound to the function $f$ of Proposition 1, indicates that $\texttt{SQG}\mathbf{M}(c)$ does not contain all graphs (given that $c < 2$). The emerging direction of research is to detect the most general classes in $\texttt{SQG}\mathbf{M}(c)$ and $\texttt{SQG}\mathbf{C}(c)$. We denote by $\mathcal{G}_H$ the class of graphs that exclude $H$ as a minor. Concerning the SQGM property, the following result was proven in [14].

▶ **Proposition 2.** *For every graph $H$, $\mathcal{G}_H \in \texttt{SQG}\mathbf{M}(1)$.*

A graph $H$ is an *apex graph* if it contains a vertex whose removal from $H$ results to a planar graph. For for the SQGC property, the following counterpart of Proposition 2 was proven in [21].

▶ **Proposition 3.** *For every apex graph $H$, $\mathcal{G}_H \in \texttt{SQG}\mathbf{C}(1)$.*

Notice that both above results concern graph classes that are defined by excluding some graph as a minor. For such graphs, Proposition 3 is indeed optimal. To see this, consider $K_h$-minor free graphs where $h \geq 6$ (these graphs are not apex graphs). Such classes do not satisfy the SQGC property: take $\Gamma_k$, add a new vertex, and make it adjacent, with all its vertices. The resulting graph excludes $\Gamma_k$ as a contraction and has treewidth $> k$.

## 1.6 String graphs

An important step extending the applicability of bidimensionality theory further than $H$-minor free graphs, was done in [23]. *Unit disk* graphs are intersections graphs of unit disks in the plane and *map* graphs are intersection graphs of face boundaries of planar graph embeddings. We denote by $\mathcal{U}_d$ the set of unit disk graphs (resp. of $\mathcal{M}_d$ map graphs) of maximum degree $d$. The following was proved in [23].

▶ **Proposition 4.** *For every positive integer $d$, $\mathcal{U}_d \in \texttt{SQG}\mathbf{M}(1)$ and $\mathcal{M}_d \in \texttt{SQG}\mathbf{M}(1)$.*

Proposition 4 was further extended for intersection graphs of more general geometric objects (in 2 dimensions) in [30]. To explain the results of [30] we need to define a more general model of intersection graphs.

Let $\mathcal{L} = \{L_1, \ldots, L_k\}$ be a collection of lines in the plane. We say that $\mathcal{L}$ is *normal* if there is no point belonging to more than two lines. The *intersection graph* $G_{\mathcal{L}}$ of $\mathcal{L}$, is the graph whose vertex set is $\mathcal{L}$ and where, for each $i, j$ where $1 \leq i < j \leq k$, the edge $\{L_i, L_j\}$ has multiplicity $|L_1 \cap L_2|$. We denote by $\mathcal{S}_d$ the set containing every graph $G_{\mathcal{L}}$ where $\mathcal{L}$ is a normal collection of lines in the plane and where each vertex of $G_{\mathcal{L}}$ has edge-degree at most $d$. i.e., is incident to at most $d$ edges. We call $\mathcal{S}_d$ *string graphs with edge-degree bounded by* $d$. It is easy to observe that $\mathcal{U}_d \cup \mathcal{M}_d \subseteq \mathcal{S}_{f(d)}$ for some quadratic function $f$. Moreover, apart from the classes considered in [23], $\mathcal{S}_d$ includes a much wider variety of classes of intersection graphs [30]. As an example, consider $\mathcal{C}_{d,\alpha}$ as the class of all graphs that are intersection graphs of $\alpha$-convex 2-dimensional bodies[1] in the plane and have degree at most $d$. In [30], it was proven that $\mathcal{C}_{d,\alpha} \subseteq \mathcal{S}_c$ where $c$ depends (polynomially) on $d$ and $\alpha$ (see [34] for other examples of classes included in $\mathcal{S}_d$).

Given a class of graph $\mathcal{G}$ and two integers $c_1$ and $c_2$, we define $\mathcal{G}^{(c_1,c_2)}$ as the set containing every graph $H$ such that there exist a graph $G \in \mathcal{G}$ and a graph $J$ that satisfy $G \leq^{(c_1)} J$ and $H \leq^{c_2} J$. Keep in mind that $\mathcal{G}^{(c_1,c_2)}$ and $\mathcal{G}^{(c_2,c_1)}$ are different graph classes. We also denote by $\mathcal{P}$ the class of all planar graphs. Using this notation, the two combinatorial results in [30] can be rewritten as follows:

▶ **Proposition 5.** *Let $c_1$ and $c_2$ be two positive integers. If $\mathcal{G} \in \mathtt{SQGC}(c)$ for some $1 \leq c < 2$, then $\mathcal{G}^{(c_1,c_2)} \in \mathtt{SQGM}(c)$.*

▶ **Proposition 6.** *For every $d \in \mathbb{N}$, $\mathcal{S}_d \subseteq \mathcal{P}^{(1,d)}$.*

Proposition 2, combined with Proposition 5, provided the wider, so far, framework on the applicability of minor-bidimensionality: $\mathtt{SQGM}(1)$ contains $\mathcal{G}_H^{(c_1,c_2)}$ for every apex graph $H$ and positive integers $c_1, c_2$. As $\mathcal{P} \in \mathtt{SQGC}(1)$ (by, e.g., Proposition 3), Propositions 5 and 6 directly classifies in $\mathtt{SQGM}(1)$ the graph class $\mathcal{S}_d$, and therefore a large family of bounded degree intersection graphs (including $\mathcal{U}_d$ and $\mathcal{M}_d$). As a result of this, the applicability of bidimensionality theory for minor-bidimensional problems has been extended to much wider families (not necessarily minor-closed) of graph classes of geometric nature.

## 1.7 Our contribution

Notice that Proposition 5 exhibits some apparent "lack of symmetry" as the assumption is "qualitatively stronger" than the conclusion. This does not permit the application of bidimensionality for *contraction*-bidimensional parameters on classes further than those of apex-minor free graphs. In other words, the results in [30] covered, for the case of $\mathcal{S}_d$, the green triangles in Figure 2 but left the red triangles open. The main result of this paper is to fill this gap by proving the following extension of Proposition 5:

▶ **Theorem 7.** *Let $c_1$ and $c_2$ be two positive integers. If $\mathcal{G} \in \mathtt{SQGC}(c)$ for some $1 \leq c < 2$, then $\mathcal{G}^{(c_1,c_2)} \in \mathtt{SQGC}(c)$.*

[1] We call a set of points in the plane a *2-dimensional body* if it is homeomorphic to the closed disk $\{(x, y) \mid x^2 + y^2 \leq 1\}$. A 2-dimensional body $B$ is a *$\alpha$-convex* if every two points can be the extremes of a line $L$ consisting of $\alpha$ straight lines and where $L \subseteq B$.

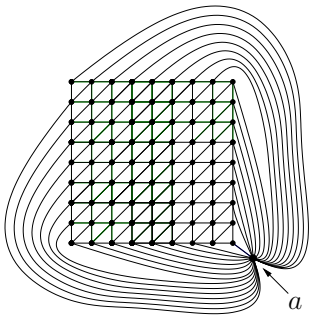



**Figure 3** The uniformly triangulated grid $\hat{\Gamma}_9$.

Combining Proposition 3 and Theorem 7 we extend the applicability horizon of contraction-bidimensionality further than apex-minor free graphs: $\texttt{SQGC}(1)$ contains $\mathcal{G}_H^{(c_1,c_2)}$ for every apex graph $H$ and positive integers $c_1, c_2$. As a special case of this, we have that $\mathcal{S}_d \in \texttt{SQGC}(1)$. Therefore, on $\mathcal{S}_d$, the results described in Subsection 1.5 apply for contraction-bidimensional problems as well (such as those enumerated in the end of Subsection 1.3).

This paper is organized as follows. In Section 2, we give the necessary definitions and some preliminary results. Section 3 is dedicated to the proof of Theorem 7. We should stress that this proof is quite different than the one of Proposition 5 in [30]. Finally, Section 4 contains some discussion and open problems.

## 2 Definitions and preliminaries

All graphs in this paper are undirected, loop-less, and may have multiple edges. If a graph has no multiple edges, we call it *simple*. Given a graph $G$, we denote by $V(G)$ its vertex set and by $E(G)$ its edge set. Let $x$ be a vertex or an edge of a graph $G$ and likewise for $y$; their *distance* in $G$, denoted by $\mathbf{dist}_G(x, y)$, is the smallest number of vertices of a path in $G$ that contains them both. Moreover if $G$ is a graph and $x \in V(G)$, we denote by $N_G^c(x)$ the set $\{y \mid y \in V(G),\ \mathbf{dist}_G(x, y) \leq c + 1\}$. For any set of vertices $S \subseteq V(G)$, we denote by $G[S]$ the subgraph of $G$ induced by the vertices from $S$. If $G[S]$ is connected, then we say that $S$ is a *connected vertex set* of $G$. We define the *diameter* of a connected subset $S$ as the maximum pairwise distance between any two vertices of $S$. The *edge-degree* of a vertex $v \in V(G)$ is the number of edges that are incident to it (multi-edges contribute with their multiplicity to this number).

For our proofs, we also need the graph $\hat{\Gamma}_k$ that is the variant of $\Gamma_k$, depicted in Figure 3. Notice that $\Gamma_k$ and $\hat{\Gamma}_k$ are both triangulated plane graphs, i.e., all their faces are triangles. In $\hat{\Gamma}_k$, we refer to the vertex $a$ (as in Figure 3) as the *apex vertex* of $\hat{\Gamma}_k$. (We avoid the formal definitions of $\boxplus_k$, $\Gamma_k$, $\hat{\Gamma}_k$ in this extended abstract – see [21] for a more precise formalism.) In each of these graphs we denote the vertices of the underlying grid by their coordinates $(i, j) \in [0, k-1]^2$ agreeing that the upper-left corner is the vertex $(0, 0)$.

### 2.1 Treewidth

A *tree-decomposition* of a graph $G$, is a pair $(T, \mathcal{X})$, where $T$ is a tree and $\mathcal{X} = \{X_t : t \in V(T)\}$ is a family of subsets of $V(G)$, called *bags*, such that the following three properties are satisfied:

- $\bigcup_{t \in V(T)} X_t = V(G)$,
- for every edge $e \in E(G)$ there exists $t \in V(T)$ such that $e \subseteq X_t$, and

- $\forall v \in V(G)$, the set $T_v = \{t \in V(T) \mid v \in X_t\}$ is a connected vertex set of $T$.

The *width* of a tree-decomposition is the cardinality of the maximum size bag minus 1 and the *treewidth* of a graph $G$ is the minimum width over all the tree-decompositions of $G$. We denote the treewidth of $G$ by $\mathbf{tw}(G)$.

▶ **Lemma 8.** *Let $G$ be a graph and let $H$ be a $c$-size contraction of $G$. Then $\mathbf{tw}(G) \leq (c+1) \cdot (\mathbf{tw}(H)+1) - 1$.*

## 3 Proof of Theorem 7

Let $H$ and $G$ be graphs and $c$ be a non-negative integer. If $H \leq_\sigma^c G$, then we say that $H$ is a *$\sigma$-contraction* of $G$, and denote this by $H \leq_\sigma G$.

Before we proceed the the proof of Theorem 7 we make first the following three observations. (In all statements, we assume that $G$ and $H$ are two graphs and $\sigma : V(G) \to V(H)$ such that $H$ is a $\sigma$-contraction of $G$.)

▶ **Observation 9.** *Let $S$ be a connected subset of $V(H)$. Then the set $\bigcup_{x \in S} \sigma^{-1}(x)$ is connected in $G$.*

▶ **Observation 10.** *Let $S_1 \subseteq S_2 \subseteq V(H)$. Then $\sigma^{-1}(S_1) \subseteq \sigma^{-1}(S_2) \subseteq V(G)$.*

▶ **Observation 11.** *Let $S$ be a connected subset of $V(G)$. Then the diameter of $\sigma(S)$ in $H$ is at most the diameter of $S$ in $G$.*

Given a graph $G$ and $S_1, S_2 \subseteq V(G)$ we say that $S_1$ and $S_2$ *touch* if either $S_1 \cap S_2 \neq \emptyset$ or there is an edge of $G$ with one endpoint in $S_1$ and the other in $S_2$.

We say that a collection $\mathcal{R}$ of paths of a graph is *internally disjoint* if none of the internal vertices, i.e., none of the vertex of degree 2, of some path in $\mathcal{R}$ is a vertex of some other path in $\mathcal{R}$. Let $\mathcal{A}$ be a collection of subsets of $V(G)$. We say that $\mathcal{A}$ is a *connected packing* of $G$ if its elements are connected and pairwise disjoint. If additionally $\mathcal{A}$ is a partition of $V(G)$, then we say that $\mathcal{A}$ is a *connected partition* of $G$ and if, additionally, all its elements have diameter bounded by some integer $c$, then we say that $\mathcal{A}$ is a *$c$-diameter partition* of $G$.

### 3.1 Λ-state configurations

Let $G$ be a graph. Let $\Lambda = (\mathcal{W}, \mathcal{E})$ be a graph whose vertex set is a connected packing of $G$, i.e., its vertices are connected subsets of $V(G)$. A *$\Lambda$-state configuration* of a graph $G$ is a quadruple $\mathcal{S} = (\mathcal{X}, \alpha, \mathcal{R}, \beta)$ where

1. $\mathcal{X}$ is a connected packing of $G$,
2. $\alpha$ is a bijection from $\mathcal{W}$ to $\mathcal{X}$ such that for every $W \in \mathcal{W}$, $W \subseteq \alpha(W)$,
3. $\mathcal{R}$ is a collection of internally disjoint paths of $G$, and
4. $\beta$ is a bijection from $\mathcal{E}$ to $\mathcal{R}$ such that if $\{W_1, W_2\} \in \mathcal{E}$ then the endpoints of $\beta(\{W_1, W_1\})$ are in $W_1$ and $W_2$ and $V(\beta(\{W_1, W_2\})) \subseteq \alpha(W_1) \cup \alpha(W_2)$.

A $\Lambda$-state configuration $\mathcal{S} = (\mathcal{X}, \alpha, \mathcal{R}, \beta)$ of $G$ is *complete* if $\mathcal{X}$ is a partition of $V(G)$. We refer to the elements of $\mathcal{X}$ as the *states* of $\mathcal{S}$ and to the elements of $\mathcal{R}$ as the *freeways* of $\mathcal{S}$. We define $\mathsf{indep}(\mathcal{S}) = V(G) \setminus \bigcup_{X \in \mathcal{X}} X$. Note that if $\mathcal{S}$ is a $\Lambda$-state configuration of $G$, $\mathcal{S}$ is complete if and only if $\mathsf{indep}(\mathcal{S}) = \emptyset$.

Let $\mathcal{A}$ be a $c$-diameter partition of $G$. We refer to the sets of $\mathcal{A}$ as the *$\mathcal{A}$-clouds* of $G$. We define $\mathsf{front}_{\mathcal{A}}(\mathcal{S})$ as the set of all $\mathcal{A}$-clouds of $G$ that are not subsets of some $X \in \mathcal{X}$. Given a $\mathcal{A}$-cloud $C$ and a state $X$ of $\mathcal{S}$ we say that $C$ *shadows* $X$ if $C \cap X \neq \emptyset$. The *coverage* $\mathsf{cov}_{\mathcal{S}}(C)$ of an $\mathcal{A}$-cloud $C$ of $G$ is the number of states of $\mathcal{S}$ that are shadowed by $C$. A $\Lambda$-state configuration $\mathcal{S} = (\mathcal{X}, \alpha, \mathcal{R}, \beta)$ of $G$ is *$\mathcal{A}$-normal* if its satisfies the following conditions:

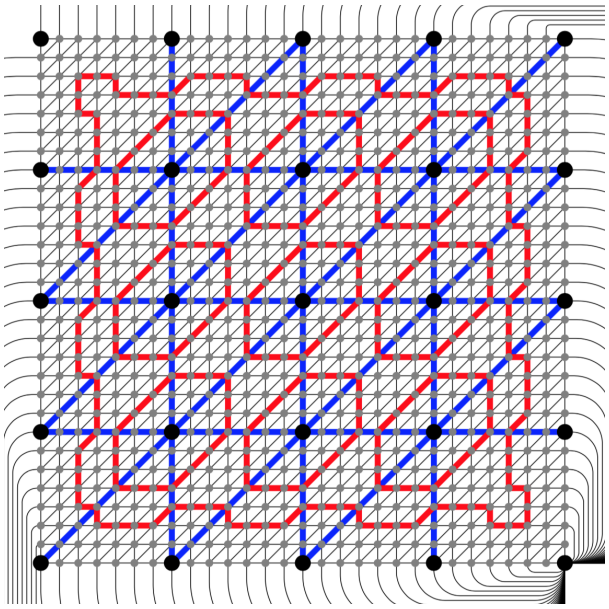


**Figure 4** A visualization of the proof of Lemma 12. In this whole graph $\Gamma_k$, we initialize our reaserch of $\hat{\Gamma}_{k'}$ such that every internal red hexagon will become a vertex of $\hat{\Gamma}_{k'}$ and correspond to a state and the border, also circle by a red line will become the vertex $b_{\text{out}}$. The blue edges correspond to the freeways. Red cycles correspond to the boundaries of the starting countries. Blue paths between big-black vertices are the freeways. Big-black vertices are the capitals.

**(A)** If a $\mathcal{A}$-cloud $C$ intersects some $W \in \mathcal{W}$, then $C \subseteq \alpha(W)$.

**(B)** If a $\mathcal{A}$-cloud over $\mathcal{S}$ intersects the vertex set of at least two freeways of $\mathcal{S}$, then it shadows at most one state of $\mathcal{S}$.

We define $\mathsf{cost}_{\mathcal{A}}(\mathcal{S}) = \sum_{C \in \mathsf{front}_{\mathcal{A}}(\mathcal{S})} \mathsf{cov}_{\mathcal{S}}(C)$. Given $S_1 \subseteq S_2 \subseteq V(G)$ where $S_1$ is connected, we define $\mathsf{cc}_G(S_2, S_1)$ as the (unique) connected component of $G[S_2]$ that contains $S_1$.

## 3.2 Triangulated grids inside triangulated grids

▶ **Lemma 12.** *Let $G$ and $H$ be graphs and $c, k$ be non-negative integers such that $H \leq^c G$ and $\Gamma_k \leq G$. Then $\Gamma_{k'} \leq H$ where $k' = \lfloor \frac{k-1}{2c+1} \rfloor - 1$.*

**Proof.** Let $k^* = 1 + (2c+1) \cdot (k'+1)$ and observe that $k^* \leq k$, therefore $\Gamma_{k^*} \leq \Gamma_k \leq G$. For simplicity we use $\Gamma = \Gamma_{k^*}$. Let $\phi : V(G) \to V(H)$ such that $H \leq^c_\phi G$ and let $\sigma : V(G) \to V(\Gamma)$ such that $\Gamma \leq_\sigma G$. We define $\mathcal{A} = \{\phi^{-1}(a) \mid a \in V(H)\}$. Notice that $\mathcal{A}$ is a $c$-diameter partition of $G$.

For each $(i,j) \in [\![0, k'+1]\!]^2$, we define $b_{i,j}$ to be the vertex of $\Gamma$ with coordinate $(i(2c+1), j(2c+1))$. We set $Q_{\text{in}} = \{b_{i,j} \mid (i,j) \in [\![1, k']\!]^2\}$ and $Q_{\text{out}} = \{b_{i,j} \mid (i,j) \in [\![0, k'+1]\!]^2\} \setminus Q_{\text{in}}$. Let also $Q = Q_{\text{in}} \cup \{b_{\text{out}}\}$ were $b_{\text{out}}$ is a new element that does not belong in $Q_{\text{in}}$. Here $b_{\text{out}}$ can be seen as a vertex that "represents" all vertices in $Q_{\text{out}}$.

Let $q, p$ be two different elements of $Q$. We say that $q$ and $p$ are *linked* if they both belong in $Q_{\text{in}}$ and their distance in $\Gamma$ is $2c+1$ or one of them is $b_{\text{out}}$ and the other is $b_{i,j}$ where $i \in \{1, k'\}$ or $j \in \{1, k'\}$.

For each $q \in Q_{\text{in}}$, we define $W_q = \sigma^{-1}(q)$. $W_q$ is connected by the definition of $\sigma$. In case $q = b_{\text{out}}$ we define $W_q = \bigcup_{q' \in Q_{\text{out}}} \sigma^{-1}(q')$. Note that as $Q_{\text{out}}$ is a connected set of $\Gamma$, then, by Observation 9, $W_{b_{\text{out}}}$ is connected in $G$. We also define $\mathcal{W} = \{W_q \mid q \in Q\}$. Given

some $q \in Q$ we call $W_q$ the *$q$-capital* of $G$ and a subset $S$ of $V(G)$ is a *capital* of $G$ if it is the $q$-capital for some $q \in Q$. Notice that $\mathcal{W}$ is a connected packing of $V(G)$.

Let $q \in Q$. If $q \in Q_{\text{in}}$ then we set $N_q = N^c_\Gamma(q)$. If $q = b_{\text{out}}$, then we set $N_q = \bigcup_{q' \in Q_{\text{out}}} N^c_\Gamma(q')$. Note that for every $q \in Q$, $N_q \subseteq V(\Gamma)$. For every $q \in Q$, we define $X_q = \sigma^{-1}(N_q)$. Note that $X_q \subseteq V(G)$. We also set $\mathcal{X} = \{X_q \mid q \in Q\}$. Let $q$ and $p$ we two linked elements of $Q$. If both $q$ and $p$ belong to $Q_{\text{in}}$, and therefore are vertices of $\Gamma$, then we define $Z_{p,q}$ as the unique shortest path between them in $\Gamma$. If $p = b_{\text{out}}$ and $q \in Q_{\text{in}}$, then we know that $q = b_{i,j}$ where $i \in \{1, k'\}$ or $j \in \{1, k'\}$. In this case we define $Z_{p,q}$ as any shortest path in $\Gamma$ between $b_{i,j}$ and the vertices in $Q_{\text{out}}$. In both cases, we define $P_{p,q}$ by picking some path between $W_p$ and $W_q$ in $G[\sigma^{-1}(V(Z_{p,q}))]$ such that $|V(P_{p,q}) \cap W_q| = 1$ and $|V(P_{p,q}) \cap W_p| = 1$.

Let $\mathcal{E} = \{\{W_p, W_q\} \mid p \text{ and } q \text{ are linked}\}$ and let $\Lambda = (\mathcal{W}, \mathcal{E})$. Notice that $\Lambda$ is isomorphic to $\hat{\Gamma}_{k'}$ and consider the isomorphism that correspond each vertex $q = b_{i,j}$, $i, j \in [\![1, k']\!]^2$ to the vertex with coordinates $(i, j)$. Moreover $b_{\text{out}}$ corresponds to the apex vertex of $\hat{\Gamma}_{k'}$.

Let $\alpha : \mathcal{W} \to \mathcal{X}$ such that for every $q \in Q$, $\alpha(W_q) = X_q$. Let also $\mathcal{R} = \{P_{p,q} \mid p, q \in Q,\ p \text{ and } q \text{ are linked}\}$. We define $\beta : \mathcal{E} \to \mathcal{R}$ such that if $q$ and $p$ are linked, then $\beta(W_q, W_p) = P_{p,q}$. We use notation $\mathcal{S} = (\mathcal{X}, \alpha, \mathcal{R}, \beta)$.

▶ **Claim 13.** *$\mathcal{S}$ is an $\mathcal{A}$-normal $\Lambda$-state configuration of $G$.*

The proof of Claim 13 is omitted in this extended abstract.

We define bellow three ways to transform a $\Lambda$-state configuration of $G$. In each of them, $\mathcal{S} = (\mathcal{X}, \alpha, \mathcal{R}, \beta)$ is an $\mathcal{A}$-normal $\Lambda$-state configuration of $G$ and $C$ is an $\mathcal{A}$-cloud in $\mathsf{front}_{\mathcal{A}}(\mathcal{S})$.

1. The *expansion procedure* applies when $C$ intersects at least two freeways of $\mathcal{S}$. Let $X$ be the state of $\mathcal{S}$ shadowed by $C$ (this state is unique because of property (B) of $\mathcal{A}$-normality). We define $(\mathcal{X}', \alpha', \mathcal{R}', \beta') = \mathsf{expand}(\mathcal{S}, C)$ such that
   - $\mathcal{X}' = \mathcal{X} \setminus \{X\} \cup \{X \cup C\}$,
   - for each $W \in \mathcal{W}$, $\alpha'(W) = X'$ where $X'$ is the unique set of $\mathcal{X}'$ such that $W \subseteq X'$,
   - $\mathcal{R}' = \mathcal{R}$, and $\beta' = \beta$.
2. The *clash procedure* applies when $C$ intersects *exactly* one freeway $P$ of $\mathcal{S}$. Let $X_1, X_2$ be the two states of $\mathcal{S}$ that intersect this freeway. Notice that $P = \beta(\alpha^{-1}(X_1), \alpha^{-1}(X_2))$, as it is the only freeway with vertices in $X_1$ and $X_2$. Assume that $(C \cap V(P)) \cap X_1 \neq \emptyset$ (if, not, then swap the roles of $X_1$ and $X_2$). We define $(\mathcal{X}', \alpha', \mathcal{R}', \beta') = \mathsf{clash}(\mathcal{S}, C)$ as follows:
   - $\mathcal{X}' = \{X_1 \cup C\} \cup \bigcup_{X \in \mathcal{X} \setminus \{X_1\}} \{\mathsf{cc}_G(X \setminus C, \alpha^{-1}(X))\}$ (notice that $\alpha^{-1}(X) \subseteq X \setminus C$, for every $X \in \mathcal{X}$, because of property (A) of $\mathcal{A}$-normality),
   - for each $W \in \mathcal{W}$, $\alpha'(W) = X'$ where $X'$ is the unique set of $\mathcal{X}'$ such that $W \subseteq X'$,
   - $\mathcal{R}' = \mathcal{R} \setminus \{P\} \cup \{P'\}$, where $P' = P_1 \cup P^* \cup P_2$ is defined as follows: let $s_i$ be the first vertex of $C$ that we meet while traversing $P$ when starting from its endpoint that belongs in $W_i$ and let $P_i$ the subpath of $P$ that we traversed that way, for $i \in \{1, 2\}$. We define $P^*$ by taking any path between $s_1$ and $s_2$ inside $G[C]$, and
   - $\beta' = \beta \setminus \{(\{W_1, W_2\}, P)\} \cup \{\{W_1, W_2\}, P'\}$.
3. The *annex procedure* applies when $C$ intersects no freeway of $\mathcal{S}$ and *touches* some country $X \in \mathcal{X}$. We define $(\mathcal{X}', \alpha', \mathcal{R}', \beta') = \mathsf{anex}(\mathcal{S}, C)$ such that
   - $\mathcal{X}' = \{X_1 \cup C\} \cup \bigcup_{X \in \mathcal{X} \setminus \{X_1\}} \{\mathsf{cc}_G(X \setminus C, \alpha^{-1}(X))\}$ (notice that $\alpha^{-1}(X) \subseteq X \setminus C$, for every $X \in \mathcal{X}$, because of property (A) of $\mathcal{A}$-normality),
   - for each $W \in \mathcal{W}$, $\alpha'(W) = X'$ where $X'$ is the unique set of $\mathcal{X}'$ such that $W \subseteq X'$,
   - $\mathcal{R}' = \mathcal{R}$, and $\beta' = \beta$.

▶ **Claim 14.** *Let $\mathcal{S} = (\mathcal{X}, \alpha, \mathcal{R}, \beta)$ be an $\mathcal{A}$-normal $\Lambda$-state configuration of $G$, and $C \in \mathsf{front}_{\mathcal{A}}(\mathcal{S})$. Let $\mathcal{S}' = \mathsf{action}(\mathcal{S}, C)$ where $\mathsf{action} \in \{\mathsf{expand}, \mathsf{clash}, \mathsf{anex}\}$. Then $\mathcal{S}'$ is an $\mathcal{A}$-normal $\Lambda$-state configuration of $G$ where $\mathsf{cost}(\mathcal{S}', \mathcal{A}) \leq \mathsf{cost}(\mathcal{S}, \mathcal{A})$. Moreover, if $\mathsf{cov}_{\mathcal{S}}(C) \geq 1$, then $\mathsf{cost}(\mathcal{S}', \mathcal{A}) < \mathsf{cost}(\mathcal{S}, \mathcal{A})$ and if $\mathsf{cov}_{\mathcal{S}}(C) = 0$ (which may be the case only when $\mathsf{action} = \mathsf{anex}$), then $|\mathsf{indep}(\mathcal{S}')| < |\mathsf{indep}(\mathcal{S})|$.*

The proof of Claim 14 is omitted in this extended abstract.

To continue with the proof of Lemma 12 we explain how to transform the $\mathcal{A}$-normal $\Lambda$-state configuration $\mathcal{S}$ of $G$ to a complete one. This is done in two phases. First, as long as there is an $\mathcal{A}$-cloud $C \in \mathsf{front}(\mathcal{S})$ where $\mathsf{cov}_{\mathcal{S}}(C) \geq 1$, we apply one of the above three procedures depending on the number of freeways intersected by $C$. We again use $\mathcal{S}$ to denote the $\mathcal{A}$-normal $\Lambda$-state configuration of $G$ that is created in the end of this first phase. Notice that, as there is no $\mathcal{A}$-cloud with $\mathsf{cov}_{\mathcal{S}}(C) \geq 1$, then $\mathsf{cost}_{\mathcal{A}}(\mathcal{S}) = 0$. The second phase is the application of $\mathsf{anex}(\mathcal{S}, C)$, as long as some $C \in \mathsf{front}_{\mathcal{A}}(\mathcal{S})$ is touching some of the countries of $\mathcal{S}$. We claim that this procedure will be applied as long as there are vertices in $\mathsf{indep}(\mathcal{S})$. Indeed, if this is the case, the set $\mathsf{front}_{\mathcal{A}}(\mathcal{S})$ is non-empty and by the connectivity of $G$, there is always a $C \in \mathsf{front}_{\mathcal{A}}(\mathcal{S})$ that is touching some country of $\mathcal{S}$. Therefore, as $\mathsf{cost}_{\mathcal{A}}(\mathcal{S}) = 0$ (by Claim 14), procedure $\mathsf{anex}(\mathcal{S}, C)$ will be applied again.

By Claim 14, $|\mathsf{indep}(\mathcal{S})|$ is strictly decreasing during the second phase. We again use $\mathcal{S}$ for the final outcome of this second phase. We have that $\mathsf{indep}(\mathcal{S}) = \emptyset$ and we conclude that $\mathcal{S}$ is a complete $\mathcal{A}$-normal $\Lambda$-state configuration of $G$ such that $|\mathsf{front}_{\mathcal{A}}(\mathcal{S})| = 0$.

We are now going to create a graph isomorphic to $\Lambda$ only by doing contractions in $G$. For this we use $\mathcal{S}$, a complete $\mathcal{A}$-normal $\Lambda$-state configuration of $G$ such that $|\mathsf{front}_{\mathcal{A}}(\mathcal{S})| = 0$, obtained as describe before. We contract in $G$ every country of $\mathcal{S}$ into a unique vertex. This can be done because the countries of $\mathcal{S}$ are connected. Let $G'$ be the resulting graph. By construction of $\mathcal{S}$, $G'$ is a contraction of $H$. Because of Condition 4 of $\Lambda$-state configuration, every freeway of $\mathcal{S}$ becomes an edge in $G'$. This implies that there is a graph isomorphic to $\Lambda$ that is a subgraph of $G'$. So $\hat{\Gamma}_{k'}$ is isomorphic to a subgraph of $G'$ with the same number of vertices. Let see $\hat{\Gamma}_{k'}$ as a subgraph of $G'$ and let $e$ be an edge of $G'$ that is not an edge of $\hat{\Gamma}_{k'}$. As $e$ is an edge of $G'$, this implies that in $G$, there is two states of $\mathcal{S}$ such that there is no freeway between them but still an edge. This is not possible by construction of $\mathcal{S}$. We deduce that $G'$ is isomorphic to $\hat{\Gamma}_{k'}$. Moreover, as $|\mathsf{front}_{\mathcal{A}}(\mathcal{S})| = 0$, then every cloud is a subset of a country. This implies that $G'$ is also a contraction of $H$. By contracting in $G'$ the edge corresponding to $\{a, (k'-1, k'-1)\}$ in $\hat{\Gamma}_{k'}$, we obtain that $\Gamma_{k'}$ is a contraction of $H$. Lemma 12 follows. ◀

**Proof of Theorem 7.** Given a graph $G$, we define $\mathbf{bcg}(G)$ as the minimum $k$ for which $G$ can be contracted to the uniformly triangulated grid $\Gamma_k$. Let $\lambda$, $c$, $c_1$, and $c_2$ be integers. It is enough to prove that there exists an integer $\lambda' = \mathcal{O}(\lambda \cdot c_1 \cdot (c_2)^c)$ such that for every graph class $\mathcal{G} \in \mathtt{SQGC}(c)$, the following holds:

$$\forall G \in \mathcal{G}\ \mathbf{tw}(G) \leq \lambda \cdot (\mathbf{bcg}(G))^c \quad \Rightarrow \quad \mathbf{tw}(G) \leq \lambda' \cdot (\mathbf{bcg}(G))^c.$$

Let $\mathcal{G} \in \mathtt{SQGC}(c)$ be a class of graph such that $\forall G \in \mathcal{G}\ \ \mathbf{tw}(G) \leq \lambda \cdot (\mathbf{bcg}(G))^c$. Let $H \in \mathcal{G}^{(c_1,c_2)}$ and let $G$ and $J$ be two graphs such that $G \in \mathcal{G}$, $G \leq^{(c_1)} J$, and $H \leq^{c_2} J$. $G$ and $J$ exist by definition of $\mathcal{G}^{(c_1,c_2)}$. By definition of $H$ and $J$, $\mathbf{tw}(H) \leq \mathbf{tw}(J)$. By Lemma 8, $\mathbf{tw}(J) \leq (c_1+1)(\mathbf{tw}(G)+1) - 1$. By definition of $\mathcal{G}$, $\mathbf{tw}(G) \leq \lambda \cdot \mathbf{bcg}(G)^c$. By Lemma 12, $\mathbf{bcg}(G) \leq (2c_2+1)(\mathbf{bcg}(H)+2)+1$. If we combine these four statements, we obtain that $\mathbf{tw}(H) \leq (c_1+1)(\lambda \cdot [(2c_2+1)(\mathbf{bcg}(H)+2)+1]^c + 1) - 1$. As this formula is independent of the graph class, the Theorem 7 follows. ◀

## 4 Conclusions and open problems

The main combinatorial result of this paper is that, for every $d$, the class $\mathcal{S}_d$ of string graphs with multi-degree at most $d$ has the SQGC property for $c = 1$. This means that, for fixed $d$, if a graph in $\mathcal{S}_d$ excludes as a contraction the uniformly triangulated grid $\Gamma_k$, then its treewidth is bounded by a linear function of $k$. Recall that string graphs are intersection graphs of lines in the plane. It is easy to extend our results for intersection graphs of lines in some orientable (or non-orientable) surface of genus $\gamma$. Let $\mathcal{S}_{d,\gamma}$ be the corresponding class. To prove that $\mathcal{S}_{d,\gamma} \in \texttt{SQGC}(1)$ we need first to extend Proposition 6 for $\mathcal{S}_{d,\gamma}$ (which is not hard) and then use Theorem 7 and the fact that the class of graphs of bounded genus belongs in $\texttt{SQGC}(1)$ (see e.g., [16]).

Of course, the main general question is to detect wide graph classes with the SQGM/SQGC property. In this direction, some insisting open issues are the following:
(1) Is the bound on the degree (or multi-degree) necessary? Are there classes of intersection graphs with unbounded or "almost bounded" maximum degree that have the SQGM/SQGC property?
(2) All so far known results classify graph classes in $\texttt{SQGM}(1)$ or $\texttt{SQGC}(1)$. Are there (interesting) graph classes in $\texttt{SQGM}(c)$ or $\texttt{SQGC}(c)$ for some $1 < c < 2$ that do not belong in $\texttt{SQGM}(1)$ or $\texttt{SQGC}(1)$ respectively? An easy (but trivial) example of such a class is the class $\mathcal{Q}_d$ of the q-dimensional grids, i.e., the cartesian products of $q \geq 2$ equal length paths. It is easy to see that the maximum $k$ for which an $n$-vertex graph $G \in \mathcal{Q}_q$ contains a $(k \times k)$-grid as a minor is $k = \Theta(n^{\frac{1}{2}})$. On the other size, it can also be proven that $\mathbf{tw}(G) = \Theta(n^{\frac{q-1}{q}})$. These two facts together imply that $\mathcal{Q}_q \in \texttt{SQGM}(2 - \frac{2}{q})$ while $\mathcal{Q}_q \notin \texttt{SQGM}(2 - \frac{2}{q} - \epsilon)$ for every $\epsilon > 0$.
(3) Usually the graph classes in $\texttt{SQGC}(1)$ are characterised by some "flatness" property. For instance, see the results in [29, 32] for $H$-minor free graphs, where $H$ is an apex graph. Can $\texttt{SQGC}(1)$ be useful as an intuitive definition of the "flatness" concept? Does this have some geometric interpretation?

## References


1 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

2 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015.

3 Julia Chuzhoy. Improved bounds for the excluded grid theorem. *CoRR*, abs/1602.02629, 2016.

4 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

5 Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. *Handbook of Graph Grammars*, pages 313–400, 1997.

6 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS 2011)*, pages 150–159, 2011.

7 Erik D. Demaine. Algorithmic Graph Minors and Bidimensionality. In *Proceedings of the 36th International Conference on Graph-theoretic Concepts in Computer Science (WG 2010)*, pages 2–2, 2010.

8 Erik D. Demaine, Fedor V. Fomin, MohammadTaghi Hajiaghayi, and Dimitrios M. Thilikos. Bidimensional parameters and local treewidth. *SIAM Journal on Discrete Mathematics*, 18(3):501–511, 2005.

9 Erik D. Demaine, Fedor V. Fomin, MohammadTaghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *Journal of the ACM*, 52(6):866–893, 2005.

10 Erik D. Demaine, Fedor V. Fomin, MohammadTaghi Hajiaghayi, and Dimitrios M. Thilikos. Bidimensional structures: Algorithms, combinatorics and logic. *Dagstuhl Reports*, 3(3):51–74, 2013.

11 Erik D. Demaine and MohammadTaghi Hajiaghayi. Fast algorithms for hard graph problems: Bidimensionality, minors, and local treewidth. In *Proceedings of the 12th International Symposium on Graph Drawing (GD 2004)*, volume 3383 of *LNCS*, pages 517–533, 2004.

12 Erik D. Demaine and MohammadTaghi Hajiaghayi. Bidimensionality: New connections between FPT algorithms and PTASs. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005)*, pages 590–601, 2005.

13 Erik D. Demaine and MohammadTaghi Hajiaghayi. The bidimensionality theory and its algorithmic applications. *The Computer Journal*, 51(3):292–302, 2008.

14 Erik D. Demaine and MohammadTaghi Hajiaghayi. Linearity of grid minors in treewidth with applications through bidimensionality. *Combinatorica*, 28(1):19–36, 2008.

15 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Dimitrios M. Thilikos. The bidimensional theory of bounded-genus graphs. *SIAM Journal on Discrete Mathematics*, 20(2):357–371, 2006.

16 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Dimitrios M. Thilikos. The bidimensional theory of bounded-genus graphs. *SIAM Journal on Discrete Mathematics*, 20(2):357–371, 2006.

17 Reinhard Diestel, Tommy R. Jensen, Konstantin Yu. Gorbunov, and Carsten Thomassen. Highly connected sets and the excluded grid theorem. *Journal of Combinatorial Theory. Series B*, 75(1):61–73, 1999.

18 Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. Fast subexponential algorithm for non-local problems on graphs of bounded genus. In *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006)*, volume 4059 of *LNCS*, pages 172–183, 2006.

19 Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. Catalan structures and dynamic programming in $H$-minor-free graphs. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008)*, pages 631–640, 2008.

20 Fedor V. Fomin, Erik D. Demaine, and MohammadTaghi Hajiaghayi. Bidimensionality. In *Encyclopedia of Algorithms*. Springer, 2015.

21 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. Contraction obstructions for treewidth. *Journal of Combinatorial Theory. Series B*, 101(5):302–314, 2011.

22 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Bidimensionality and EPTAS. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2011)*, pages 748–759, 2011.

23 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Bidimensionality and geometric graphs. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 1563–1575, 2012.

24 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 142–151, 2014.

**25** Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. In *Proceedings of the 21th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pages 503–510, 2010.

**26** Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. *CoRR*, abs/1606.05689, 2016.

**27** Fedor V. Fomin and Dimitrios M. Thilikos Petr Golovach and. Contraction bidimensionality: the accurate picture. In *17th Annual European Symposium on Algorithms*, volume 5757 of *LNCS*, pages 706–717. Springer, 2009.

**28** Fanica Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974.

**29** Archontia C. Giannopoulou and Dimitrios M. Thilikos. Optimizing the graph minors weak structure theorem. *SIAM Journal on Discrete Mathematics*, 27(3):1209–1227, 2013.

**30** Alexander Grigoriev, Athanassios Koutsonas, and Dimitrios M. Thilikos. Bidimensionality of Geometric Intersection Graphs. In *Proceedings of the 40th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2014)*, volume 8327 of *LNCS*, pages 293–305, 2014.

**31** Ken-ichi Kawarabayashi and Yusuke Kobayashi. Linear min-max relation between the treewidth of H-minor-free graphs and its largest grid. In *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, volume 14 of *LIPIcs*, pages 278–289, 2012.

**32** Ken-ichi Kawarabayashi, Robin Thomas, and Paul Wollan. New proof of the flat wall theorem. *Journal of Combinatorial Theory, Series B*, 2017. To appear.

**33** Alexander Leaf and Paul D. Seymour. Tree-width and planar minors. *Journal of Combinatorial Theory. Series B*, 111:38–53, 2015.

**34** Jiří Matoušek. String graphs and separators. In *Geometry, Structure and Randomness in Combinatorics*, pages 61–97. Springer, 2014.

**35** Michal Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In *Proceedings of the 36th International Conference on Mathematical Foundations of Computer Science (MFCS 2011)*, pages 520–531, 2011.

**36** Neil Robertson and Paul D. Seymour. Graph Minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

**37** Neil Robertson and Paul D. Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory. Series B*, 41(1):92–114, 1986.

**38** Neil Robertson, Paul D. Seymour, and Robin Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory. Series B*, 62(2):323–348, 1994.

**39** Juanjo Rué, Ignasi Sau, and Dimitrios M. Thilikos. Dynamic programming for H-minor-free graphs. In *Proceedings of Computing and Combinatorics - 18th Annual International Conference, (COCOON 2012)*, pages 86–97, 2012.

**40** Juanjo Rué, Ignasi Sau, and Dimitrios M. Thilikos. Dynamic programming for graphs on surfaces. *ACM Transactions on Algorithms*, 10(2):1–8, 2014.

**41** Dimitrios M. Thilikos. Graph minors and parameterized algorithm design. In *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, pages 228–256, 2012.

# Generalized Kakeya Sets for Polynomial Evaluation and Faster Computation of Fermionants*

**Andreas Björklund[1], Petteri Kaski[2], and Ryan Williams[3]**

1 **Department of Computer Science, Lund University, Lund, Sweden**

2 **Department of Computer Science, Aalto University, Helsinki, Finland**

3 **Department of Electrical Engineering and Computer Science & CSAIL, MIT, Cambridge, USA**

## Abstract

We present two new data structures for computing values of an $n$-variate polynomial $P$ of degree at most $d$ over a finite field of $q$ elements. Assuming that $d$ divides $q-1$, our first data structure relies on $(d+1)^{n+2}$ tabulated values of $P$ to produce the value of $P$ at any of the $q^n$ points using $O(nqd^2)$ arithmetic operations in the finite field. Assuming that $s$ divides $d$ and $d/s$ divides $q-1$, our second data structure assumes that $P$ satisfies a degree-separability condition and relies on $(d/s+1)^{n+s}$ tabulated values to produce the value of $P$ at any point using $O(nq^s sq)$ arithmetic operations. Our data structures are based on generalizing upper-bound constructions due to Mockenhaupt and Tao (2004), Saraf and Sudan (2008), and Dvir (2009) for Kakeya sets in finite vector spaces from linear to higher-degree polynomial curves.

As an application we show that the new data structures enable a faster algorithm for computing integer-valued *fermionants*, a family of self-reducible polynomial functions introduced by Chandrasekharan and Wiese (2011) that captures numerous fundamental algebraic and combinatorial invariants such as the determinant, the permanent, the number of Hamiltonian cycles in a directed multigraph, as well as certain partition functions of strongly correlated electron systems in statistical physics. In particular, a corollary of our main theorem for fermionants is that the permanent of an $m \times m$ integer matrix with entries bounded in absolute value by a constant can be computed in time $2^{m-\Omega\left(\sqrt{m/\log\log m}\right)}$, improving an earlier algorithm of Björklund (2016) that runs in time $2^{m-\Omega\left(\sqrt{m/\log m}\right)}$.

**1998 ACM Subject Classification** F.2.1 Numerical Algorithms and Problems, F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics, G.2.2 Graph Theory

**Keywords and phrases** Besicovitch set, fermionant, finite field, finite vector space, Hamiltonian cycle, homogeneous polynomial, Kakeya set, permanent, polynomial evaluation, tabulation

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.6

* This research was funded by the Swedish Research Council grant VR 2016-03855 "Algebraic Graph Algorithms" (A.B.), the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement 338077 "Theory and Practice of Advanced Search and Enumeration" (P.K.), and the U.S. National Science Foundation under grants CCF-1741638 and CCF-1741615 (R.W.).





# 1 Introduction

The protagonist of this paper is the following task. We want an efficient representation of an $n$-variate degree-$d$ polynomial $P$ over a finite field $\mathbb{F}_q$ of order $q$, that permits us to evaluate $P$ on arbitrary points $a \in \mathbb{F}_q^n$. What kind of resource trade-offs can be achieved between space (for representing $P$) and query time (for computing $P(a)$ at a given $a$)?

The study of data structures that enable fast "polynomial evaluation" queries for multivariate polynomials was initiated by Kedlaya and Umans [11] for polynomials with bounded individual variable degrees, motivated by applications to fast polynomial factorization. (For *univariate* polynomial evaluation, cf. von zur Gathen and Gerhard [24].) Here we focus on the case when $P$ has (total) degree $d$, in particular, when $d$ is less than $n$.[1]

We seek data structures consisting of a set $K \subseteq \mathbb{F}_q^n$ and an associated list $((a, P(a)) : a \in K)$ of evaluations. There are two extremes for such designs. At one extreme, we can set $K = \mathbb{F}_q^n$, put all evaluations in a sorted array, and binary search achieves $O(n \log q)$ query time. At the other extreme, to uniquely identify $P$ we must tabulate $\Omega(\binom{n+d}{d})$ points, as this is the dimension of the monomial basis. However, when $K$ is this small, we are only aware of brute-force ($n^{O(d)}$-time) algorithms to evaluate the polynomial in any other point. Between these two extremes, we seek constructions for sets $K$ that suffice for evaluating $P$ at any point outside $K$ in time that scales sub-exponentially in $d$. Our motivation is to accelerate the best known algorithms for canonical #P-hard problems (cf. Section 1.2).

## 1.1 Polynomial evaluation based on generalized Kakeya sets

Let $\mathbb{F}_q[x]$ be the ring of polynomials over indeterminates $x = (x_1, x_2, \ldots, x_n)$ with coefficients in $\mathbb{F}_q$. Our first main theorem constructs an explicit set $K \subseteq \mathbb{F}_q^n$ of cardinality at most $(d+1)^{n+2}$ which allows for relatively quick evaluation of any degree-$d$ $P$ at all points in $\mathbb{F}_q^n$.

▶ **Theorem 1.** *Let $d$ divide $q-1$. There is a set $K \subseteq \mathbb{F}_q^n$ of size $|K| \le (d+1)^{n+2}$ along with functions $g_1, g_2, \ldots, g_{(q-1)(d+1)^2} : \mathbb{F}_q^n \to K$ and scalars $\gamma_1, \gamma_2, \ldots, \gamma_{(q-1)(d+1)^2} \in \mathbb{F}_q$ such that for every polynomial $P \in \mathbb{F}_q[x]$ of degree at most $d$ and every vector $a \in \mathbb{F}_q^n$,*

$$P(a) = \sum_{j=1}^{(q-1)(d+1)^2} \gamma_j P(g_j(a)) \,.$$

▶ Remark. Let us write $\mathrm{M}(q)$ for the time complexity[2] of multiplication and division in $\mathbb{F}_q$. The construction in Theorem 1 is *explicit* in the sense that (a) there is an algorithm that in time $O(|K| nq \mathrm{M}(q))$ lists the elements of $K$; and (b) there is an algorithm that in time $O\big(nqd^2 \mathrm{M}(q)\big)$ computes the values $g_j(a) \in \mathbb{F}_q^n$ and $\gamma_j \in \mathbb{F}_q$ for all $j = 1, 2, \ldots, (q-1)(d+1)^2$ when given $a \in \mathbb{F}_q^n$ as input. The quadratic dependence on $d$ has not been optimized.

The size of $K$ can be further reduced for polynomials $P$ satisfying a certain (natural) restriction which holds for several well-studied polynomials. Suppose we partition the variable set $X = \{x_1, x_2, \ldots, x_n\}$ into $X = X_1 \cup X_2 \cup \cdots \cup X_d$ such that $|X_1| = |X_2| = \cdots = |X_d| = n/d$. Let us say that a degree-$d$ polynomial $P \in \mathbb{F}_q[x]$ is *degree-separable* relative to $X_1, X_2, \ldots, X_d$ if every monomial of $P$ contains one variable from each $X_i$. Note a degree-separable $P$ is

[1] In contrast, Kedlaya and Umans [11] focus on the case $n \le d^{o(1)}$; cf. [11, Corollaries 4.3, 4.5, and 6.4]. *Notational caveat:* Kedlaya and Umans use "$m$" for the number of variables.

[2] For example, $\mathrm{M}(q) = O\big((\log q)^{1+\epsilon}\big)$ holds for any constant $\epsilon > 0$; we refer to e.g. von zur Gathen and Gerhard [24] for sharper bounds.

in particular both multilinear and homogeneous of degree $d$. Degree-separability enables a trade-off between the size of $K$ and the query time for evaluation:

▶ **Theorem 2.** *Let $s$ divide $d$ and $d/s$ divide $q-1$. There is a set $K \subseteq \mathbb{F}_q^n$ of size $|K| \leq \binom{d/s+1}{}^{n+s}$ along with $g_1, g_2, \ldots, g_{(q-1)^s} : \mathbb{F}_q^n \to K$ and $\gamma_1, \gamma_2, \ldots, \gamma_{(q-1)^s} \in \mathbb{F}_q$ such that for every degree-separable degree-$d$ $P \in \mathbb{F}_q[x]$ relative to a fixed partition $X_1, X_2, \ldots, X_d$ and every vector $a \in \mathbb{F}_q^n$,*

$$P(a) = \sum_{j=1}^{(q-1)^s} \gamma_j P(g_j(a)) \,.$$

▶ Remark. The construction in Theorem 2 is explicit in the sense that (a) there is an algorithm that in time $O(|K|nq\mathrm{M}(q))$ lists the elements of $K$; and (b) there is an algorithm that in time $O(n(q-1)^s sq\mathrm{M}(q))$ computes the values $g_j(a) \in \mathbb{F}_q^n$ and $\gamma_j \in \mathbb{F}_q$ for all $j = 1, 2, \ldots, (q-1)^s$ when given $a \in \mathbb{F}_q^n$ as input.

We need $K$ to contain enough points that "interpolation" at all the other points is possible. One intuition for designing a small $K \subseteq \mathbb{F}_q^n$ for polynomial evaluation is that such a set must enable "localization" of any target polynomial inside the set. At one extreme, we may think of the simplest non-constant family of polynomials, namely *lines*. In Euclidean spaces, this line of thought leads to the study of dimensionality of sets that contain a unit line segment in every direction, or the *Kakeya problem*, which has been extensively studied since the 1920s and the seminal work of Besicovitch [2]. We refer to Wolff [26], Mockenhaupt and Tao [18], and Dvir [9, 10] for surveys both in the continuous and finite settings. In what follows we focus on finite vector spaces.

▶ **Definition 3.** A *Kakeya set* (or *Besicovitch set*) in a vector space of dimension $n$ over $\mathbb{F}_q$ is a subset $K \subseteq \mathbb{F}_q^n$ together with a function $f : \mathbb{F}_q^n \to \mathbb{F}_q^n$ such that for every vector $a \in \mathbb{F}_q^n$ and every scalar $\tau \in \mathbb{F}_q$ it holds that

$$f(a) + \tau a \in K \,. \tag{1}$$

That is, a Kakeya set $K$ has the property that for any possible direction of a line in $\mathbb{F}_q^n$ (that is, any nonzero vector $a \in \mathbb{F}_q^n$), the set $K$ contains an entire line (through $f(a)$) with this direction. To support our objective of polynomial evaluations for higher-order curves than lines, an intuition is now to generalize (1) to polynomials of higher degree in the indeterminate $\tau$. This is the methodological gist of our main contribution in this paper, which will be described further in Section 2.

As an illustrative application of our new data structures, we use Theorem 2 to derive a faster algorithm for computing fermionants, which are a family of self-reducible and degree-separable polynomials introduced by Chandrasekharan and Wiese [7] to generalize various fundamental polynomials. We start with a brief introduction to fermionants to motivate their study from a computational perspective.

## 1.2 Fermionants

We continue to work over $\mathbb{F}_q$. As usual, $S_m$ is the symmetric group over $[m] = \{1, 2, \ldots, m\}$. We write $c(\sigma)$ for the number of cycles in a permutation $\sigma \in S_m$, where each fixed point of $\sigma$ is

counted as a cycle of length 1. Let $A = (a_{ij} : i, j \in [m])$ be an $m \times m$ matrix of indeterminates. The *fermionant* of $A$ with (indeterminate) parameter $t$ is the $(m^2 + 1)$-variable polynomial

$$\operatorname{fer}_t A = (-1)^m \sum_{\sigma \in S_m} (-t)^{c(\sigma)} \prod_{i=1}^{m} a_{i,\sigma(i)} \,. \tag{2}$$

The fermionant is multilinear and homogeneous of degree $m$ with respect to the variables $\{a_{i,j}\}$, and of degree $m$ with respect to $t$. Furthermore, note that with respect to $\{a_{i,j}\}$ the fermionant is degree-separable under the partition $\{\{a_{ij} : j \in [m]\} : i \in [m]\}$.

The fermionant captures several extensively studied algebraic and combinatorial invariants, such as the determinant of a matrix

$$\det A = (-1)^m \sum_{\sigma \in S_m} (-1)^{c(\sigma)} \prod_{i=1}^{m} a_{i,\sigma(i)} \,,$$

the permanent of a matrix

$$\operatorname{per} A = \sum_{\sigma \in S_m} \prod_{i=1}^{m} a_{i,\sigma(i)} \,,$$

the generating function for directed Hamiltonian cycles

$$\operatorname{hc} A = \sum_{\substack{\sigma \in S_m \\ c(\sigma)=1}} \prod_{i=1}^{m} a_{i,\sigma(i)} \,,$$

as well as certain partition functions of strongly correlated electron systems in statistical physics (see Chandrasekharan and Wiese [7]). It is immediate that the aforementoined invariants can be obtained as special cases of the fermionant via

$$\det A = \operatorname{fer}_1 A \,, \qquad \operatorname{per} A = (-1)^m \operatorname{fer}_{-1} A \,, \qquad \text{and} \qquad \operatorname{hc} A = (-1)^{m-1} \{t\} \operatorname{fer}_t A \,,$$

where in the last equality we write $\{t^k\}P$ for the coefficient of $t^k$ in the polynomial $P$.

The invariants captured by the fermionant have received such substantial attention that is not possible to discuss the literature exhaustively here. For example, the permanent and the determinant are central to arithmetic circuit complexity [22] and geometric complexity theory [15]. Similarly, the numerous symmetries and self-reducibility properties of fermionants enable their use in e.g. interactive proof systems [5, 16, 25]. We restrict our present discussion of earlier work mostly to algorithms for the permanent.

Computing the permanent of a given $m \times m$ matrix appears to be an extremely hard problem. Indeed, the best known general algorithm is over 50 years old, given by Ryser [19] in 1963, and it uses $O(2^m m)$ arithmetic operations. Valiant [23] proved that the permanent for $\{0, 1\}$-matrix inputs is #P-hard, even if the number of ones per row is at most three. In the more general setting of fermionants, Mertens and Moore [17] showed that the fermionant is #P-hard for any $\tau > 2$ and $\oplus$P-hard for $\tau = 2$, even for the adjacency matrices of planar graphs. For the permanent, no less-than-$2^m$-sized arithmetic circuit is known despite substantial efforts (for example, it is a prominent open problem in the *Art of Computer Programming* [12]).

However, there are faster ways to compute the permanent if we allow random-access tabulation *along with* arithmetic operations. Most notably, there are modest speed-ups for $\{0, 1\}$-matrices over the integers. Bax and Franklin [1] gave an $2^{m - \Omega(m^{1/3}/\log m)}$ expected

time algorithm. Recently, Björklund [3] presented a deterministic $2^{m-\Omega(\sqrt{m/\log q})}$ time algorithm over any finite field of order $q \geq m^2+1$, by exploiting the self-reducibility of the permanent. Applying the Chinese Remainder Theorem, he also obtains a $2^{m-\Omega(\sqrt{m/\log m})}$-time algorithm for integer matrices with entries whose absolute value is bounded from above by a constant. There are also faster algorithms for sparse matrices. Cygan and Pilipczuk [8] gave a $2^{m-\Omega(m/r)}$ time algorithm for matrices with at most $r$ non-zero entries per row. Very recently, Björklund, Husfeldt, and Lyckberg [4] and Björklund, Kaski, and Koutis [6] show that if the result is bounded in absolute value by $c^m$ for a constant $c > 1$, then there are $2^{m\left(1-1/c^{\Omega(1)}\right)}m^{O(1)}$-time algorithms for the permanent and the number of directed Hamiltonian cycles, respectively. Both algorithms work by computing the permanent and the number of directed Hamiltonian cycles modulo small primes. In particular, the algorithms over $\mathbb{F}_p$ run in time $2^{m(1-1/p^{\Omega(1)})}$, faster than the algorithms of this paper for small $p$.

Our main technical result for fermionants is that, given mild technical conditions on the order of the field, we can compute obtain a faster algorithm over finite fields:

▶ **Theorem 4.** *There is an algorithm that computes the fermionant* $\operatorname{fer}_t A \in \mathbb{F}_q[t]$ *of a given* $m \times m$ *matrix* $A$ *with entries in* $\mathbb{F}_q$ *in time* $2^{m-\Omega\left(\sqrt{m/\log\log q}\right)}O(\mathrm{M}(q))$*, provided that* $q-1$ *has a divisor in the interval* $(1.1\log q, 10\log q)$*,* $q \geq m^2+1$*, and* $m = \omega\left(\log^2 q \log\log q\right)$*.*

The Chinese Remainder Theorem and a uniform variant of the Prime Number Theorem for arithmetic progressions yield the following corollary for integer-valued fermionants.

▶ **Corollary 5.** *Let* $\tau$ *be an integer with* $|\tau| \leq O(m)$ *and let* $M$ *be a constant. The fermionant* $\operatorname{fer}_\tau A$ *can be computed in time* $2^{m-\Omega\left(\sqrt{m/\log\log m}\right)}$*, for all* $m \times m$ *matrices* $A$ *with integer values in* $[-M, M]$*.*

The idea behind Theorem 4 is to apply our polynomial evaluation results to a self-reduction for fermionants. Following Björklund's results for the permanent [3], we show how to compute a fermionant on an $m \times m$ matrix via $2^{m-k}m^{O(1)}$ calls to the fermionant on $k \times k$ matrices. Applying Theorem 2, we set $k$ so that it is possible to evaluate the $k \times k$ fermionant polynomial over all points of $K$ in $2^{0.999m}$ time. Once we know the polynomial on all points in $K$, we can then evaluate the fermionant on any $m \times m$ matrix in time about $2^{m-\Omega(k)}m^{O(1)}$. We show $k \approx \sqrt{m/\log\log q}$ suffices.

**Organisation.** In Section 2, we present our generalization of Kakeya sets in finite vector spaces, together with explicit constructions. Next in Section 3 we prove our main evaluation theorems, Theorem 1 and Theorem 2. In Section 4 we use the self-reducibility of the fermionant to prove Theorem 4 and Corollary 5, showing how to compute fermionants faster.

## 2 Generalized Kakeya sets in finite vector spaces

Here we study the following generalization of Kakeya sets for lines (Definition 3) to higher-degree polynomial curves:

▶ **Definition 6.** A *Kakeya set* of *degree* $r$ in a vector space of dimension $n$ over $\mathbb{F}_q$ consists of a set $K \subseteq \mathbb{F}_q^n$ together with functions $f_0, f_1, \ldots, f_{r-1} : \mathbb{F}_q^n \to \mathbb{F}_q^n$ such that for every vector $a \in \mathbb{F}_q^n$ and every scalar $\tau \in \mathbb{F}_q$ it holds that

$$F(a,\tau) = f_0(a) + f_1(a)\tau + f_2(a)\tau^2 + \ldots + f_{r-1}(a)\tau^{r-1} + a\tau^r \in K\,. \tag{3}$$

We say that a construction for Kakeya sets is *explicit* if

**(i)** there is an algorithm that outputs $K$ (given $q$, $r$, and $n$) in $O\big(|K|nr\mathrm{M}(q)\big)$ time, and

**(ii)** there is an algorithm that given $a \in \mathbb{F}_q^n$ outputs the values $f_0(a), f_1(a), \ldots, f_{r-1}(a) \in \mathbb{F}_q^n$ in $O\big(nr\mathrm{M}(q)\big)$ time.

The following construction of sparse Kakeya sets of degree $r$ generalizes the design of the best known Kakeya sets (cf. Mockenhaupt and Tao [18], Saraf and Sudan [20], Dvir [9, §2.4], Kopparty, Lev, Saraf, and Sudan [13], and Kyureghyan, Müller, and Wang [14]).

▶ **Lemma 7.** *For every $r+1$ that divides $q-1$ there is an explicit Kakeya set $K \subseteq \mathbb{F}_q^n$ of degree $r$ and size $|K| \leq \left(\frac{q-1}{r+1}+1\right)^{n+1}$.*

**Proof.** We begin with three simple observations. First, since $r+1$ divides $q-1$, we have that $r+1$ has a multiplicative inverse in $\mathbb{F}_q$.[3] Second, for all $\alpha, \tau \in \mathbb{F}_q$ from the Binomial Theorem we have

$$\left(\frac{\alpha}{r+1}+\tau\right)^{r+1} - \tau^{r+1} = \sum_{i=0}^{r-1} \binom{r+1}{i} \left(\frac{\alpha}{r+1}\right)^{r+1-i} \tau^i \;+\; \alpha\tau^r\,. \tag{4}$$

Third, since the multiplicative subgroup $\mathbb{F}_q^\times$ is cyclic of order $q-1$, the subgroup consisting of $(r+1)$th powers of elements of $\mathbb{F}_q^\times$ has size exactly $\frac{q-1}{r+1}$. Including the zero element, we observe that $|\{\beta^{r+1} : \beta \in \mathbb{F}_q\}| = \frac{q-1}{r+1}+1$.

Let us now define $K \subseteq \mathbb{F}_q^n$ to consist of all vectors of the form

$$\left(\left(\frac{\alpha_1}{r+1}+\tau\right)^{r+1} - \tau^{r+1}, \left(\frac{\alpha_2}{r+1}+\tau\right)^{r+1} - \tau^{r+1}, \ldots, \left(\frac{\alpha_n}{r+1}+\tau\right)^{r+1} - \tau^{r+1}\right) \tag{5}$$

with $\alpha_1, \alpha_2, \ldots, \alpha_n, \tau \in \mathbb{F}_q$. It follows immediately from (5) and our third observation that $|K| \leq \left(\frac{q-1}{r+1}+1\right)^{n+1}$. Furthermore, (4) and (5) imply that the generalized Kakeya property (3) holds when we define the functions $f_i : \mathbb{F}_q^n \to \mathbb{F}_q^n$ for all $i = 0, 1, \ldots, r-1$ and $a = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{F}_q^n$ by

$$f_i(a) = \left(\binom{r+1}{i}\left(\frac{\alpha_1}{r+1}\right)^{r+1-i}, \binom{r+1}{i}\left(\frac{\alpha_2}{r+1}\right)^{r+1-i}, \ldots, \binom{r+1}{i}\left(\frac{\alpha_n}{r+1}\right)^{r+1-i}\right). \tag{6}$$

It is immediate from the definitions (5) and (6) that the construction is explicit. ◀

## 3 Polynomial evaluation

This section proves our two main theorems for polynomial evaluation. The key idea is Mellin-transform-like sieving (8) enabled by an elementary observation about sums over finite fields (7) below, which we then extend to an $s$-fold product form in (12).

Let us start with a homogeneous version of Theorem 1.

[3] Indeed, $q = p^a$ for a prime $p$ and positive integer $a$. Note $r+1$ has a multiplicative inverse if and only if $p$ does not divide $r+1$. By assumption we have $(r+1)Q = p^a - 1$ for an integer $Q$ and thus $r+1 = pb$ for an integer $b$ would lead to a contradiction $p(bQ - p^{a-1}) = -1$.

▶ **Lemma 8.** *Let $d$ divide $q-1$. There is a set $K \subseteq \mathbb{F}_q^n$ of size $|K| \leq (d+1)^{n+1}$ together with functions $g_1, g_2, \ldots, g_{q-1} : \mathbb{F}_q^n \to K$ and coefficients $\gamma_1, \gamma_2, \ldots, \gamma_{q-1} \in \mathbb{F}_q$ such that for every homogeneous polynomial $P \in \mathbb{F}_q[x]$ of degree $h \leq d$ and every vector $a \in \mathbb{F}_q^n$,*

$$P(a) = \sum_{j=1}^{q-1} \gamma_j P(g_j(a)) \,.$$

**Proof.** Let $d$ divide $q-1$. Set $r = (q-1)/d-1$, and note that $r+1$ divides $q-1$. Apply Lemma 7 to obtain $K$ and the functions $f_0, f_1, \ldots, f_{r-1}$. Let $P \in \mathbb{F}_q[x]$ be a homogeneous polynomial of degree $h \leq d$ over the indeterminates $x = (x_1, x_2, \ldots, x_n)$, and let $a = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{F}_q^n$ be an assignment of values to the indeterminates. Our goal is to compute the value $P(a) \in \mathbb{F}_q$ using evaluations of $P$ at $K$. Recalling the function $F(a, \tau)$ from (3), we will rely on values of the composition $P(F(a, \tau))$ for $\tau \in \mathbb{F}_q$ to obtain $P(a)$.

Towards this end, we first observe that

$$\sum_{\tau \in \mathbb{F}_q^\times} \tau^e = \begin{cases} -1 & \text{if } q-1 \text{ divides } e, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

To see this, let $g$ be a generator of the multiplicative subgroup $\mathbb{F}_q^\times$. If $q-1$ divides $e$ then $\tau^e = 1$ for all $\tau$, and thus the sum is $|\mathbb{F}_q^\times| = q-1$ (modulo the characteristic). Otherwise, $g^e \neq 1$, and we have $\sum_{\tau \in \mathbb{F}_q^\times} \tau^e = \sum_{\tau \in \mathbb{F}_q^\times} (g\tau)^e = g^e \sum_{\tau \in \mathbb{F}_q^\times} \tau^e$, so the sum must be 0.

Let $t = q - 1 - rh$ and observe that $t \geq 1$. We now claim that

$$P(a) = -\sum_{\tau \in \mathbb{F}_q^\times} \tau^t P(F(a, \tau)) \,. \tag{8}$$

By linearity, it suffices to consider the case when $P$ is a single monomial $P = x_1^{h_1} x_2^{h_2} \cdots x_n^{h_n}$ of degree $h = h_1 + h_2 + \ldots + h_n \leq d$. Recalling (3) and (7), we observe that the right-hand side of (8) expands to

$$\begin{aligned}
&-\sum_{\tau \in \mathbb{F}_q^\times} \tau^t P(F(a, \tau)) \\
&= -\sum_{\tau \in \mathbb{F}_q^\times} \tau^{q-1-rh} \left( \tau^{rh} \alpha_1^{h_1} \alpha_2^{h_2} \cdots \alpha_n^{h_n} + \tau^{rh-1}(\cdots) + \tau^{rh-2}(\cdots) + \ldots + \tau^0(\cdots) \right) \\
&= -\sum_{\tau \in \mathbb{F}_q^\times} \left( \tau^{q-1} \alpha_1^{h_1} \alpha_2^{h_2} \cdots \alpha_n^{h_n} + \tau^{q-2}(\cdots) + \tau^{q-3}(\cdots) + \ldots + \tau^{q-1-rh}(\cdots) \right) \\
&= \alpha_1^{h_1} \alpha_2^{h_2} \cdots \alpha_n^{h_n} \\
&= P(a) \,.
\end{aligned}$$

That is, by multiplying each term by $\tau^t$, we ensure that all other terms appearing inside of $P(F(a, \tau))$ cancel, except for the desired term $\alpha_1^{h_1} \alpha_2^{h_2} \cdots \alpha_n^{h_n}$ which is the coefficient of $\tau^{rh}$.

Now let $\beta_1, \beta_2, \ldots, \beta_{q-1}$ be an enumeration of the elements of $\mathbb{F}_q^\times$. For all $j = 1, 2, \ldots, q-1$, set $g_j(a) = F(a, \beta_j)$ and $\gamma_j = -\beta_j^t$. The lemma now follows from (8). ◀

## 3.1 Proof of Theorem 1

We are now ready to prove Theorem 1. Our strategy is to interpolate the homogeneous components of our given polynomial, then apply Lemma 8. Towards this end, let $P \in \mathbb{F}_q[x]$

have degree at most $d$ and let $P = \sum_{h=0}^{d} P_h$ where $P_h \in \mathbb{F}_q[x]$ is either zero or homogeneous of degree $h$, for all $h = 0, 1, \ldots, d$. Let $\nu_0, \nu_1, \ldots, \nu_d$ be any $d+1$ distinct elements of $\mathbb{F}_q$. Recalling the definition of $K$ in (5), let $\hat{K} \subseteq \mathbb{F}_q^n$ be the set of all vectors of the form

$$\nu\left(\left(\frac{\alpha_1}{r+1}+\tau\right)^{r+1} - \tau^{r+1}, \left(\frac{\alpha_2}{r+1}+\tau\right)^{r+1} - \tau^{r+1}, \ldots, \left(\frac{\alpha_n}{r+1}+\tau\right)^{r+1} - \tau^{r+1}\right) \tag{9}$$

where $\alpha_1, \alpha_2, \ldots, \alpha_n, \tau \in \mathbb{F}_q$, and $\nu \in \{\nu_0, \nu_1, \ldots, \nu_d\}$.

In particular, from (9) and (5) we have that $|\hat{K}| \leq (d+1)|K|$.

Assuming we have constant-time access to $P(a)$ for all $a \in \hat{K}$, we can access each $P_h$ at $k \in K$ by univariate interpolation over the $d+1$ distinct values of $\nu$, via the identity $P(\nu k) = \sum_{h=0}^{d} P_h(k)\nu^h$. That is, for $h, j = 0, 1, \ldots, d$, let $\lambda_{hj} \in \mathbb{F}_q$ be the Lagrange interpolation coefficients that satisfy $P_h(k) = \sum_{j=0}^{d} \lambda_{hj} P(\nu_j k)$ for all $k \in K$. Observe in particular that the coefficients $\lambda_{hj}$ depend only on $\nu_0, \nu_1, \ldots, \nu_d$. With access to values of $P_h$ at $K$, given a query $a \in \mathbb{F}_q^n$ we can use Lemma 8 to sieve for $P_h(a)$ for each $h = 0, 1, \ldots, d$. That is, we have $P(a) = \sum_{h=0}^{d} P_h(a) = -\sum_{h=0}^{d} \sum_{\tau \in \mathbb{F}_q^\times} \sum_{j=0}^{d} \tau^{q-1-rh} \lambda_{hj} P(\nu_j F(a, \tau))$. ◀

## 3.2 Proof of Theorem 2

Suppose $s$ divides $d$ and $d/s$ divides $q-1$. Let $X_1, X_2, \ldots, X_d$ be the partition of variables for degree-separability. For $i = 1, 2, \ldots, s$, take

$$Y_i = X_{(i-1)d/s+1} \cup X_{(i-1)d/s+2} \cup \cdots \cup X_{id/s}$$

and observe that $|Y_i| = n/s$ for all $i$. Furthermore, observe that every monomial of a polynomial $P \in \mathbb{F}_q[x]$ that is degree-separable relative to $X_1, X_2, \ldots, X_d$ has degree exactly $d/s$ when restricted to the variables of $Y_i$.

Let us extend the construction in Lemma 7 into an $s$-fold product form over the partition $Y_1, Y_2, \ldots, Y_s$. Accordingly, we work with a multivariate polynomial over $s$ indeterminates $\tau_1, \tau_2, \ldots, \tau_s$ instead of a univariate polynomial (3) over $\tau$. Let $a = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{F}_q^n$ and let us write $a_{Y_i} \in \mathbb{F}_q^{ns/d}$ for the restriction of $a$ to coordinates in $Y_i$. Set $r = (q-1)s/d - 1$. Let us write $F_{Y_i}(a_{Y_i}, \tau_i) \in \mathbb{F}_q^n$ for the vector obtained by applying the construction given by (3) and (6) to the vector $a_{Y_i}$ and $\tau_i$, thereby obtaining a vector of length $ns/d$ indexed by $Y_i$, followed by padding with 0-entries outside the indices $Y_i$ to obtain a vector of length $n$. Let us now define the (vector-valued) multivariate polynomial

$$F(a, \tau_1, \tau_2, \ldots, \tau_s) = F_{Y_1}(a_{Y_1}, \tau_1) + F_{Y_2}(a_{Y_2}, \tau_2) + \ldots + F_{Y_s}(a_{Y_s}, \tau_s)\,. \tag{10}$$

We observe by (3), (6), and (4) that $F(a, \tau_1, \tau_2, \ldots, \tau_s)$ ranges over all vectors of the form

$$\begin{aligned}
&\left(\left(\frac{\alpha_1}{r+1}+\tau_1\right)^{r+1} - \tau_1^{r+1}, \left(\frac{\alpha_2}{r+1}+\tau_1\right)^{r+1} - \tau_1^{r+1}, \ldots, \left(\frac{\alpha_{n/s}}{r+1}+\tau_1\right)^{r+1} - \tau_1^{r+1},\right.\\
&\left(\frac{\alpha_{n/s+1}}{r+1}+\tau_2\right)^{r+1} - \tau_2^{r+1}, \left(\frac{\alpha_{n/s+2}}{r+1}+\tau_2\right)^{r+1} - \tau_2^{r+1}, \ldots, \left(\frac{\alpha_{2n/s}}{r+1}+\tau_2\right)^{r+1} - \tau_2^{r+1},\\
&\cdots,\\
&\left.\left(\frac{\alpha_{n-n/s+1}}{r+1}+\tau_s\right)^{r+1} - \tau_s^{r+1}, \left(\frac{\alpha_{n-n/s+2}}{r+1}+\tau_s\right)^{r+1} - \tau_s^{r+1}, \ldots, \left(\frac{\alpha_n}{r+1}+\tau_s\right)^{r+1} - \tau_s^{r+1}\right)
\end{aligned} \tag{11}$$

with $\alpha_1, \alpha_2, \ldots, \alpha_n, \tau_1, \tau_2, \ldots, \tau_s \in \mathbb{F}_q$. We define $K$ to be the set of all such vectors. By similar reasoning as in the proof of Theorem 1, note that $|K| \leq \left(\frac{q-1}{r+1}+1\right)^{n+s} = \left(\frac{d}{s}+1\right)^{n+s}$.

Let $t = q - 1 - rd/s$ and observe that $t \geq 1$. From (7) and proceeding analogously as with the reasoning for (8) in the proof of Theorem 1, we thus have

$$P(a) = (-1)^s \sum_{\tau_1,\tau_2,\ldots,\tau_s \in \mathbb{F}_q^\times} \tau_1^t \tau_2^t \cdots \tau_s^t P(F(a, \tau_1, \tau_2, \ldots, \tau_s)) \,. \tag{12}$$

Let $\beta_1, \beta_2, \ldots, \beta_{q-1}$ be an enumeration of the elements of $\mathbb{F}_q^\times$. For all $j = (j_1, j_2, \ldots, j_s) \in \{1, 2, \ldots, q-1\}^s$ take

$$g_j(a) = F(a, \beta_{j_1}, \beta_{j_2}, \ldots, \beta_{j_s}) \qquad \text{and} \qquad \gamma_j = (-1)^s \beta_{j_1}^t \beta_{j_2}^t \cdots \beta_{j_s}^t \,.$$

The theorem now follows from (12). ◀

# 4 Fermionants

This section proves our two main theorems for evaluating fermionants. We start by noting that the fermionant is self-reducible, a result that easily follows from earlier work by Björklund [3], followed by the proofs of our present main theorems.

## 4.1 Self-reducibility of the fermionant

This subsection reviews how Björklund's [3] self-reducibility for permanents can be extended to fermionants. In essence, his methodology can be used to reduce the task of computing one fermionant of size $m \times m$ to the task of computing $2^{m-k} m^{O(1)}$ fermionants of size $k \times k$. We stress that this subsection is provided for ease of exposition only and no claim of originality is made.

Let $r$, $t$, and $a_{ij}$ for $i, j \in [m]$ be polynomial indeterminates and let $\mathbb{F}$ be the coefficient field. For $S \subseteq [m]$, $i, j \in S$, and $\ell = 0, 1, \ldots, m$, consider the inductively-defined family of polynomials:

$$W^S_{\ell,i,j}(r) = \begin{cases} 1 & \text{if } \ell = 0 \text{ and } i = j; \\ 0 & \text{if } \ell = 0 \text{ and } i \neq j; \\ \sum_{u \in S} a_{iu} r W^S_{\ell-1,u,j}(r) & \text{if } \ell \geq 1. \end{cases} \tag{13}$$

The polynomial $W^S_{\ell,i,j}$ can be viewed as a multivariate generating function for (edge-multisets of) walks of length $\ell$ inside $S \subseteq [m]$ that start at $i$ and end at $j$, on the complete graph of $m$ vertices. The degree of each monomial in the indeterminate $r$ is equal to $\ell$. The indeterminates $a_{uv}$ track the edges $(u, v)$ traversed by the walk, with degree indicating the multiplicity, that is, how many times each edge was traversed.

For $S \subseteq [m]$ and $i \in [m]$, let us write $S_{\geq i} = \{u \in S : i \leq u\}$. For $S \subseteq [m]$, $i \in S$, and $\ell = 0, 1, \ldots, m$, introduce the following bivariate polynomial

$$C_i^S(r, t) = 1 - t \sum_{\ell=1}^{m} W^{S_{\geq i}}_{\ell,i,i}(r) \,. \tag{14}$$

This polynomial forms a multivariate generating function for (edge-multisets of) closed walks inside $S$ and "anchored" at $i$, including the possibility of no walk at all. Here, by "anchored" at $i$ we mean that the lowest-numbered vertex of the closed walk is $i$. Let

$$C^S(r, t) = \prod_{i \in S} C_i^S(r, t) \,. \tag{15}$$

Let $k = 0, 1, \ldots, m$. For $i, j \in [k]$ and $S \subseteq [m] \setminus [k]$, introduce the univariate polynomial

$$\tilde{a}^S_{i,j}(r) = a_{ij} + \sum_{\ell=0}^{m-1} \sum_{u,v \in S} a_{iu} W^S_{\ell,u,v} a_{vj} r \,. \tag{16}$$

This polynomial is a multivariate generating function for representing the (edge-multisets of) walk segments that traverse $[m]$ so that the first vertex of the segment is $i$ and the last vertex of the segment is $j$; the walk can either proceed directly from $i$ to $j$, or perform a walk of length $\ell$ in $S$ before ending at $j$. Let us arrange the coefficients $\tilde{a}^S_{i,j}(r)$ into a $k \times k$ matrix $\tilde{A}^S(r)$.

For a polynomial $P$ in the indeterminate $r$, let us write $\{r^j\}P$ for the coefficient (polynomial) of the monomial $r^j$. By the principle of inclusion and exclusion, we have:

▶ **Theorem 9.** *For all $k = 0, 1, \ldots, m$, we have the polynomial identity*

$$\operatorname{fer}_t A = \{r^{m-k}\} \sum_{S \subseteq [m] \setminus [k]} (-1)^{|S|} C^S(r,t) \operatorname{fer}_t \tilde{A}^S(r) \,. \tag{17}$$

Observing that the right-hand side of (17) has degree at most $m^2$ in $r$, and using Lagrange interpolation together with dynamic programming on the recurrences (13), (14), (15), and (16), we have:

▶ **Theorem 10.** *Suppose $|\mathbb{F}| \geq m^2 + 1$ and let $k = 0, 1, \ldots, m$. Then, there is a value $L = 2^{m-k} m^{O(1)}$ computable in time polynomial in $m$, and an algorithm that given as input a matrix $A \in \mathbb{F}^{m \times m}$, $\tau \in \mathbb{F}$, and an integer $j = 1, 2, \ldots, L$, runs in time $m^{O(1)}$, executes $m^{O(1)}$ arithmetic operations in $\mathbb{F}$, and outputs a matrix $\tilde{A}_j \in \mathbb{F}^{k \times k}$ together with a coefficient $\alpha_j \in \mathbb{F}$ such that:*

$$\operatorname{fer}_\tau A = \sum_{j=1}^{L} \alpha_j \operatorname{fer}_\tau \tilde{A}_j \,. \tag{18}$$

*In particular, the fermionant $\operatorname{fer}_\tau A$ of a given $A \in \mathbb{F}^{m \times m}$ at $\tau \in \mathbb{F}$ can be computed in $2^m m^{O(1)}$ time and arithmetic operations in $\mathbb{F}$.*

## 4.2 Proof of Theorem 4

Let $A \in \mathbb{F}_q^{m \times m}$ be given together with $\tau \in \mathbb{F}_q$. We seek to compute $\operatorname{fer}_\tau A$ and will deploy the self-reducibility enabled by Theorem 10 towards this end. By assumption we have that $q - 1$ has a divisor $u$ with $1.1 \log q \leq u \leq 10 \log q$. Since $m = \omega(\log^2 q \log\log q)$, for all large enough $m$ we can let $k$ be a multiple of $u$ with

$$0.98\sqrt{m/\log\log q} \leq k \leq 0.99\sqrt{m/\log\log q} \,.$$

With the objective of applying Theorem 2, take $n = k^2$, $d = k$, and $s = k/u$. Observe that the fermionant (2) of a $k \times k$ matrix $A$ at $\tau \in \mathbb{F}_q$ is a degree-separable polynomial $P$ of degree $d$ over the $n$ variables in $A$. Furthermore, $s$ divides $d$ and $d/s$ divides $q - 1$, so the assumptions of Theorem 2 hold. By Theorem 10 we can evaluate this $P$ at any given point (that is, for any given $k \times k$ matrix) in time $2^k k^{O(1)}$ and operations in $\mathbb{F}_q$. The tabulation of $P$ for Theorem 2 thus can be done in time

$$\begin{aligned} 2^k k^{O(1)} \left(\frac{d}{s} + 1\right)^{n+s} \mathrm{M}(q) &\leq 2^k k^{O(1)} \, (u+1)^{0.99m/\log\log q + \sqrt{m}} \, \mathrm{M}(q) \\ &\leq 2^k k^{O(1)} \, (20 \log q)^{0.999m/\log\log q} \, \mathrm{M}(q) \\ &\leq 2^{0.9999m} \mathrm{M}(q) \,. \end{aligned}$$

Once the tabulation of $P$ is complete, we can use the algorithms in Theorem 2 to query the $2^{m-k}m^{O(1)}$ fermionants of size $k \times k$ required by (18) in time $O(n(q-1)^s s\mathrm{M}(q))$ per query. Thus, the total time is at most

$$\begin{aligned} 2^{m-k}q^s m^{O(1)}\mathrm{M}(q) &\leq 2^{m-0.98\sqrt{m/\log\log q}}2^{(\log q)0.99\sqrt{m/\log\log q}/(1.1\log q)}m^{O(1)}\mathrm{M}(q) \\ &\leq 2^{m-0.07\sqrt{m/\log\log q}}m^{O(1)}\mathrm{M}(q)\,. \end{aligned}$$

◀

## 4.3 Proof of Corollary 5

Here we show how to extend the algorithm to integers, via the Chinese Remainder Theorem. Let $A$ be an integer matrix of size $m \times m$ with entries in $[-M, M]$ for $M = O(1)$. Let $\tau$ be an integer with $|\tau| = O(m)$. By Bertrand's postulate (e.g. [21, §I.1]) for all large enough $m$ we can select a prime $u$ with $5\log m \leq u \leq 10\log m$. Let us study the number of primes $p$ in the interval $Mm^2 < p < Mm^4$ such that $u$ divides $p-1$. Let us write $\varphi$ for Euler's totient function and recall the uniform variant of the Prime Number Theorem for arithmetic progressions [21, Corollary 8.31]. Namely, there is a constant $\gamma > 0$ such that, for any function $h(x)$ tending to infinity with $x$, and uniformly for $x \geq 3$ and $1 \leq u \leq (\ln x)^2/\left(h(x)^2(\ln\ln x)^6\right)$, we have

$$\sum_{\substack{p\leq x \\ p\equiv 1 \pmod u}} 1 \;=\; \frac{x}{\varphi(u)\ln x}\left(1+O\left(\frac{1}{(\ln x)^{\gamma h(x)}}\right)\right). \tag{19}$$

Here the left-hand side sum in (19) is over all primes $p$ at most $x$ congruent to 1 modulo $u$.

Since $u$ is prime, we have $\varphi(u) = u - 1 = \Theta(\log m)$. Thus from (19) we conclude that for all large enough $m$ there exist at least $2m$ distinct primes $p$ such that both $Mm^2 < p < Mm^4$ and $u$ divides $p-1$. With the objective of satisfying the assumptions of Theorem 4, we conclude $u$ is in the interval $(1.1\log p, 10\log p)$ for these $2m$ primes $p$. Indeed, since $M$ is a constant, for all large enough $m$ we have $1.99\log m \leq \log p \leq 4.01\log m$, which implies $(5/4.01)\log p \leq 5\log m \leq u \leq 10\log m \leq (10/1.99)\log p$.

From (2) we observe that $|\operatorname{fer}_\tau A| \leq m!\cdot O(m)^m M^m < \frac{1}{2}m^{4m}M^{2m}$. Applying the Chinese Remainder Theorem together with Theorem 4 on $A$ and $\tau$ over $\mathbb{F}_p$ for each of the $2m$ primes $p$ in turn, we recover $\operatorname{fer}_\tau A$ over the integers, in time $2^{m-\Omega\left(\sqrt{m/\log\log m}\right)}$. ◀

## References


1. Eric T. Bax and Joel Franklin. A finite-difference sieve to count paths and cycles by length. *Inf. Process. Lett.*, 60(4):171–176, 1996. doi:10.1016/S0020-0190(96)00159-7.
2. A. S. Besicovitch. On Kakeya's problem and a similar one. *Math. Z.*, 27(1):312–320, 1928. doi:10.1007/BF01171101.
3. Andreas Björklund. Below all subsets for some permutational counting problems. In Rasmus Pagh, editor, *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016, June 22-24, 2016, Reykjavik, Iceland*, volume 53 of *LIPIcs*, pages 17:1–17:11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.SWAT.2016.17.
4. Andreas Björklund, Thore Husfeldt, and Isak Lyckberg. Computing the permanent modulo a prime power. *Inf. Process. Lett.*, 125:20–25, 2017. doi:10.1016/j.ipl.2017.04.015.
5. Andreas Björklund and Petteri Kaski. How proofs are prepared at camelot: Extended abstract. In George Giakkoupis, editor, *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 391–400. ACM, 2016. doi:10.1145/2933057.2933101.

6 Andreas Björklund, Petteri Kaski, and Ioannis Koutis. Directed hamiltonicity and out-branchings via generalized laplacians. 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 91:1–91:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.91.

7 Shailesh Chandrasekharan and Uwe-Jens Wiese. Partition functions of strongly correlated electron systems as "fermionants". *cond-mat.str-el*, abs/1108.2461, 2011. arXiv:1108.2461v1.

8 Marek Cygan and Marcin Pilipczuk. Faster exponential-time algorithms in graphs of bounded average degree. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 364–375. Springer, 2013. doi:10.1007/978-3-642-39206-1_31.

9 Zeev Dvir. From randomness extraction to rotating needles. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:77, 2009. URL: http://eccc.hpi-web.de/report/2009/077.

10 Zeev Dvir. Incidence theorems and their applications. *CoRR*, abs/1208.5073, 2012. arXiv:1208.5073.

11 Kiran S. Kedlaya and Christopher Umans. Fast polynomial factorization and modular composition. *SIAM J. Comput.*, 40(6):1767–1802, 2011. doi:10.1137/08073408X.

12 Donald E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, 3rd edition, 1998.

13 Swastik Kopparty, Vsevolod F. Lev, Shubhangi Saraf, and Madhu Sudan. Kakeya-type sets in finite vector spaces. *J. Algebraic Combin.*, 34(3):337–355, 2011. doi:10.1007/s10801-011-0274-8.

14 Gohar Kyureghyan, Peter Müller, and Qi Wang. On the size of Kakeya sets in finite vector spaces. *Electron. J. Combin.*, 20(3):#P36, 2013. URL: http://www.combinatorics.org/ojs/index.php/eljc/article/view/v20i3p36.

15 J. M. Landsberg. An introduction to geometric complexity theory. *Eur. Math. Soc. Newsl.*, 99:10–18, 2016.

16 Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992. doi:10.1145/146585.146605.

17 Stephan Mertens and Cristopher Moore. The complexity of the fermionant, and immanants of constant width. *CoRR*, abs/1110.1821, 2011. arXiv:1110.1821.

18 Gerd Mockenhaupt and Terence Tao. Restriction and Kakeya phenomena for finite fields. *Duke Math. J.*, 121(1):35–74, 2004. doi:10.1215/S0012-7094-04-12112-8.

19 H. J. Ryser. *Combinatorial Mathematics*. Number 14 in The Carus Mathematical Monographs. Mathematical Association of America, 1963.

20 Shubhangi Saraf and Madhu Sudan. An improved lower bound on the size of Kakeya sets over finite fields. *Anal. PDE*, 1(3):375–379, 2008. doi:10.2140/apde.2008.1.375.

21 Gérald Tenenbaum. *Introduction to Analytic and Probabilistic Number Theory*, volume 163 of *Graduate Studies in Mathematics*. American Mathematical Society, 3rd edition, 2015.

22 Leslie G. Valiant. Completeness classes in algebra. In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 249–261. ACM, 1979. doi:10.1145/800135.804419.

23 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

**24** Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, Cambridge, third edition, 2013. doi:10.1017/CBO9781139856065.

**25** Richard Ryan Williams. Strong ETH breaks with merlin and arthur: Short non-interactive proofs of batch evaluation. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, volume 50 of *LIPIcs*, pages 2:1–2:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CCC.2016.2.

**26** Thomas Wolff. Recent work connected with the Kakeya problem. In *Prospects in Mathematics (Princeton, NJ, 1996)*, pages 129–162. Amer. Math. Soc., Providence, RI, 1999.

# Generalized Feedback Vertex Set Problems on Bounded-Treewidth Graphs: Chordality Is the Key to Single-Exponential Parameterized Algorithms*†

**Édouard Bonnet[1], Nick Brettell[2], O-joung Kwon[3], and Dániel Marx[4]**

1 **Department of Computer Science, Middlesex University, London, UK**
2 **School of Mathematics and Statistics, Victoria University of Wellington, Wellington, New Zealand**
3 **Logic and Semantics, Technische Universität Berlin, Berlin, Germany**
4 **Institute for Computer Science and Control, Hungarian Academy of Sciences, (MTA SZTAKI), Budapest, Hungary**

## Abstract

It has long been known that Feedback Vertex Set can be solved in time $2^{\mathcal{O}(w \log w)}n^{\mathcal{O}(1)}$ on graphs of treewidth $w$, but it was only recently that this running time was improved to $2^{\mathcal{O}(w)}n^{\mathcal{O}(1)}$, that is, to single-exponential parameterized by treewidth. We investigate which generalizations of Feedback Vertex Set can be solved in a similar running time. Formally, for a class of graphs $\mathcal{P}$, Bounded $\mathcal{P}$-Block Vertex Deletion asks, given a graph $G$ on $n$ vertices and positive integers $k$ and $d$, whether $G$ contains a set $S$ of at most $k$ vertices such that each block of $G - S$ has at most $d$ vertices and is in $\mathcal{P}$. Assuming that $\mathcal{P}$ is recognizable in polynomial time and satisfies a certain natural hereditary condition, we give a sharp characterization of when single-exponential parameterized algorithms are possible for fixed values of $d$:

- if $\mathcal{P}$ consists only of chordal graphs, then the problem can be solved in time $2^{\mathcal{O}(wd^2)}n^{\mathcal{O}(1)}$,
- if $\mathcal{P}$ contains a graph with an induced cycle of length $\ell \geqslant 4$, then the problem is not solvable in time $2^{o(w \log w)}n^{\mathcal{O}(1)}$ even for fixed $d = \ell$, unless the ETH fails.

We also study a similar problem, called Bounded $\mathcal{P}$-Component Vertex Deletion, where the target graphs have connected components of small size instead of having blocks of small size, and present analogous results.

**1998 ACM Subject Classification** G.2.1 Combinatorial Algorithms, G.2.2 Graph Algorithms

**Keywords and phrases** fixed-parameter tractable algorithms, treewidth, feedback vertex set

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.7

## 1 Introduction

Treewidth is a measure of how well a graph accommodates a decomposition into a tree-like structure. In the field of parameterized complexity, many NP-hard problems have been shown to have FPT algorithms when parameterized by treewidth; for example, Coloring, Vertex Cover, Feedback Vertex Set, and Steiner Tree. In fact, Courcelle [6] established a

* All authors were supported by ERC Starting Grant PARAMTIGHT (No. 280152) and ERC Consolidator Grant SYSTEMATICGRAPH (No. 725978). The third author was also supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant DISTRUCT, agreement No. 648527).

† The full version can be found in [5], `https://arxiv.org/abs/1704.06757`.





meta-theorem that every problem definable in $\mathrm{MSO}_2$ logic can be solved in linear time on graphs of bounded treewidth. While Courcelle's Theorem is a very general tool for obtaining algorithmic results, for specific problems dynamic programming techniques usually give algorithms where the running time $f(w)n^{\mathcal{O}(1)}$ has better dependence on treewidth $w$. There is some evidence that careful implementation of dynamic programming (plus maybe some additional ideas) gives optimal dependence for some problems (see, e.g., [12]).

For Feedback Vertex Set, standard dynamic programming techniques give $2^{\mathcal{O}(w \log w)} n^{\mathcal{O}(1)}$-time algorithms and it was considered plausible that this could be the best possible running time. Hence it was a remarkable surprise when it turned out that $2^{\mathcal{O}(w)}n^{\mathcal{O}(1)}$ algorithms are also possible for this problem by various techniques: Cygan et al. [7] obtained a $3^w n^{\mathcal{O}(1)}$-time randomized algorithm by using the so-called Cut & Count technique, and Bodlaender et al. [2] showed there is a deterministic $2^{\mathcal{O}(w)}n^{\mathcal{O}(1)}$-time algorithm by using a rank-based approach and the concept of representative sets. This was also later shown in the more general setting of representative sets in matroids by Fomin et al. [11].

**Generalized feedback vertex set problems.** We explore the extent to which these results apply for generalizations of Feedback Vertex Set. The Feedback Vertex Set problem asks for a set $S$ of at most $k$ vertices such that $G - S$ is acyclic, or in other words, every block of $G - S$ is a single edge or vertex. We consider generalizations where we allow the blocks to be some other type of small graph, such as triangles, small cycles, or small cliques; these generalizations were first studied in [4]. The main result of this paper is that the existence of single-exponential algorithms is closely linked to whether the small graphs we are allowing are all chordal or not. Formally, we consider the following problem:

| Bounded $\mathcal{P}$-Block Vertex Deletion | **Parameter:** $d$, $w$ |
|---|---|
| **Input:** A graph $G$ of treewidth at most $w$, and positive integers $d$ and $k$. | |
| **Question:** Is there a set $S$ of at most $k$ vertices in $G$ such that each block of $G - S$ has at most $d$ vertices and is in $\mathcal{P}$? | |

The result of Bodlaender et al. [2] implies that when $d = 2$, Bounded $\mathcal{P}$-Block Vertex Deletion can be solved in time $2^{\mathcal{O}(w)}n^{\mathcal{O}(1)}$. Our main question is for which graph classes $\mathcal{P}$ can this problem be solved in time $2^{\mathcal{O}(w)}n^{\mathcal{O}(1)}$, when we regard $d$ as a fixed constant. A graph is *chordal* if it has no induced cycles of length at least 4. We show that if $\mathcal{P}$ consists of only chordal graphs, then we can solve this problem in single-exponential time for fixed $d$.

▶ **Theorem 1.** *Let $\mathcal{P}$ be a class of graphs that is block-hereditary, recognizable in polynomial time, and consists of only chordal graphs. Then* Bounded $\mathcal{P}$-Block Vertex Deletion *can be solved in time $2^{\mathcal{O}(wd^2)}k^2n$ on graphs with $n$ vertices and treewidth $w$.*

The condition that $\mathcal{P}$ is block-hereditary ensures that the class of graphs with blocks in $\mathcal{P}$ is hereditary; a formal definition is given in Section 2. We complement this result by showing that if $\mathcal{P}$ contains a graph that is not chordal, then single-exponential algorithms are not possible (assuming ETH), even for fixed $d$. Note that if $\mathcal{P}$ is block-hereditary and contains a graph that is not chordal, then this graph contains a chordless cycle on $\ell \geqslant 4$ vertices and consequently the cycle graph on $\ell$ vertices is also in $\mathcal{P}$.

▶ **Theorem 2.** *If $\mathcal{P}$ contains the cycle graph on $\ell \geqslant 4$ vertices, then* Bounded $\mathcal{P}$-Block Vertex Deletion *is not solvable in time $2^{o(w \log w)}n^{\mathcal{O}(1)}$ on graphs of treewidth at most $w$ even for fixed $d = \ell$, unless the ETH fails.*

Baste et al. [1] recently studied the complexity of a similar problem, where the task is to find a set of vertices whose deletion results in a graph with no minor in a given collection

of graphs $\mathcal{F}$, parameterized by treewidth. When $\mathcal{F} = \{C_4\}$, this is equivalent to Bounded $\mathcal{P}$-Block Vertex Deletion where $\mathcal{P} = \{K_2, K_3\}$, and the complexity they obtain in this case is consistent with our result.

Whether this lower bound of Theorem 2 is best possible when $\mathcal{P}$ contains a cycle on $\ell \geqslant 4$ vertices remains open. However, as partial evidence towards this, we note that when $\mathcal{P}$ contains all graphs, the result by Baste et al. [1] implies that that Bounded $\mathcal{P}$-Block Vertex Deletion can be solved in time $2^{\mathcal{O}(w \log w)} n^{\mathcal{O}(1)}$ when $d$ is fixed, as the minor obstruction set $\mathcal{F}$ consists of all of 2-connected graphs with $d+1$ vertices.

**Bounded-size components.** Using a similar technique, we can obtain analogous results for a slightly simpler problem, that we call Bounded $\mathcal{P}$-Component Vertex Deletion, where we want to remove at most $k$ vertices such that each connected component of the resulting graph has at most $d$ vertices and belongs to $\mathcal{P}$. If we have only the size constraint (i.e., $\mathcal{P}$ contains every graph), then this problem is known as Component Order Connectivity [9]. Drange et al. [9] studied the parameterized complexity of a weighted variant of the Component Order Connectivity problem; their results imply, in particular, that Component Order Connectivity can be solved in time $2^{\mathcal{O}(k \log d)} n$, but is $W[1]$-hard parameterized by only $k$ or $d$. The corresponding edge-deletion problem, parameterized by treewidth, was studied by Enright and Meeks [10].

▶ **Theorem 3.** *Let $\mathcal{P}$ be a class of graphs that is hereditary, recognizable in polynomial time, and consists of only chordal graphs. Then* Bounded $\mathcal{P}$-Component Vertex Deletion *can be solved in time $2^{\mathcal{O}(wd^2)} k^2 n$ on graphs with $n$ vertices and treewidth $w$.*

▶ **Theorem 4.** *If $\mathcal{P}$ contains the cycle graph on $\ell \geqslant 4$ vertices, then* Bounded $\mathcal{P}$-Component Vertex Deletion *is not solvable in time $2^{o(w \log w)} n^{\mathcal{O}(1)}$ on graphs of treewidth at most $w$ even for fixed $d = \ell$, unless the ETH fails.*

The result of Baste et al. [1] implies that when $\mathcal{P}$ contains all graphs, Bounded $\mathcal{P}$-Component Vertex Deletion can be solved in time $2^{\mathcal{O}(w \log w)} n^{\mathcal{O}(1)}$. When $d$ is not fixed, one might ask whether Bounded $\mathcal{P}$-Component Vertex Deletion admits an $f(w)n^{\mathcal{O}(1)}$-time algorithm; that is, an FPT algorithm parameterized only by treewidth. We provide a negative answer: the problem is $W[1]$-hard when $\mathcal{P}$ contains all chordal graphs, even parameterized by both treewidth and $k$. Furthermore, two stronger lower bound results hold, under the assumption of the ETH.

▶ **Theorem 5.** *Let $\mathcal{P}$ be a hereditary class containing all chordal graphs. Then* Bounded $\mathcal{P}$-Component Vertex Deletion *is $W[1]$-hard parameterized by the combined parameter $(w, k)$. Moreover, unless the ETH fails, (1) this problem has no $f(w)n^{o(w)}$-time algorithm; and (2) it has no $f(k')n^{o(k'/\log k')}$-time algorithm, where $k' = w + k$.*

**Techniques.** A pair $(G, S)$ consisting of a graph $G$ and a vertex subset $S$ of $G$ will be called a boundaried graph, and an $S$-block of $G$ is a block of $G$ containing an edge with both endpoints in $S$. The algorithm for Bounded $\mathcal{P}$-Block Vertex Deletion uses several lemmas on $S$-blocks of boundaried graphs $(G, S)$, which appear in Section 3. The key property is the following: (*) when we merge two boundaried graphs $(G, S)$ and $(H, S)$ into a graph $G'$, to decide whether each $S$-block of $G'$ is some fixed target graph that is chordal, it is sufficient to know, for each non-trivial block $B$ of $G[S]$ or $H[S]$, some local information about $B$ in the $S$-block containing $B$ in $G$ or $H$, respectively. We think of target graphs as labeled graphs where any two vertices in the same block have distinct labels in

$\{1, \ldots, d\}$, and the local information referred to in (*) is the set of labels of neighbors of $B$ in the $S$-block containing $B$. The related result is stated as Proposition 6. This will be used to determine whether each of the $S$-blocks of $G'$ is one of the target graphs in $\mathcal{P}$. After then, to decide whether $G'$ is a required graph, it remains to check that the whole graph has no chordless cycle, since there is a possibility of linking two controlled blocks by a sequence of uncontrolled blocks in both sides $G$ and $H$, and thus creating a chordless cycle in $G'$. This second part can be dealt with in a similar manner to the single-exponential time algorithm for Feedback Vertex Set, using representative-set techniques.

## 2 Preliminaries

We follow the terminology of Diestel [8], unless otherwise specified. A vertex $v$ of $G$ is a *cut vertex* if the deletion of $v$ from $G$ increases the number of connected components. We say $G$ is *biconnected* if it is connected and has no cut vertices. Note that every connected graph on at most two vertices is biconnected. A *block* of $G$ is a maximal biconnected subgraph of $G$. We say $G$ is 2-*connected* if it is biconnected and $|V(G)| \geqslant 3$. An induced cycle of length at least four is called a *chordless cycle*. A graph is *chordal* if it has no chordless cycles. For a class of graphs $\mathcal{P}$, a graph is called a $\mathcal{P}$-*block graph* if each of its blocks is in $\mathcal{P}$. A class $\mathcal{C}$ of graphs is *block-hereditary* if for every $G \in \mathcal{C}$ and every biconnected induced subgraph $H$ of $G$, $H \in \mathcal{C}$. For two integers $d_1, d_2$ with $d_1 \leqslant d_2$, let $[d_1, d_2]$ be the set of all integers $i$ with $d_1 \leqslant i \leqslant d_2$, and for a positive integer, let $[d] := [1, d]$. For a function $f : X \to Y$ and $X' \subseteq X$, the function $f' : X' \to Y$ where $f'(x) = f(x)$ for all $x \in X'$ is called the *restriction* of $f$ on $X'$, and is denoted $f|_{X'}$. We also say that $f$ *extends* $f'$ to the set $X$.

**Block $d$-labeling.** A *block $d$-labeling* of a graph $G$ is a function $L : V(G) \to [d]$ such that for each block $B$ of $G$, $L|_{V(B)}$ is an injection. If $G$ is equipped with a block $d$-labeling $L$, then it is called a *(block) $d$-labeled graph*, and we call $L(v)$ the *label* of $v$. Two $d$-labeled graphs $G$ and $H$ are *label-isomorphic* if there is a graph isomorphism from $G$ to $H$ that is label preserving. For biconnected block $d$-labeled graphs $G$ and $H$, $H$ is *partially label-isomorphic* to $G$ if $H$ is label-isomorphic to the subgraph of $G$ induced by the vertices with labels in $H$.

**Treewidth.** A *tree decomposition* of a graph $G$ is a pair $(T, \mathcal{B})$ consisting of a tree $T$ and a family $\mathcal{B} = \{B_t\}_{t \in V(T)}$ of sets $B_t \subseteq V(G)$, called *bags*, satisfying the following three conditions: (1) $V(G) = \bigcup_{t \in V(T)} B_t$, (2) for every edge $uv$ of $G$, there exists a node $t$ of $T$ such that $u, v \in B_t$, (3) for $t_1, t_2, t_3 \in V(T)$, $B_{t_1} \cap B_{t_3} \subseteq B_{t_2}$ whenever $t_2$ is on the path from $t_1$ to $t_3$ in $T$. The *width* of a tree decomposition $(T, \mathcal{B})$ is $\max\{|B_t| - 1 : t \in V(T)\}$. The *treewidth* of $G$ is the minimum width over all tree decompositions of $G$. A tree decomposition $(T, \mathcal{B} = \{B_t\}_{t \in V(T)})$ is *nice* if $T$ is a rooted tree with root node $r$, and every node $t$ of $T$ is one of the following: (1) a *leaf node*: $t$ is a leaf of $T$ and $B_t = \emptyset$; (2) an *introduce node*: $t$ has exactly one child $t'$ and $B_t = B_{t'} \cup \{v\}$ for some $v \in V(G) \setminus B_{t'}$; (3) a *forget node*: $t$ has exactly one child $t'$ and $B_t = B_{t'} \setminus \{v\}$ for some $v \in B_{t'}$; or (4) a *join node*: $t$ has exactly two children $t_1$ and $t_2$, and $B_t = B_{t_1} = B_{t_2}$.

**Boundaried graphs.** For a graph $G$ and $S \subseteq V(G)$, the pair $(G, S)$ is a *boundaried graph*. When $G$ is a $d$-labeled graph, we simply say that $(G, S)$ is a $d$-labeled graph. Two $d$-labeled graphs $(G, S)$ and $(H, S)$ are said to be *compatible* if $V(G - S) \cap V(H - S) = \emptyset$, $G[S] = H[S]$, and $G$ and $H$ have the same labels on $S$. For two compatible $d$-labeled graphs $(G, S)$ and $(H, S)$, the *sum* of two graphs $(G, S) \oplus (H, S)$ is the graph obtained from the disjoint union of

$G$ and $H$ by identifying each vertex in $S$ and removing an edge if multiple edges appear. We denote by $L_G \oplus L_H$ the function from $V((G,S) \oplus (H,S))$ to $[d]$ where for $v \in V(G) \cup V(H)$, $(L_G \oplus L_H)(v) = L_G(v)$ if $v \in V(G)$ and $(L_G \oplus L_H)(v) = L_H(v)$ otherwise. For two unlabeled boundaried graphs, we define the sum in the same way, but ignoring the label condition.

A block of a graph is *non-trivial* if it has at least two vertices. For a boundaried graph $(G,S)$, a block $B$ of $G$ is called an *$S$-block* if it contains an edge of $G[S]$. Note that every non-trivial block of $G[S]$ is contained in a unique $S$-block of $G$ because two distinct blocks share at most one vertex. Let $(G,S)$ be a boundaried graph. We define $\mathbf{Aux}(G,S)$ as the bipartite boundaried graph with bipartition $(\mathcal{C}_1, \mathcal{C}_2)$ and boundary $\mathcal{C}_2$ such that (1) $\mathcal{C}_1$ is the set of components of $G$, and $\mathcal{C}_2$ is the set of components of $G[S]$, (2) for $C_1 \in \mathcal{C}_1$ and $C_2 \in \mathcal{C}_2$, $C_1C_2 \in E(\mathbf{Aux}(G,S))$ if and only if $C_2$ is contained in $C_1$. When $(G,S)$ and $(H,S)$ are two compatible $d$-labeled graphs, $\mathbf{Aux}(G,S) \oplus \mathbf{Aux}(H,S)$ is well-defined, as $G$ and $H$ have the same set of components on $S$. For a set $S$ and a set $\mathcal{X}$ of subsets of $S$, let $\mathbf{Inc}(S,\mathcal{X})$ be the bipartite graph on the bipartition $(S,\mathcal{X})$ where for $v \in S$ and $X \in \mathcal{X}$, $v$ and $X$ are adjacent in $\mathbf{Inc}(S,\mathcal{X})$ if and only if $v \in X$. For a boundaried graph $(G,S)$, when $\mathcal{P}$ is the partition of the set $\mathcal{C}$ of components of $G[S]$ such that two components of $G[S]$ are in the same part if and only if they are in the same component of $G$, we denote by $\mathbf{Inc}(\mathcal{C},\mathcal{P}) \sim \mathbf{Aux}(G,S)$.

## 3 Lemmas about $S$-blocks

We present several lemmas regarding $S$-blocks. For a biconnected $d$-labeled graph $Q$, a $d$-labeled graph $(G,S)$ is *block-wise partially label-isomorphic to $Q$* if every $S$-block $B$ of $G$ is partially label-isomorphic to $Q$. For two compatible $d$-labeled graphs $(G,S)$ and $(H,S)$ with labelings $L_G$ and $L_H$ respectively, we say $(G,S)$ and $(H,S)$ are *block-wise $Q$-compatible* if

1. $(G,S)$ and $(H,S)$ are block-wise partially label-isomorphic to $Q$; and
2. for every non-trivial block $B$ of $G[S]$, letting $B_1$ and $B_2$ be the $S$-blocks of $G$ and $H$ that contain $B$, respectively, $L_G(N_{B_1}(V(B)) \setminus S) \cap L_H(N_{B_2}(V(B)) \setminus S) = \emptyset$, and, for $\ell_1 \in L_G(N_{B_1}(V(B)) \setminus S)$ and $\ell_2 \in L_H(N_{B_2}(V(B)) \setminus S)$, the vertices in $Q$ with labels $\ell_1$ and $\ell_2$ are not adjacent.

We describe sufficient conditions for when, given a chordal labeled graph $Q$, the sum of two given labeled graphs $(G,S)$ and $(H,S)$, each partially label-isomorphic to $Q$, is also partially label-isomorphic to $Q$.

▶ **Proposition 6.** *Let $Q$ be a biconnected $d$-labeled chordal graph. Let $(G,S)$ and $(H,S)$ be two block-wise $Q$-compatible $d$-labeled graphs such that $\mathbf{Aux}(G,S) \oplus \mathbf{Aux}(H,S)$ has no cycles. Then $(G,S) \oplus (H,S)$ is block-wise partially label-isomorphic to $Q$.*

We use the following essential property of chordal graphs.

▶ **Lemma 7.** *Let $F$ be a connected graph and let $Q$ be a connected chordal graph. Let $\mu : V(F) \to V(Q)$ be a function such that for every induced path $p_1 \cdots p_m$ in $F$ of length at most two, $\mu(p_1), \ldots, \mu(p_m)$ are pairwise distinct and $\mu(p_1) \cdots \mu(p_m)$ is an induced path of $Q$. Then $\mu$ is an injection and preserves the adjacency relation.*

▶ **Lemma 8.** *Let $(G,S)$ and $(H,S)$ be two compatible $d$-labeled graphs such that $\mathbf{Aux}(G,S) \oplus \mathbf{Aux}(H,S)$ has no cycles. (1) If $F$ is an $S$-block of $(G,S) \oplus (H,S)$ and $uv$ is an edge in $F$, then $uv$ is contained in some $S$-block of $G$ or $H$. (2) Suppose each $S$-block of $G$ or $H$ is chordal. If $F$ is an $S$-block of $(G,S) \oplus (H,S)$ and $uvw$ is an induced path in $F$ such that $u$ and $w$ are not contained in the same $S$-block of $G$ or $H$, then $v \in S$, and there is an induced path $q_1q_2 \cdots q_\ell$ from $u = q_1$ to $w = q_\ell$ in $F - v$ such that each $q_i$ is a neighbor of $v$.*

**Proof of Proposition 6.** Let $F$ be an $S$-block of $(G,S)\oplus(H,S)$. Let $L_G$ and $L_H$ be labelings of $G$ and $H$, respectively, and let $L := L_G \oplus L_H$. We may assume $|V(F)| \geqslant 3$. By Lemma 8, every edge of $F$ is contained in some $S$-block of $G$ or $H$. Thus, for $uv \in E(F)$, we have $L(u) \neq L(v)$ and the vertices with labels $L(u)$ and $L(v)$ are adjacent in $Q$. Moreover, since $(G,S)$ and $(H,S)$ are block-wise partially label-isomorphic to $Q$, we have $L(V(F)) \subseteq L_Q(V(Q))$. Let $\mu : V(F) \to V(Q)$ such that for each $v \in V(F)$, $L(v) = L_Q(\mu(v))$.

To apply Lemma 7, it is sufficient to prove that if $uvw$ is an induced path in $F$, then $L(u) \neq L(w)$ and $\mu(u)\mu(v)\mu(w)$ is an induced path in $Q$. Since $(G,S)$ and $(H,S)$ are block-wise partially label-isomorphic to $Q$, if all of $u,v,w$ are contained in an $S$-block of $G$ or $H$, then it follows from the given condition. We may assume $u$ and $w$ are not contained in the same $S$-block of $G$ or $H$. Then by (2) of Lemma 8, $v \in S$, and there is an induced path $q_1q_2\cdots q_\ell$ from $u = q_1$ to $w = q_\ell$ in $F - v$ such that each $q_i$ is a neighbor of $v$.

We show that for $i \in \{1,\ldots,\ell-2\}$, $L(q_i), L(q_{i+1}), L(q_{i+2})$ are pairwise distinct, and $\mu(q_i)\mu(q_{i+1})\mu(q_{i+2})$ is an induced path of $Q$. If all of $q_i, q_{i+1}, q_{i+2}$ are contained in $G$ or $H$, then they are contained in the same $S$-block as $v$, and the claim follows. We may assume $q_i$ and $q_{i+2}$ are in distinct graphs of $G - S$ and $H - S$. Then the $S$-block containing $q_i, q_{i+1}, v$ and the $S$-block containing $q_{i+1}, q_{i+2}, v$ share the edge $q_{i+1}v$. Since $(G,S)$ and $(H,S)$ are block-wise $Q$-compatible, $L(q_i) \neq L(q_{i+2})$ and $\mu(q_i)$ is not adjacent to $\mu(q_{i+2})$ in $Q$.

We verify that $\mu(q_1)\mu(q_2)\cdots\mu(q_\ell)$ is an induced path of $Q$. Suppose this is false, and choose $i_1, i_2 \in \{1,2,\ldots,\ell\}$ with $i_2 - i_1 > 1$ and minimum $i_2 - i_1$ such that $\mu(q_{i_1})$ is adjacent to $\mu(q_{i_2})$ in $Q$. By minimality, $\mu(q_{i_1})\cdots\mu(q_{i_2-1})$ and $\mu(q_{i_1+1})\cdots\mu(q_{i_2})$ are induced paths and have length at least 2. Thus $\mu(q_{i_1})\cdots\mu(q_{i_2})$ is an induced cycle of length at least 4, contradicting the assumption that $Q$ is chordal. Therefore, $\mu(q_1)\mu(q_2)\cdots\mu(q_\ell)$ is an induced path of $Q$, and, in particular, $L(u) \neq L(w)$ and $\mu(u)$ and $\mu(w)$ are not adjacent in $Q$, as required. By Lemma 7, we conclude that $F$ is partially label-isomorphic to $Q$. ◀

Using Lemma 8, we can also prove the following.

▶ **Lemma 9.** *Let $A$ be a set, let $(G,S)$ and $(H,S)$ be two compatible $d$-labeled graphs, and let $\mathcal{B}$ be the set of non-trivial blocks in $G[S]$. Suppose $g : \mathcal{B} \to A$ is a function where each $S$-block of $G$ or $H$ is chordal, $\mathbf{Aux}(G,S) \oplus \mathbf{Aux}(H,S)$ has no cycles, and for every $B_1, B_2 \in \mathcal{B}$ where $B_1$ and $B_2$ are contained in an $S$-block of $G$ or $H$, $g(B_1) = g(B_2)$. If $F$ is an $S$-block of $(G,S)\oplus(H,S)$ and $B_1, B_2 \in \mathcal{B}$ where $V(B_1), V(B_2) \subseteq V(F)$, then $g(B_1) = g(B_2)$.*

▶ **Proposition 10.** *Let $(G,S)$ and $(H,S)$ be two compatible $d$-labeled graphs such that every $S$-block of $(G,S)\oplus(H,S)$ is chordal. Then $(G,S)\oplus(H,S)$ is chordal if and only if $\mathbf{Aux}(G,S)\oplus\mathbf{Aux}(H,S)$ has no cycles.*

**Proof.** We briefly sketch the proof of one direction. Suppose that $\mathbf{Aux}(G,S)\oplus\mathbf{Aux}(H,S)$ has a cycle $C_1 - A_1 - C_2 - A_2 - \cdots - C_n - A_n - C_1$ where $C_1,\ldots,C_n$ are components of $G[S]$. For each $i \in \{1,\ldots,n\}$, let $P_i$ be the shortest path from $C_i$ to $C_{i+1}$ in $A_i$, and let $v_i, w_i$ be the end vertices of $P_i$ where $v_i \in V(C_i)$ and $w_i \in V(C_{i+1})$. Let $Q_i$ be the shortest path from $w_i$ to $v_{i+1}$ in $C_{i+1}$. We may assume $n \geqslant 3$; it is easy when $n = 2$. Then $v_1P_1 - Q_1 - P_2 - Q_2 - \cdots - P_n - Q_nv_1$ is a cycle in $(G,S)\oplus(H,S)$, but is not necessarily a chordless cycle. We claim that it contains a chordless cycle. Let $x$ be the vertex following $v_2$ in $P_2$, and let $y$ be the vertex preceding $w_n$ in $P_n$. Take a shortest path $P$ from $x$ to $y$ in the path $y - Q_n - P_1 - Q_1 - x$. Clearly $P$ has length at least 2, as $x$ and $y$ are contained in distinct connected components of $G$ or $H$. Also, every internal vertex of $P$ has no neighbors in the other path of the cycle $v_1P_1 - Q_1 - P_2 - Q_2 - \cdots - P_n - Q_nv_1$ between $x$ and $y$. So, if we take a shortest path $P'$ from $x$ to $y$ along the other part of the cycle $v_1P_1 - Q_1 - P_2 - Q_2 - \cdots - P_n - Q_nv_1$, then $P \cup P'$ is a chordless cycle. ◀

## 4 Bounded $\mathcal{P}$-Block Vertex Deletion

We prove Theorem 1. We first focus on $S$-blocks of boundaried graphs $(G, S)$. For each non-trivial block of $G[S]$, we guess its final shape, as a $d$-labeled biconnected graph, and store the labelings of the vertices and their neighbors in the $S$-block of $G$ containing it. Collectively, we call this information a *characteristic* of $(G, S)$. Using characteristics, we control $S$-blocks in $(G, S) \oplus (H, S)$, where $(H, S)$ is a compatible $d$-labeled graph. By the previous step, we may assume that every $S$-block of $(G, S) \oplus (H, S)$ is in $\mathcal{P}$ and has at most $d$ vertices. Note that $(G, S) \oplus (H, S)$ still may have a chordless cycle. By Proposition 10, if we assume that every $S$-block of $(G, S) \oplus (H, S)$ is in $\mathcal{P}$, then $(G, S) \oplus (H, S)$ is chordal if and only if $\mathbf{Aux}(G, S) \oplus \mathbf{Aux}(H, S)$ has no cycles. So, instead of keeping $\mathbf{Aux}(G, S)$, we store the corresponding partition of the set of components of $G[S]$.

For convenience, we fix an integer $d \geqslant 2$ and a class $\mathcal{P}$ of graphs that is block-hereditary, recognizable in polynomial time, and consists of only chordal graphs. Let $\mathcal{U}_d$ be the set of all $d$-labeled biconnected $\mathcal{P}$-block graphs, where each $H$ in $\mathcal{U}_d$ has labeling $L_H$. For a boundaried graph $(G, S)$, we denote by $\mathrm{Block}(G, S)$ the set of all non-trivial blocks in $G[S]$.

For a $d$-labeled graph $(G, S)$ with a labeling $L$, a *characteristic* of $(G, S)$ is a pair $(g, h)$ of functions $g : \mathrm{Block}(G, S) \to \mathcal{U}_d$ and $h : \mathrm{Block}(G, S) \to 2^{[d]}$ satisfying the following, for each $B \in \mathrm{Block}(G, S)$ and the unique $S$-block $H$ of $G$ containing $B$,

1. (label-isomorphic condition) $H$ is partially label-isomorphic to $g(B)$;
2. (coincidence condition) for every $B' \in \mathrm{Block}(G, S)$ with $V(B') \subseteq V(H)$, $g(B') = g(B)$;
3. (neighborhood condition) $h(B) = L(N_H(V(B)) \setminus S)$; and
4. (complete condition) for every $w$ where $w \in V(H) \setminus S$ or $\{w\} = V(H) \cap V(C)$ for some component $C$ of $G[S]$, $H[N_H[w]]$ is label-isomorphic to $g(B)[N_{g(B)}[z]]$ where $z$ is the vertex in $g(B)$ with label $L(w)$.

We say that the sum $(G, S) \oplus (H, S)$ *respects* $(g, h)$ if for each $B \in \mathrm{Block}(G, S)$, the $S$-block of $(G, S) \oplus (H, S)$ containing $B$ is label-isomorphic to $g(B)$. The following is the main combinatorial result regarding characteristics.

▶ **Theorem 11.** *Let $(G_1, S)$, $(G_2, S)$, $(H, S)$ be $d$-labeled $\mathcal{P}$-block graphs such that each $(G_i, S)$ is compatible with $(H, S)$, $(G_1, S)$ and $(G_2, S)$ have the same characteristic $(g, h)$, and $\mathbf{Aux}(G_2, S) \oplus \mathbf{Aux}(H, S)$ has no cycles. If $(G_1, S) \oplus (H, S)$ is a $d$-labeled $\mathcal{P}$-block graph that respects $(g, h)$, then $(G_2, S) \oplus (H, S)$ is a $d$-labeled $\mathcal{P}$-block graph that respects $(g, h)$.*

**Proof.** We show $(G_2, S) \oplus (H, S)$ respects $(g, h)$. Choose a non-trivial block $B$ of $G_2[S]$, let $Q := g(B)$, let $F$ be the $S$-block of $(G_2, S) \oplus (H, S)$ containing $B$, $L_F$ be the function from $V(F)$ to $[d]$ that sends each vertex to its label from $G_2$ or $H$, and $L_Q$ be the labeling of $Q$.

We claim that $F$ is label-isomorphic to $Q$. We regard $F$ as the sum of $(F \cap G_2, V(F) \cap S)$ and $(F \cap H, V(F) \cap S)$ and verify the conditions of Proposition 6. Using Lemma 9, for every $B' \in \mathrm{Block}(G_2, S)$ with $V(B') \subseteq V(F)$, $g(B') = Q$. We also observe that $\mathbf{Aux}(F \cap G_2, S_F) \oplus \mathbf{Aux}(F \cap H, S_F)$ has no cycles as $\mathbf{Aux}(G_2, S) \oplus \mathbf{Aux}(H, S)$ has no cycles. Since $(g, h)$ is a characteristic of $(G_2, S)$ and $(G_1, S) \oplus (H, S)$ respects $(g, h)$, we can confirm that both $F \cap G$ and $F \cap H$ are block-wise partially label-isomorphic to $Q$. The second condition of being block-wise $Q$-compatible follows from the fact that $(G_1, S)$ and $(G_2, S)$ have the same characteristic $(g, h)$. Thus, $F \cap G_2$ and $F \cap H$ are block-wise $Q$-compatible, and this implies that $F$ is partially label-isomorphic to $Q$ by Proposition 6. By the 'complete condition' of a characteristic, we can show that $L_Q(V(Q)) \subseteq L_F(V(F))$, so $F$ is label-isomorphic to $Q$.

Lastly, we can confirm that $(G_2, S) \oplus (H, S)$ is a $d$-labeled $\mathcal{P}$-block graph by showing that every non $S$-block of $(G_2, S) \oplus (H, S)$ is fully contained in $G_2$ or $H$. We can argue this using the fact that $(G_2, S) \oplus (H, S)$ is chordal, which is implied by Proposition 10. ◀

**Proof of Theorem 1.** We obtain a nice tree decomposition $(T, \mathcal{B} = \{B_t\}_{t \in V(T)})$ of $G$ with root node $r$ and width at most $5w + 4$ in time $\mathcal{O}(c^w \cdot n)$ for some constant $c$ using the approximation algorithm by Bodlaender et al. [3]. For $t \in V(T)$, let $G_t$ be the subgraph of $G$ induced by the union of all bags $B_{t'}$ where $t'$ is a descendant of $t$. Let $\mathrm{Comp}(t, X)$ be the set of all components of $G[B_t \setminus X]$, and $\mathrm{Part}(t, X)$ be the set of all partitions of $\mathrm{Comp}(t, X)$.

For each node $t$ of $T$, $X \subseteq B_t$, and a function $L : B_t \setminus X \to [d]$, we define $\mathcal{F}(t, X, L)$ as the set of all pairs $(g, h)$ consisting of functions $g : \mathrm{Block}(t, X) \to \mathcal{U}_d$ and $h : \mathrm{Block}(t, X) \to 2^{[d]}$. We say that $(g, h)$ is *valid*, if (1) $L$ is a $d$-labeling of $G[B_t \setminus X]$, (2) for each $B \in \mathrm{Block}(t, X)$, $B$ is partially label-isomorphic to $g(B)$, and (3) for each $B \in \mathrm{Block}(t, X)$, $L(V(B)) \cap h(B) = \emptyset$. For $i \in \{0, 1, \ldots, k\}$ and $(g, h) \in \mathcal{F}(t, X, L)$, let $c[t, (X, L, i, (g, h))]$ be the family of all partitions $\mathcal{X} \in \mathrm{Part}(t, X)$ satisfying the following property: there exist $S \subseteq V(G_t) \setminus B_t$ with $|S| = i$ and a $d$-labeling $L'$ of $G_t - (X \cup S)$ where (1) $L = L'|_{B_t \setminus X}$, (2) $G_t - (X \cup S)$ is a $\mathcal{P}$-block graph, (3) $(g, h)$ is a characteristic of $(G_t - (X \cup S), B_t \setminus X)$, and (4) $\mathbf{Inc}(\mathrm{Comp}(t, X), \mathcal{X}) \sim \mathbf{Aux}(G_t - (X \cup S), B_t \setminus X)$. Such a pair $(S, L')$ is a *partial solution* with respect to $\mathcal{X}$.

The main idea is that instead of fully computing $c[t, M]$ for $M = (X, L, i, (g, h))$, we recursively enumerate a set $r[t, M]$ that may represent partial solutions for $c[t, M]$. Formally, for a subset $r[t, M] \subseteq c[t, M]$, we denote $r[t, M] \equiv c[t, M]$ if for every $\mathcal{X} \in c[t, M]$ and a partial solution $(S, L')$ with respect to $\mathcal{X}$ and $S_{out} \subseteq V(G) \setminus V(G_t)$ where $G - (S \cup X \cup S_{out})$ is a $d$-labeled $\mathcal{P}$-block graph respecting $(g, h)$, there exists $\mathcal{X}_1 \in r[t, M]$ and a partial solution $(S', L'')$ with respect to $\mathcal{X}_1$ such that $G - (S' \cup X \cup S_{out})$ is a $d$-labeled $\mathcal{P}$-block graph respecting $(g, h)$. By the definition of $r[t, M]$, the problem is a YES-instance if and only if there exists $(X, L, i, (g, h))$ for the root node $r$ with $|X| + i \leqslant k$ such that $r[r, (X, L, i, (g, h)] \neq \emptyset$.

Whenever we update $r[t, M]$, we confirm that $|r[t, M]| \leqslant w \cdot 2^{w-1}$. This will be the application of the representative set technique developed by Bodlaender et al. [2]. For a set $S$ and a set $\mathcal{A}$ of partitions of $S$, a subset $\mathcal{A}'$ of $\mathcal{A}$ is called a *representative set* if for every $\mathcal{X}_1 \in \mathcal{A}$ and every partition $\mathcal{Y}$ of $S$ where $\mathbf{Inc}(S, \mathcal{X}_1 \cup \mathcal{Y})$ has no cycles, there exists a partition $\mathcal{X}_2 \in \mathcal{A}'$ such that $\mathbf{Inc}(S, \mathcal{X}_2 \cup \mathcal{Y})$ has no cycles.

▶ **Proposition 12.** *Given a family $\mathcal{A}$ of partitions of a set $S$, one can output a representative set of $\mathcal{A}$ of size at most $|S| \cdot 2^{|S|-1}$ in time $\mathcal{A}^{\mathcal{O}(1)} 2^{\mathcal{O}(|S|)}$.*

We sketch how to update families $r[t, M]$ when $t$ is an introduce node with child node $t'$. We may assume $(g, h)$ is valid, otherwise $c[t, M] = \emptyset$.

Let $v$ be the vertex in $B_t \setminus B_{t'}$. If $v \in X$, then $G_t - X = G_{t'} - (X \setminus \{v\})$ and $B_t \setminus X = B_{t'} \setminus (X \setminus \{v\})$. Thus, we can set $r[t, M] := r[t', (X \setminus \{v\}, L, i, (g, h))]$. We assume $v \notin X$, and let $L_{res} := L|_{B_{t'} \setminus X}$. For $(g, h) \in \mathcal{F}(t, X, L)$, a pair $(g', h') \in \mathcal{F}(t', X, L_{res})$ is called the *restriction* of $(g, h)$ if (1) for $B_1 \in \mathrm{Block}(t', X)$ and $B_2 \in \mathrm{Block}(t, X)$ with $V(B_1) \subseteq V(B_2)$, $g'(B_1) = g(B_2)$, and if $v \in V(B_2)$, then every vertex in $g'(B_1)$ with label in $h'(B_1)$ is not adjacent to the vertex in $g'(B_1)$ with label $L(v)$, (2) for $B_1 \in \mathrm{Block}(t', X)$ and $B_2 \in \mathrm{Block}(t, X)$ with $V(B_1) \subseteq V(B_2)$ and $v \notin V(B_2)$, $h'(B_1) = h(B_2)$, and (3) for $B_2 \in \mathrm{Block}(t, X)$ containing $v$, $h(B_2) = \bigcup_{B_1 \in \mathrm{Block}(t', X), V(B_1) \subseteq V(B_2)} h(B_1)$.

▶ **Claim 13.** *For $\mathcal{X} \in \mathrm{Part}(t, X)$, $\mathcal{X} \in c[t, M]$ if and only if there exist a restriction $(g', h')$ of $(g, h)$ and $\mathcal{Y} \in c[t', (X, L_{res}, i, (g', h'))]$ such that (1) $v$ has neighbors on at most one component in each part of $\mathcal{Y}$, and (2) if $v$ has at least one neighbor in $G[B_t \setminus X]$, then $\mathcal{X}$ is the partition obtained from $\mathcal{Y}$ by, for parts $Y_1, \ldots, Y_m$ of $\mathcal{Y}$ containing components having a neighbor of $v$, removing all of $Y_1, \ldots, Y_m$ and adding a part that consists of all components of $G[B_t \setminus X]$ not contained in parts of $\mathcal{Y} \setminus \{Y_1, \ldots, Y_m\}$; and otherwise, $\mathcal{X} = \mathcal{Y} \cup \{\{v\}\}$.*

We update $r[t, M]$ as follows. Set $\mathcal{K} := \emptyset$. For a pair of functions $(g', h')$, we test whether $(g', h')$ is a restriction of $(g, h)$. Assume $(g', h')$ is a restriction of $(g, h)$. For each $\mathcal{Y} \in r[t', (X, L_{res}, i, (g', h'))]$, we check the two conditions for $(g', h')$ and $\mathcal{Y}$ in Claim 13, and if they are satisfied, then add the set $\mathcal{X}$ described in Claim 13 to $\mathcal{K}$; otherwise, skip it. The whole procedure can be done in time $2^{\mathcal{O}(wd^2)}$. After we do this for all possible candidates, we take a representative set of $\mathcal{K}$ using Proposition 12, and assign the resulting set to $r[t, M]$.

We claim that $r[t, M] \equiv c[t, M]$. Let $G_{out} := G - (V(G_t) \setminus B_t)$, $\mathcal{X} \in c[t, M]$, and $(S, L')$ be a partial solution with respect to $\mathcal{X}$, and suppose there exists $S_{out} \subseteq V(G) \setminus V(G_t)$ where $(G_t - (X \cup S), B_t \setminus X) \oplus (G_{out} - (X \cup S_{out}), B_t \setminus X)$ is a $d$-labeled $\mathcal{P}$-block graph respecting $(g, h)$. Every $(B_{t'} \setminus X)$-block of $G - (S \cup X \cup S_{out})$ is chordal as such a block is a $(B_t \setminus X)$-block of $G - (S \cup X \cup S_{out})$. Since $G - (S \cup X \cup S_{out})$ is chordal, by Proposition 10, $\mathbf{Aux}(G_{t'} - (X \cup S), B_{t'} \setminus X) \oplus \mathbf{Aux}(G_{out} - (X \cup S_{out}), B_{t'} \setminus X)$ has no cycles. Let $M_{res} := (X, L_{res}, i, (g', h'))$. As $r[t', M_{res}] \equiv c[t', M_{res}]$, there exist $\mathcal{Y} \in r[t', M_{res}]$ and a partial solution $(S', L'')$ with respect to $\mathcal{Y}$ such that $\mathbf{Inc}(\mathrm{Comp}(t', X), \mathcal{Y}) \sim \mathbf{Aux}(G_{t'} - (X \cup S'), B_{t'} \setminus X)$ has no cycles. By Theorem 11, $G - (S' \cup X \cup S_{out})$ is a $d$-labeled $\mathcal{P}$-block graph respecting $(g, h)$.

By the procedure, $\mathcal{X}_1$ where $\mathbf{Inc}(\mathrm{Comp}(t, X), \mathcal{X}_1) \sim \mathbf{Aux}(G_t - (X \cup S'), B_t \setminus X)$ is added to $\mathcal{K}$. And there exist $\mathcal{X}_2 \in r[t, M]$ and a partial solution $(S'', L''')$ with respect to $\mathcal{X}_2$ such that $G - (S'' \cup X \cup S_{out})$ is a $d$-labeled $\mathcal{P}$-block graph. Thus, $r[t, M] \equiv c[t, M]$.

**Total running time.** We denote $|V(G)|$ by $n$. Note that the number of nodes in $T$ is $\mathcal{O}(wn)$. For fixed $t \in V(T)$, there are at most $2^{w+1}$ possible choices for $X \subseteq B_t$, and for fixed $X \subseteq B_t$, there are at most $d^{w+1}$ possible functions $L$. Furthermore, the size of $\mathcal{F}(t, X, L)$ is bounded by $2^{\mathcal{O}(wd^2)}$. Thus, there are $\mathcal{O}(n \cdot k \cdot \max(2, d)^{w+1} \cdot 2^{\mathcal{O}(wd^2)})$ tables. In summary, the algorithm runs in time $\mathcal{O}(n \cdot k \cdot \max(2, d)^{w+1}) \cdot 2^{\mathcal{O}(wd^2)} \cdot k = 2^{\mathcal{O}(wd^2)} k^2 n$. ◀

## 5 Lower bound for fixed $d$

We showed that Bounded $\mathcal{P}$-Component Vertex Deletion and Bounded $\mathcal{P}$-Block Vertex Deletion admit single-exponential time algorithms parameterized by treewidth, whenever $\mathcal{P}$ is a class of chordal graphs. We now establish that, assuming the ETH, this is no longer the case when $\mathcal{P}$ contains a graph that is not chordal.

In the $k \times k$ Independent Set problem, one is given a graph $G = ([k] \times [k], E)$ over the $k^2$ vertices of a *$k$-by-$k$ grid*. We denote by $\langle i, j \rangle$ with $i, j \in [k]$ the vertex of $G$ in the $i$-th *row* and $j$-th *column*. The goal is to find an independent set of size $k$ in $G$ that contains exactly one vertex in each row. The Permutation $k \times k$ Independent Set problem is similar but with the additional constraint that the independent set should also contain exactly one vertex per column.

▶ **Theorem 14.** *If $\mathcal{P}$ contains the cycle graph on $\ell \geqslant 4$ vertices, then* Bounded $\mathcal{P}$-Component Vertex Deletion*, or* Bounded $\mathcal{P}$-Block Vertex Deletion*, is not solvable in time $2^{o(w \log w)} n^{\mathcal{O}(1)}$ on graphs of treewidth at most $w$ even for fixed $d = \ell$, unless the ETH fails.*

**Proof.** To prove this theorem, we reduce from Permutation $k \times k$ Independent Set which, like Permutation $k \times k$ Clique, cannot be solved in time $2^{o(k \log k)} k^{\mathcal{O}(1)}$ unless the ETH fails [13]. Let $G = ([k] \times [k], E)$ be an instance of Permutation $k \times k$ Independent Set. We assume that $\forall h, i, j \in [k]$ with $h \neq i$, $\langle i, j \rangle \langle h, j \rangle \in E$. Adding these edges does not change the Yes- and No-instances, but has the virtue of making Permutation $k \times k$ Independent Set equivalent to $k \times k$ Independent Set. We also assume that $\forall h, i, j \in [k]$, $\langle i, j \rangle \langle i, h \rangle \notin E$,

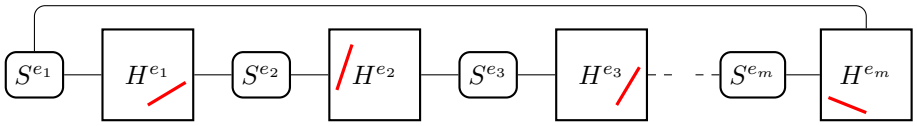



**Figure 1** A high-level schematic of $G'$ and $G''$. The $H^{e_i}$s only differ by a constant number of edges (in red/light gray) that encode their edge $e_i$ of $G$.

since at most one of $\langle i,j\rangle$ and $\langle i,h\rangle$ can be in a given solution. Let $m := |E| = \mathcal{O}(k^4)$ be the number of edges of $G$.

**Outline.** We build two graphs $G' = (V', E')$ and $G'' = (V', E'')$ with treewidth at most $(3d+4)k+6d-5 = \mathcal{O}(k)$, and $((3d-2)k^2+2k)m$ vertices, where the following are equivalent:

1. $G$ has an independent set of size $k$ with one vertex per row of $G$.
2. There is a set $S \subseteq V'$ of size at most $(3d-2)k(k-1)m$ such that each connected component of $G'-S$ has size at most $d$ and belongs to $\mathcal{P}$.
3. There is a set $S \subseteq V'$ of size at most $(3d-2)k(k-1)m$ such that each block of $G''-S$ has size at most $d$ and belongs to $\mathcal{P}$.

The overall construction of $G'$ and $G''$ will display $m$ *almost* copies of the encoding of an *edgeless* $G$ arranged in a cycle. Each copy embeds one distinct edge of $G$. The point of having the information of $G$ distilled edge by edge in $G'$ and $G''$ is to control the treewidth. This general idea originates from a paper of Lokshtanov et al. [12].

**Construction.** We first describe $G'$. As a slight abuse of notation, a gadget (and, more generally, a subpart of the construction) may refer to either a subset of vertices or to an induced subgraph. For each $e = \langle i^e, j^e\rangle\langle i'^e, j'^e\rangle \in E$, we detail the internal construction of $H^e$ and $S^e$ of Figure 1 and how they are linked to one another. Each vertex $v = \langle i,j\rangle$ of $G$ is represented by a gadget $H^e(v)$ on $3d-2$ vertices in $G'$: a path on $d-3$ vertices whose endpoints are $v^e_{-a}$ and $v^e_{-b}$, an isolated vertex $v^e_+$, and two disjoint cycles of length $d$. Observe that if $d=4$, then $v^e_{-a}$ and $v^e_{-b}$ is the same vertex. We add all the edges between $H^e(\langle i,j\rangle)$ and $H^e(\langle i,j'\rangle)$ for $i,j,j' \in [k]$ with $j \neq j'$. We also add all the edges between $H^e(\langle i^e, j^e\rangle)$ and $H^e(\langle i'^e, j'^e\rangle)$. We call $H^e$ the graph induced by the union of every $H^e(v)$, for $v \in V(G)$. The *row/column selector* gadget $S^e$ consists of a set $S^e_r$ of $k$ vertices with one vertex $r^e_i$ for each row index $i \in [k]$, and a set $S^e_c$ of $k$ vertices with one vertex $c^e_j$ for each column index $j \in [k]$. The gadget $S^e$ forms an independent set of size $2k$. We arbitrarily number the edges of $G$: $e_1, e_2, \ldots, e_m$. For each $h \in [m]$ and $v = \langle i,j\rangle \in V$, we link $v^{e_h}_{-a}$ to $r^{e_h}_i$ (the row index of $v$) and $v^{e_h}_{-b}$ to $c^{e_h}_j$ (the column index of $v$). We also link, for every $h \in [m-1]$, $v^{e_h}_+$ to $r^{e_{h+1}}_i$ and to $c^{e_{h+1}}_j$, and $v^{e_m}_+$ to $r^{e_1}_i$ and to $c^{e_1}_j$. That concludes the construction (see Figure 2). To obtain $G''$ from $G'$, we add the edges $c^{e_h}_j c^{e_h}_{j+1}$ for every $h \in [m]$ and $j \in [k-1]$. We ask for a deletion set $S$ of size $s := (3d-2)k(k-1)m$.

**Treewidth of $G'$ and $G''$.** For any edge $e \in E$, we set $H(e) := H^e(\langle i^e, j^e\rangle) \cup H^e(\langle i'^e, j'^e\rangle)$. For any $i \in [m-1]$, we set $\tilde{S}_i := S^{e_1} \cup S^{e_i} \cup S^{e_{i+1}}$, and $\tilde{S}_m := S^{e_1} \cup S^{e_m}$. For each $e \in E$, and $i \in [k]$, $H^e(i)$ denotes the union of the $H^e(v)$ for all vertices $v$ of the $i$-th row. Here is a path decomposition of $G'$ and $G''$:

$$\tilde{S}_1 \cup H(e_1) \cup H^{e_1}(1) \to \tilde{S}_1 \cup H(e_1) \cup H^{e_1}(2) \to \ldots \to \tilde{S}_1 \cup H(e_1) \cup H^{e_1}(k) \to$$
$$\vdots$$
$$\tilde{S}_m \cup H(e_m) \cup H^{e_m}(1) \to \tilde{S}_m \cup H(e_m) \cup H^{e_m}(2) \to \ldots \to \tilde{S}_m \cup H(e_m) \cup H^{e_m}(k).$$

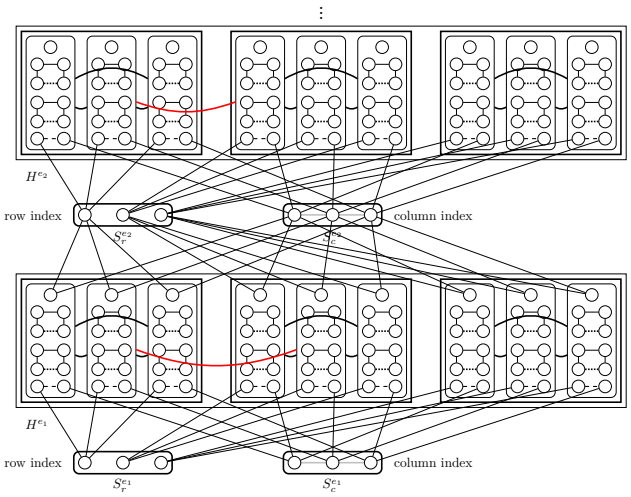



**Figure 2** The overall picture of $G'$ and $G''$ with $k = 3$. Dotted edges are subdivided $d-4$ times; if $d = 4$, they are simply edges. Dashed edges are subdivided $d-5$ times; if $d = 4$, the two endpoints are in fact a single vertex. Edges between two boxes link each vertex of one box to each vertex of the other box. The gray edges in the column selectors $S_c^{e_h}$ are only present in $G''$.

As, for any $h \in [m]$, $|\tilde{S}_h| \leqslant 6k$, $|H(e_h)| = 2(3d-2)$, and $|H^{e_h}(i)| \leqslant (3d-2)k$ for any $i \in [k]$, the size of a bag is bounded by $\max_{h\in[m],i\in[k]} |\tilde{S}_h \cup H(e_h) \cup H^{e_h}(i)| \leqslant 6k+2(3d-2)+(3d-2)k = (3d+4)k+6d-4$.

**Correctness.** If there is an independent set $I$ of size $k$ in $G$, a solution to a Bounded $\mathcal{P}$-Component Vertex Deletion or Bounded $\mathcal{P}$-Block Vertex Deletion instance can be obtained by deleting from each $H^e$ every $H^e(v)$ such that $v \notin I$.

We show that $2 \Rightarrow 1$ and $3 \Rightarrow 1$. We assume that there is a set $S \subseteq V'$ of size at most $s$ such that all the blocks of $G'' - S$ (resp. $G' - S$) have size at most $d$. We note that this corresponds to assuming condition 3 (resp. a weaker assumption than condition 2) holds. We show that there are at most $3d-2$ vertices of $H^e(i)$ remaining in $G'' - S$ (or $G' - S$). Assume, for the sake of contradiction, that $H^e(i) - S$ contains at least $3d-1$ vertices. Observe that $H^e(i) - S$ cannot contain at least one vertex from three distinct $H^e(u)$, $H^e(v)$, and $H^e(w)$ (with $u$, $v$ and $w$ in the $i$-th row of $G$), since then $H^e(i) - S$ would be 2-connected (and of size $> d$). For the same reason, $H^e(i) - S$ cannot contain at least two vertices in $H^e(u)$ and at least two vertices in another $H^e(v)$. Therefore, the only way of fitting $3d-1$ vertices in $H^e(i) - S$ is the $3d-2$ vertices of an $H^e(u)$ plus one vertex from some other $H^e(v)$. But then, this vertex of $H^e(v)$ would form, together with one $C_d$ of $H^e(u)$, a 2-connected subgraph of $G'' - S$ (or $G' - S$) of size $d+1$. Now, we know that $|H^e(i) \cap S| \geqslant (3d-2)(k-1)$. As there are precisely $mk$ sets $H^e(i)$ in $G'$ (and they are disjoint), it further holds that $|H^e(i) \cap S| = (3d-2)(k-1)$, since otherwise $S$ would contain strictly more than $s = (3d-2)k(k-1)m$ vertices. Thus, $H^e(i) - S$ contains exactly $3d-2$ vertices. By the previous remarks, $H^e(i) - S$ can only consist of the $3d-2$ vertices of the same $H^e(u)$ or $3d-3$ vertices of $H^e(u)$ plus one vertex from another $H^e(v)$. In fact, the latter case is not possible, since the vertex of $H^e(v)$ would form, with at least one remaining $C_d$ of the $3d-3$ vertices of $H^e(u)$, a 2-connected subgraph of $G'' - S$ (or $G' - S$) of size $d+1$. This is why we needed two disjoint $C_d$s in the construction instead of just one. So far, we have proved that, assuming condition 2 or condition 3 holds, for any $e \in E$ and $i \in [k]$, $H^e(i) \cap S = H^e(v_{i,e})$ for some vertex $v_{i,e}$ of the $i$-th row of $G$, and for any $e \in E$, $S^e \cap S = \emptyset$.

In what follows, we show that $v_{i,e}$ does not depend on $e$. Formally, we want to show that there is a $v_i$ such that, for any $e \in E$, $v_{i,e} = v_i$. Observe that it is enough to derive that, for any $h \in [m]$, $v_{i,e_h} = v_{i,e_{h+1}}$ (with $e_{m+1} = e_1$). Let $j \in [k]$ (resp. $j' \in [k]$) be the column of $v_{i,e_h}$ (resp. $v_{i,e_{h+1}}$) in $G$. We first assume condition 2 holds. For any $h \in [m]$, $v_{i,e_h}{}_{+}^{e_h}$, $r_i^{e_{h+1}}$, $c_{j'}^{e_{h+1}}$, $c_j^{e_{h+1}}$ plus the path $P_{v_{i,e_{h+1}}}^{e_{h+1}}$ (between $v_{i,e_{h+1}}{}_{-a}^{e_{h+1}}$ and $v_{i,e_{h+1}}{}_{-b}^{e_{h+1}}$) induces a path (in particular, a connected subgraph) of size $d+1$ in $G'' - S$, unless $j = j'$ (with $e_{m+1} = e_1$). Therefore, $j = j'$. As $v_{i,e_h}$ and $v_{i,e_{h+1}}$ have the same column $j$ and the same row $i$ in $G$, $v_{i,e_h} = v_{i,e_{h+1}}$. Showing the same property under 3 is done similarly. We can now safely define $v_i := v_{i,e}$ and conclude by proving that $\{v_1, v_2, \ldots, v_k\}$ is a clique. ◀

## References


1 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Optimal algorithms for hitting (topological) minors on graphs of bounded treewidth. *CoRR*, abs/1704.07284, 2017. arXiv:1704.07284.

2 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.

3 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k$ n 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. doi:10.1137/130947374.

4 Édouard Bonnet, Nick Brettell, O-joung Kwon, and Dániel Marx. Parameterized vertex deletion problems for hereditary graph classes with a block property. In *Graph-Theoretic Concepts in Computer Science*, volume 9941 of *Lecture Notes in Comput. Sci.*, pages 233–244, 2016.

5 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. *ArXiv e-prints*, 2017. arXiv:1704.06757.

6 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.

7 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal 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.

8 Reinhard Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg, fourth edition, 2010. doi:10.1007/978-3-642-14279-6.

9 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.

10 Jessica Enright and Kitty Meeks. Deleting edges to restrict the size of an epidemic: A new application for treewidth. In Zaixin Lu, Donghyun Kim, Weili Wu, Wei Li, and Ding-Zhu Du, editors, *Combinatorial Optimization and Applications - 9th International Conference, COCOA 2015, Houston, TX, USA, December 18-20, 2015, Proceedings*, volume 9486 of *Lecture Notes in Computer Science*, pages 574–585. Springer, 2015. doi:10.1007/978-3-319-26626-8_42.

11 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algo-*

*rithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 142–151. SIAM, 2014. doi:10.1137/1.9781611973402.10.

**12** Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 777–789. SIAM, 2011. doi:10.1137/1.9781611973082.61.

**13** Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 760–776. SIAM, 2011. doi:10.1137/1.9781611973082.60.

# On the Parameterized Complexity of Red-Blue Points Separation*†

**Édouard Bonnet[1], Panos Giannopoulos[2], and Michael Lampis[3]**

1 **Middlesex University, Department of Computer Science, London, UK**
edouard.bonnet@dauphine.fr
2 **Middlesex University, Department of Computer Science, London, UK**
p.giannopoulos@mdx.ac.uk
3 **Université Paris-Dauphine, PSL Research University, CNRS, LAMSADE, Paris, France**
michail.lampis@dauphine.fr

## Abstract

We study the following geometric separation problem: Given a set $\mathcal{R}$ of red points and a set $\mathcal{B}$ of blue points in the plane, find a minimum-size set of lines that separate $\mathcal{R}$ from $\mathcal{B}$. We show that, in its full generality, parameterized by the number of lines $k$ in the solution, the problem is unlikely to be solvable significantly faster than the brute-force $n^{O(k)}$-time algorithm, where $n$ is the total number of points. Indeed, we show that an algorithm running in time $f(k)n^{o(k/\log k)}$, for any computable function $f$, would disprove ETH. Our reduction crucially relies on selecting lines from a set with a large number of different slopes (i.e., this number is not a function of $k$).

Conjecturing that the problem variant where the lines are required to be axis-parallel is FPT in the number of lines, we show the following preliminary result. Separating $\mathcal{R}$ from $\mathcal{B}$ with a minimum-size set of axis-parallel lines is FPT in the size of either set, and can be solved in time $O^*(9^{|\mathcal{B}|})$ (assuming that $\mathcal{B}$ is the smallest set).

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** red-blue points separation, geometric problem, W[1]-hardness, FPT algorithm, ETH-based lower bound

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.8

## 1 Introduction

We study the parameterized complexity of the following Red-Blue Separation problem: Given a set $\mathcal{R}$ of red points and a set $\mathcal{B}$ of blue points in the plane and a positive integer $k$, find a set of at most $k$ lines that together separate $\mathcal{R}$ from $\mathcal{B}$ (or report that such a set does not exist). Separation here means that each cell in the arrangement induced by the lines in the solution is either monochromatic, i.e., contains points of one color only, or empty. Equivalently, $\mathcal{R}$ is separated from $\mathcal{B}$ if every straight-line segment with one endpoint in $\mathcal{R}$ and the other one in $\mathcal{B}$ is intersected by at least one line in the solution. Note here that we opt for *strict* separation that is, no point in $\mathcal{R} \cup \mathcal{B}$ is on a separating line. Let $n := |\mathcal{R} \cup \mathcal{B}|$.

The variant where the separating lines sought must be axis-parallel will be simply referred to as Axis-Parallel Red-Blue Separation.

* Research partially supported by EPSRC grant EP/N029143/1.
† A full version of the paper is available at https://arxiv.org/abs/1710.00637.


Apart from being interesting in its own right, Red-Blue Separation is also directly motivated by the problem of univariate discretization of continuous variables in the context of machine learning [4, 8]. For example, its two-dimensional version models problem instances with decision tables of two real-valued attributes and a binary decision function. The lines to be found represent cut points determining a partition of the values into intervals and one opts for a minimum-size set of cuts that is consistent with the given decision table. The problem is also known as *minimum linear classification*; see [9] for an application in signal processing. For the case where $k = 1$ and $k = 2$, Red-Blue Separation is solvable in $O(n)$ and $O(n \log n)$ time respectively [7]. When $k$ is part of the input, it is known to be NP-hard [10] and APX-hard [2] even for the axis-parallel variant. The latter also admits a 2-approximation [2].

**Results.** We first show that Red-Blue Separation is W[1]-hard in the solution size $k$ and that it cannot be solved in $f(k)n^{o(k/\log k)}$ time (for any computable function $f$) unless ETH fails. Our reduction is from Structured 2-Track Hitting Set, see Section 2, which has been recently used for showing hardness for another classical geometric optimization problem [1]. Then, in Section 3, we show that Axis-Parallel Red-Blue Separation is FPT in the size of either of $\mathcal{R}$ and $\mathcal{B}$. Our algorithm is simple and is based on reducing the problem to $9^{|\mathcal{B}|+2}$ instances of 2-SAT (assuming, w.l.o.g., that $\mathcal{B}$ is the smallest set).

**Related work.** The following monochromatic points separation problem has also been studied: Given a set of points in the plane, find a smallest set of lines that separates every point from every other point in the set (i.e., each cell in the induced arrangement must contain at most one point). It has been shown to be NP-hard [5], APX-hard [2] and, in the axis-parallel case, to admit a 2-approximation [2]. Very recently, the problem has been also shown to admit an $\mathrm{OPT} \log \mathrm{OPT}$-approximation [6]. Note here that it is trivially FPT in the number of lines, as the number of cells in the arrangement of $k$ lines is at most $\Theta(k^2)$. For results on several other related separation problems, see [7, 3].

## 2 Parameterized hardness for arbitrary slopes

We show that Red-Blue Separation is unlikely to be FPT with respect to the number of lines $k$ and establish that, unless the ETH fails, the $n^{O(k)}$-time brute-force algorithm is almost optimal. We reduce from Structured 2-Track Hitting Set [1], see below.

For positive integers $x$, $y$, let $[x]$ be the set of integers between 1 and $x$, and $[x, y]$ the set of integers between $x$ and $y$. For a totally ordered (finite) set $X$, an *$X$-interval* is any subset of $X$ of consecutive elements. In the 2-Track Hitting Set problem, the input consists of an integer $k$, two totally ordered ground sets $A$ and $B$ of the same cardinality, and two sets $\mathcal{S}_A$ of $A$-intervals and $\mathcal{S}_B$ of $B$-intervals. The elements of $A$ and $B$ are in one-to-one correspondence $\phi : A \to B$ and each pair $(a, \phi(a))$ is called a *2-element*. The goal is to decide if there is a set $S$ of $k$ 2-elements such that the first projection of $S$ is a hitting set of $\mathcal{S}_A$, and the second projection of $S$ is a hitting set of $\mathcal{S}_B$. We will refer to the interval systems $(A, \mathcal{S}_A)$ and $(B, \mathcal{S}_B)$ as track A and track B.

Structured 2-Track Hitting Set (S2-THS for short) is the same problem with color classes over the 2-elements and a restriction on the one-to-one mapping $\phi$. Given two integers $k$ and $t$, $A$ is partitioned into $(C_1, C_2, \ldots, C_k)$ where $C_j = \{a_1^j, a_2^j, \ldots, a_t^j\}$ for each $j \in [k]$. $A$ is ordered: $a_1^1, a_2^1, \ldots, a_t^1, a_1^2, a_2^2, \ldots, a_t^2, \ldots, a_1^k, a_2^k, \ldots, a_t^k$. We define $C'_j := \phi(C_j)$ and $b_i^j := \phi(a_i^j)$ for all $i \in [t]$ and $j \in [k]$. We now impose that $\phi$ is such that, for each

$j \in [k]$, the set $C'_j$ is a $B$-interval. That is, $B$ is ordered: $C'_{\sigma(1)}, C'_{\sigma(2)}, \ldots, C'_{\sigma(k)}$ for some permutation on $[k]$, $\sigma \in \mathfrak{S}_k$. For each $j \in [k]$, the order of the elements within $C'_j$ can be described by a permutation $\sigma_j \in \mathfrak{S}_t$ such that the ordering of $C'_j$ is: $b^j_{\sigma_j(1)}, b^j_{\sigma_j(2)}, \ldots, b^j_{\sigma_j(t)}$. In what follows, it will be convenient to see an instance of S2-THS as a tuple $\mathcal{I} = (k \in \mathbb{N}, t \in \mathbb{N}, \sigma \in \mathfrak{S}_k, \sigma_1 \in \mathfrak{S}_t, \ldots, \sigma_k \in \mathfrak{S}_t, \mathcal{S}_A, \mathcal{S}_B)$, where $\mathcal{S}_A$ is a set of $A$-intervals and $\mathcal{S}_B$ is a set of $B$-intervals. We denote by $[a^j_i, a^{j'}_{i'}]$ (resp. $[b^j_i, b^{j'}_{i'}]$) all the elements $a \in A$ (resp. $b \in B$) such that $a^j_i \leq_A a \leq_A a^{j'}_{i'}$ (resp. $b^j_i \leq_B b \leq_B b^{j'}_{i'}$).

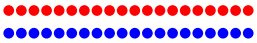
If one deconstructs S2-THS, one finds intervals, a permutation of the color classes $\sigma$, and $k$ permutations $\sigma_j$'s of the elements within the classes. Intervals, thanks to their geometric nature, can be realized by two red points which have to be separated from a diagonal of blue points (see Figure 2), while permutation $\sigma$, being on $k$ elements, can be designed straightforwardly without blowing-up the size of the solution (see Figure 3). For these gadgets, we would like to force the chosen lines to be axis-parallel. We obtain this by surrounding them with *long alleys* made off long red paths parallel and next to long blue paths (see Figure 1). The main challenge is to get the permutations $\sigma_j$'s on $t$ elements. To attain this, we match a selected line $L_i$ (corresponding to an element of index $i \in [t]$) to a specific angle $\alpha_i$, which *leads* to the intended position of the element of index $\sigma_j(l) = i$, for some $l \in [t]$ (see Figure 4). Note that the depicted gadget actually links the element of index $i$ to elements equal to or smaller than the element indexed at $\sigma_j(l)$. By combining two of these gadgets we can easily obtain only the intended position (see Figure 5).

▶ **Theorem 1.** Red-Blue Separation *is $W[1]$-hard w.r.t. the number of lines $k$, and unless ETH fails, cannot be solved in time $f(k)n^{o(k/\log k)}$ for any computable function $f$.*

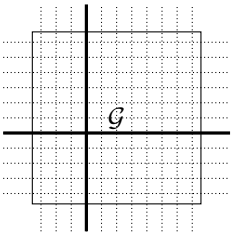
**Proof.** We reduce from S2-THS, which is $W[1]$-hard and has the above lower bound under ETH [1]. Let $\mathcal{I} = (k \in \mathbb{N}, t \in \mathbb{N}, \sigma \in \mathfrak{S}_k, \sigma_1 \in \mathfrak{S}_t, \ldots, \sigma_k \in \mathfrak{S}_t, \mathcal{S}_A, \mathcal{S}_B)$ be an instance of S2-THS. We build an instance $\mathcal{J} = (\mathcal{R}, \mathcal{B}, 6k + 14)$ of Red-Blue Separation such that $\mathcal{I}$ is a YES-instance for S2-THS if and only if $\mathcal{R}$ and $\mathcal{B}$ can be separated with $6k + 14$ lines.

The points in $\mathcal{R}$ and $\mathcal{B}$ will have rational coordinates. More precisely, most points will be pinned to a $z$-by-$z$ grid where $z$ is polynomial in the size of $\mathcal{I}$. The rest will have rational coordinates with nominator and denominator polynomial in $z$. Let $\Gamma$ be the $z$-by-$z$ grid corresponding to the set of points with coordinates in $[z] \times [z]$. We call *horizontally* (resp. vertically) *consecutive points* a set of points of $\Gamma$ with coordinates $(a, y), (a+1, y), \ldots (b-1, y), (b, y)$ for $a, b, y \in [z]$ and $a < b$ (resp. $(x, a), (x, a+1), \ldots (x, b-1), (x, b)$ for $a, b, x \in [z]$ and $a < b$). We denote those points by $\mathcal{C}(a \to b, y)$ (resp. $\mathcal{C}(x, a \to b)$).

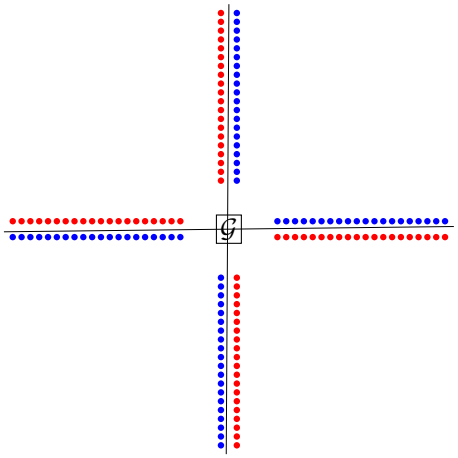
**Long alley gadgets.** In the gadgets encoding the intervals (see next paragraph), we will need to restrict the selected separating lines to be almost horizontal or almost vertical. To enforce that, we use the *long alley* gadgets. A *horizontal long alley* gadget is made of $\ell$ horizontally consecutive red points $\mathcal{C}(a \to a + \ell - 1, y)$ and $\ell$ horizontally consecutive blue points $\mathcal{C}(a \to a + \ell - 1, y')$ with $a, a + \ell - 1, y \neq y' \in [z]$ (see Figure 1a). A *vertical long alley* is defined analogously. Long alleys are called so because $\ell \gg |y - y'|$ thus, separating the red points from the blue points of a horizontal (resp. vertical) long alley with a budget of only one line, requires the line to be almost horizontal (resp. vertical). The use of the long alleys will be the following. Let $\mathcal{G}$ be a gadget for which we wish the separating lines to be almost horizontal or vertical. Say, $\mathcal{G}$ occupies a $g$-by-$g$ subgrid of $\Gamma$ (with $g \ll z$). We place four long alley gadgets to the left, top, right, and bottom of $\mathcal{G}$: horizontal ones to the left and right, vertical ones to the top and bottom (as depicted in Figure 1c). The left horizontal (resp. bottom vertical) long alley starts at the $x$-coordinate (resp. $y$ coordinate)



**(a)** A horizontal long alley. Separating this subset of points with one line requires the line to be almost horizontal.




**(b)** Zoom in gadget $\mathcal{G}$. The horizontal (resp. vertical) lines are *entering* the gadget to the left (resp. at the top) and *exiting* it to the right (resp. at the bottom) with almost the same $y$-coordinates (resp. $x$-coordinates). Possible lines are thin dotted while an actual choice of two lines is shown in bold.




**(c)** We put four *long alleys* to the left, top, right, and bottom of gadget $\mathcal{G}$ where we want the selected lines to be almost axis parallel.

**Figure 1** The long alley gadget and its use in combination with another gadget.

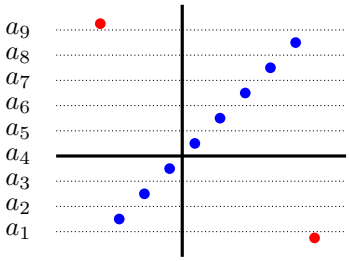
of 1, whereas the right horizontal (resp. top vertical) long alley ends at the $x$-coordinate (resp. $y$ coordinate) of $z$; see Figure 5, where the long alleys are depicted by thin rectangles.

Note that we will not surround each and every gadget of the construction by four long alleys. At some places, it will indeed be crucial that the lines can have arbitrary slopes.

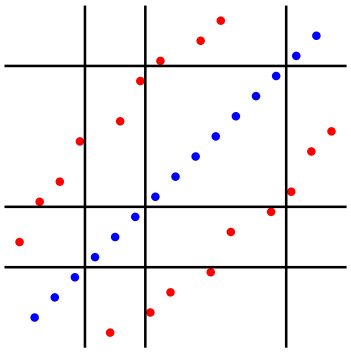
**Interval gadgets and encoding track A.** The elements of $A$ are represented by a diagonal of $kt-1$ blue points. More precisely, we add the points $(x_0^A, y_0^A), (x_0^A+4, y_0^A+4), (x_0^A+8, y_0^A+8), \ldots, (x_0^A+4kt-8, y_0^A+4kt-8)$ to $\mathcal{B}$ for some offset $x_0^A, y_0^A \in [z]$ that we will specify later. We think those points as going from the first $(x_0^A, y_0^A)$ to the last $(x_0^A+4kt-8, y_0^A+4kt-8)$. An almost horizontal (resp. vertical) line just below (resp. just to the left of) the $s$-th blue point of this diagonal translates as selecting the $s$-th element of $A$ in the order fixed by $\leq_A$. The almost horizontal (resp. vertical) line just above (resp. just to the right of) the last blue point corresponds to selecting the $kt$-th, i.e., last, element of $A$.

For each interval $[a_i^j, a_{i'}^{j'}]$ in $\mathcal{S}_A$ (for some $i, i' \in [k]$, $j, j' \in [t]$), that is, the interval between the $s := ((j-1)t+i)$-th and the $s' := ((j'-1)t+i')$-th elements of $A$, we add two red points: one at $(x_0^A+4s-7, y_0^A+4s'-5)$ and one at $(x_0^A+4s'-5, y_0^A+4s-7)$ (see Figure 2a for one interval gadget and Figure 2b for track $A$). Let $R([a_i^j, a_{i'}^{j'}])$ be this pair of red points. Informally, one red point has its projection along the $x$-axis just to the left of the $s$-th blue point and its projection along the $y$-axis just above the $s'$-th blue point; the other one has its projection along the $x$-axis just to the right of the $s'$-th blue point and its projection along the $y$-axis just below the $s$-th blue point. For technical reasons, we add, for every $j \in [k]$, the pair $R([a_1^j, a_t^j])$ encoding the interval formed by all the elements of the $j$-th color class of $A$. Adding these intervals to $\mathcal{S}_A$ does not constrain the problem more.

We surround this encoding of track $A$, which we denote by $\mathcal{G}(A)$, with $4k$ long alleys, whose width is $4t-4$, from $x$-coordinates $x_0^A+4(j-1)t-2$ to $x_0^A+4jt-6$ for vertical alleys (from $y$-coordinates $y_0^A+4(j-1)t-2$ to $y_0^A+4jt-6$ for horizontal alleys). We




**(a)** The interval gadget corresponding to $[a_1, a_9] = \{a_1, \ldots, a_9\}$. In thin dotted, the mapping between elements and potential lines. In bold, the choice of the lines corresponding to picking $a_4$. If one wants to separate these points with two lines, one almost horizontal and one almost vertical, the choice of the former imposes the latter.



**(b)** The interval gadgets put together. A representation of one track. Separating these points with the fewest axis-parallel lines requires taking the horizontal and vertical lines associated to a minimum hitting set.

**Figure 2** To the left, one interval. To the right, several put together to form one track.

alternate red-blue[1] alleys and blue-red alleys for two contiguous alleys so that there is no need to separate one from the other. We start with a red-blue alley for the left horizontal and top vertical groups of alleys, and with a blue-red alley for the right horizontal and bottom vertical. This last detail is not in any way crucial but permits the construction to be defined uniquely and consistently with the choices of Figure 1c. This, together with the description of long alleys in the previous paragraph, fully defines the $4k$ long alleys (see Figure 5).

The general intention is that in order to separate those two red points from the blue diagonal with a budget of two almost axis-parallel lines, one should take two lines (one almost horizontal and one almost vertical) corresponding to the selection of the same element of $A$ which hits the corresponding interval. In particular, taking two almost horizontal lines (resp. two almost vertical lines) is made impossible due to those vertical (resp. horizontal) long alleys. More precisely, the intended pairs of lines separating the red points $R([a_i^j, a_{i'}^{j'}])$ from the blue diagonal are of the form $x = x_0^A + 4\hat{s} - 6, y = y_0^A + 4\hat{s} - 6$ for $\hat{s} \in [s, s']$. Furthermore, the $4k$ long alleys force a pair of (almost) horizontal and vertical lines corresponding to one element per color class to be taken.

For any $s \in [tk]$, $i \in [t]$, and $j \in [k]$, such that $s = (j-1)t + i$, let $\mathrm{HL}(s)$ be the horizontal line of equation $y = y_0^A + 4s - 6$ and $\mathrm{VL}(s)$ the vertical line of equation $x = x_0^A + 4s - 6$. They *correspond* to selecting $a_i^j$, the $i$-th element in the $j$-th color class of $A$. The goal of the remaining gadgets is to ensure that when the lines $\mathrm{HL}(s)$ and $\mathrm{VL}(s)$ (with $s = (j-1)t + i$) are chosen, additional lines corresponding to selecting element $b_i^j$ of $B$ have to be expressly selected. We define $\mathrm{HL} := \{\mathrm{HL}(s) \mid s \in [tk]\}$ and $\mathrm{VL} := \{\mathrm{VL}(s) \mid s \in [tk]\}$.

**Encoding inter-class permutation $\sigma$.** To encode the permutation $\sigma$ of the $k$ color classes of $\mathcal{I}$, we allocate a square subgrid of the same dimension as the space used for the encoding of track $A$, roughly $4tk$-by-$4tk$, and we place it to the right of $A$ right as depicted in Figure 5. This square subgrid is naturally and regularly split into $k^2$ smaller square subgrids of equal dimension (roughly $4t$-by-$4t$). This decomposition can be seen as the $k$ color classes of

[1] i.e., for horizontal (resp. vertical) alleys, the red points are above (resp. to the left of) the blue points.

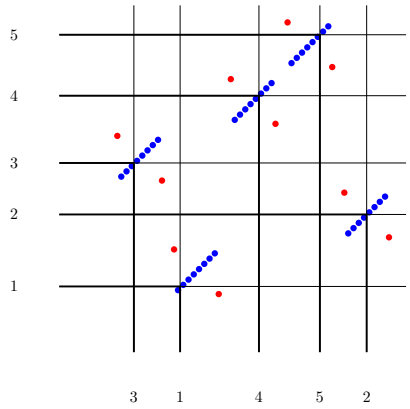



**Figure 3** Encoding permutation $\sigma = 31452$. The choices within the five color classes are transferred from almost horizontal lines to almost vertical ones. This way, we obtain the desired reordering of the color classes.

$\mathcal{I}$, or equivalently, the $k$-by-$k$ crossing[2] obtained by drawing horizontal lines between two contiguous horizontal long alleys and vertical lines between two contiguous vertical long alleys. We only put points in exactly one smaller square subgrid per column and per row. Let $\sigma := \sigma(1)\sigma(2)\dots\sigma(k)$ and $\mathrm{Cell}(a,b)$ be the smaller square subgrid in the $a$-th row and $b$-th column of the $k$-by-$k$ crossing. For each $j \in [k]$, we put in $\mathrm{Cell}(j,\sigma(j))$ a diagonal of $t-1$ blue points and two red points corresponding to the full interval $[a_1^j, a_t^j]$ (see Figure 3). We denote by $\mathcal{G}(\sigma)$ those sets of red and blue points in the encoding of $\sigma$. We surround $\mathcal{G}(\sigma)$ by $2k$ vertical long alleys similar to the $2k$ long alleys surrounding $\mathcal{G}(A)$. Notice that $\mathcal{G}(\sigma)$ and $\mathcal{G}(A)$ share the same $2k$ surrounding horizontal long alleys.

The way the gadget $\mathcal{G}(\sigma)$ works is quite intuitive. Given $k$ choices of horizontal lines originating from a separation in $\mathcal{G}(A)$ and a budget of $k$ extra lines for the separation within $\mathcal{G}(\sigma)$, the only option is to copy with the vertical line the choice of the horizontal line. It results in a vertical propagation of the initial choices accompanied by the desired reordering of the *color classes*. The vertical line matching the choice of $\mathrm{HL}(s)$ in the corresponding cell of $\mathcal{G}(\sigma)$ is denoted by $\mathrm{VL}'(s)$. Let $\mathrm{VL}' := \{\mathrm{VL}'(s) \mid s \in [tk]\}$. Note that corresponding lines in VL and in VL$'$ have a different order from left to right.

**Encoding of the intra-class permutations $\sigma_j$'s and track B.** If the encoding of permutation $\sigma$ is conceptually simple, the number of intended lines separating red and blue points in $\mathcal{G}(\sigma)$ has to be linear in the number of permuted elements. Since we wish to encode a permutation $\sigma_j$ (for every $j \in [k]$) on $t$ elements, we cannot use the same mechanism as it would blow-up our parameter dramatically and would not result in an FPT reduction.

For the gadget $\mathcal{G}_{\approx v}(\sigma_j)$ partially encoding the permutation $\sigma_j$, we will crucially use the fact that separating lines can have arbitrary slopes. Slightly to the right (at distance at least $\ell$) of the vertical line bounding the right end of $\mathcal{G}(\sigma)$ and far in the south direction, we place a gadget $\mathcal{G}(B)$ encoding track B similarly to the encoding of track A up to some symmetry that we will make precise later. We also incline the whole encoding of track B with a small, say 5, degree angle, in a way that its top-left corner is to the right of its bottom-left corner. We round up the real coordinates that this rotation incurs to rationals at distance

[2] we use this term informally to avoid confusion with what we have been calling *grids* so far.

less than, say, $(kt)^{-10}$. We denote by $\hat{v}$ the distance along the $y$-axis between $\mathcal{G}(\sigma)$ and $\mathcal{G}(B)$. Eventually $\hat{v}$ will be chosen much larger than $\Theta(kt)$, which is the size of $\mathcal{G}(A)$, $\mathcal{G}(B)$, $\mathcal{G}(\sigma)$. Below $\mathcal{G}(\sigma)$ at a distance $2\hat{v}$ along the $y$-axis, we place gadgets $\mathcal{G}_{\approx_v}(\sigma_j)$'s; from left to right, we place $\mathcal{G}_{\approx_v}(\sigma_{\sigma(1)})$, $\mathcal{G}_{\approx_v}(\sigma_{\sigma(2)})$, $\ldots$, $\mathcal{G}_{\approx_v}(\sigma_{\sigma(k)})$ such that for every $i \in [k]$, $G_{\approx_v}(\sigma_{\sigma(i)})$ falls below the $i$-th column of the $k$-by-$k$ crossing of $\mathcal{G}(\sigma)$. Gadgets $\mathcal{G}_{\approx_v}(\sigma_j)$'s are represented by small round shapes in Figure 5. Notwithstanding what is drawn on the overall picture, the $\mathcal{G}_{\approx_v}(\sigma_j)$'s can be all placed at the same $y$-coordinates. Let $y_1 := y_0 - 2\hat{v}$ (the exact value of $y_1$ is not crucial). Also, we represent track B slanted by a 45 degree angle, instead of the actual 5 degree angle, to be able to fit everything on one page and convey the main ideas of the construction. In general, for the figure to be readable, the true proportions are not respected. The size of every gadget is much smaller than the distance between two different groups of gadgets, so that every line *entering* a gadget traverses it in an axis-parallel fashion.

Gadget $\mathcal{G}_{\approx_v}(\sigma_j)$ is built as follows. For each $i \in [t]$ and $j \in [k]$, we draw a fictitious points $p_i^j$ corresponding to the intersection of a close to vertical line corresponding to picking element $b_i^j$ in gadget $\mathcal{G}(B)$ with the bottom end of $\mathcal{G}(B)$. From left to right, the $p_i^j$'s have the same order as the $b_i^j$'s in $(B, \leq_B)$. For every $s = (j-1)t + i$ (with $j \in [k]$ and $i \in [t]$), let $q_i^j$ be the point of $y$-coordinate $y_1$ on the line $\mathrm{VL}'(s)$. We define the line $\mathrm{SL}(s)$ as going through $p_i^j$ and $q_i^j$, and set $\mathrm{SL} := \{\mathrm{SL}(s) \mid s \in [tk]\}$. We add two blue points just to the left and just to the right of $q_i^j$ at distance $\epsilon := z^{-10}$. We add two blue points on line $\mathrm{SL}(s)$, one to the left of $q_i^j$ and one to the right of $q_i^j$. Finally, we place two red points for each $\mathcal{G}_{\approx_v}(\sigma_j)$ at the bottom-left and top-right of the gadget (see Figure 4). In the figure, the lines in SL form a large angle with the $y$-axis, while in fact they are quite close to a 5 degree angle and behave like *relatively vertical*[3] lines within $\mathcal{G}(B)$ (since $\mathcal{G}(B)$ is also inclined by 5 degrees).

Assuming that line $\mathrm{VL}'(s = (j-1)t + i)$ has been selected, it can be observed from Figure 4 that separating the red points from the blue points in $\mathcal{G}_{\approx_v}(\sigma_j)$ with a budget of one additional line requires to take a line crossing $\mathrm{VL}'(s)$ at (or very close to) $q_i^j$ and with a higher or equal slope to $\mathrm{SL}(s)$. It is not quite what we wanted. What we achieved so far is only to link the *choice of* $a_i^j$ with the *choice of an element smaller or equal to* $b_i^j$. We will use a symmetry $\mathcal{G}_{\approx_h}(\sigma_j)$ of gadget $\mathcal{G}_{\approx_v}(\sigma_j)$ to get the other inequality so that choosing some lines corresponding to $a_i^j$ actually forces to take some lines corresponding to $b_i^j$.

We add a gadget $\mathcal{G}(\mathrm{id})$ below the $\mathcal{G}_{\approx_v}(\sigma_j)$'s. $\mathcal{G}(\mathrm{id})$ is obtained by mimicking $\mathcal{G}(\sigma)$ for the identity permutation. We surround $\mathcal{G}(\mathrm{id})$ by $2k$ new horizontal long alleys. The horizontal line matching the choice of $\mathrm{VL}'(s)$ in $\mathcal{G}(\mathrm{id})$ is denoted by $\mathrm{HL}'(s)$. At a distance $\hat{h} \approx \hat{v}/(cos(5°) \cdot sin(5°))$ to the right of $\mathcal{G}(\mathrm{id})$ we place gadgets $\mathcal{G}_{\approx_h}(\sigma_j)$'s analogously to the $\mathcal{G}_{\approx_v}(\sigma_j)$'s. The fictitious points $p'^j_i$, $p_i^j$ used for the construction of the lines $\mathrm{SL}'(s)$, $\mathrm{SL}(s)$ are located at the right end of $\mathcal{G}(B)$ and ordered as $B$ when read from top to bottom. The difference in the construction of $\mathcal{G}(B)$ from the $B$-intervals (compared to $\mathcal{G}(A)$ from the $A$-intervals) is that the diagonal of blue points go from the top-left corner to the bottom-right corner (instead of bottom-left to top-right). Similarly to our previous definitions, we define $\mathrm{HL}' := \{\mathrm{HL}'(s) \mid s \in [tk]\}$ and $\mathrm{SL}' := \{\mathrm{SL}'(s) \mid s \in [tk]\}$. The choice of $\hat{h}$ makes the lines of $\mathrm{SL}'$ form a close to 5 degree angle with the $x$-axis and so *enter* $\mathcal{G}(B)$ relatively horizontal.

**Putting the pieces together.** We already hinted at how the different gadgets are combined together. We choose the different typical values so that: $kt \ll \hat{v} < \hat{h} \ll z$. For instance, $\hat{v} := 100((kt)^2 + 1)$ and $z := 100(\hat{h}^5 + 1)$. An important and somewhat hidden consequence of $z$ being much greater than $\hat{v}$ and $\hat{h}$ is that the bulk of the construction (say, all the gadgets

[3] By that, we mean that the lines are close to vertical for axes aligned with the encoding of track B.

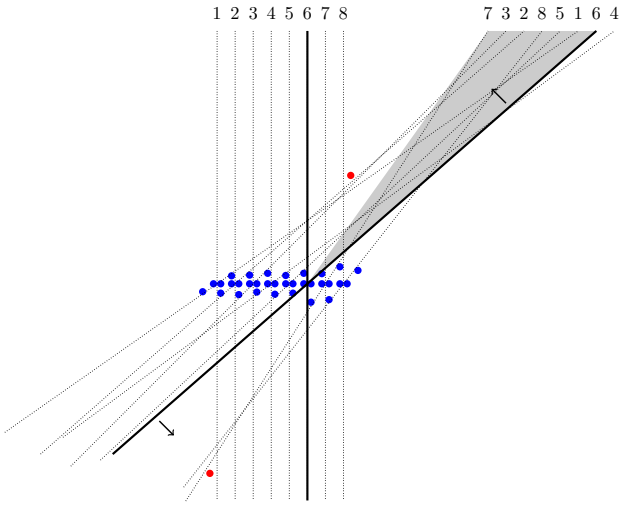



**Figure 4** Half-encoding of permutation $\sigma_j = 73285164$ of the $j$-th color class. Observe that the choice of the, say, sixth almost horizontal candidate line only forces to take the slanted line depicted in bold or a line having the same intersection with the almost horizontal line but a larger slope. For the sake of legibility, the angles between the vertical lines and the slanted lines are exaggerated.

which are not long alleys) occupies a tiny space in the top-left corner of $\Gamma$. We set the length $\ell$ of the long alleys to $100(k^2+1)$. Point $(x_0^A, y_0^A)$ corresponds to the bottom-left corner of the square in bold with a diagonal close to the overall top-left corner.

Slightly outside grid $\Gamma$ we place 14 pairs of long alleys (7 horizontal and 7 vertical) of width, say, $(kt)^{-10}$ to force the 14 lines in bold in Figure 5. Note that, on the figure, we do not explicitly represent those long alleys but only the lines they force. The purpose of those new long alleys is to separate groups of gadgets from each other. Going clockwise all around the grid $\Gamma$, we alternate red-blue and blue-red alleys so that two consecutive long alleys do not need a further separation. The even parity of those alleys make this alternation possible. Each one of the 64 faces that those 14 lines define is called a *super-cell*.

The four lines in bold surrounding $\mathcal{G}(B)$ are close (say, at distance $10t$) to the north, south, west, and east ends of that gadget. On the four super-cells adjacent to the super-cell containing $\mathcal{G}(B)$, shown in gray, we place $4k$ long alleys each of width $4t-4$, analogously to what was done for $\mathcal{G}(A)$, but slanted by a 5 degree angle (as the gadget $\mathcal{G}(B)$). As for track A, these alleys force, relatively to the orientation of $\mathcal{G}(B)$, one *close to horizontal* line and one *close to vertical line* per color class. The long alleys are placed just next to $\mathcal{G}(B)$ and are not crossed by any other candidate lines.

This finishes the construction. We ask for a separation of $\mathcal{R}$ and $\mathcal{B}$ with $6k+14$ lines. The correctness of the reduction is deferred to the long version of the paper. ◀

## 3 FPT Algorithm Parameterized by Size of Smaller Set

We present a simple FPT algorithm for Axis-Parallel Red-Blue Separation parameterized by $\min\{|\mathcal{R}|, |\mathcal{B}|\}$. In the following, w.l.o.g., we assume that $\mathcal{B}$ is the smaller set.

▶ **Theorem 2.** *An optimal solution of* Axis-Parallel Red-Blue Separation *can be computed in* $O(n \log n + n|\mathcal{B}|9^{|\mathcal{B}|})$ *time.*

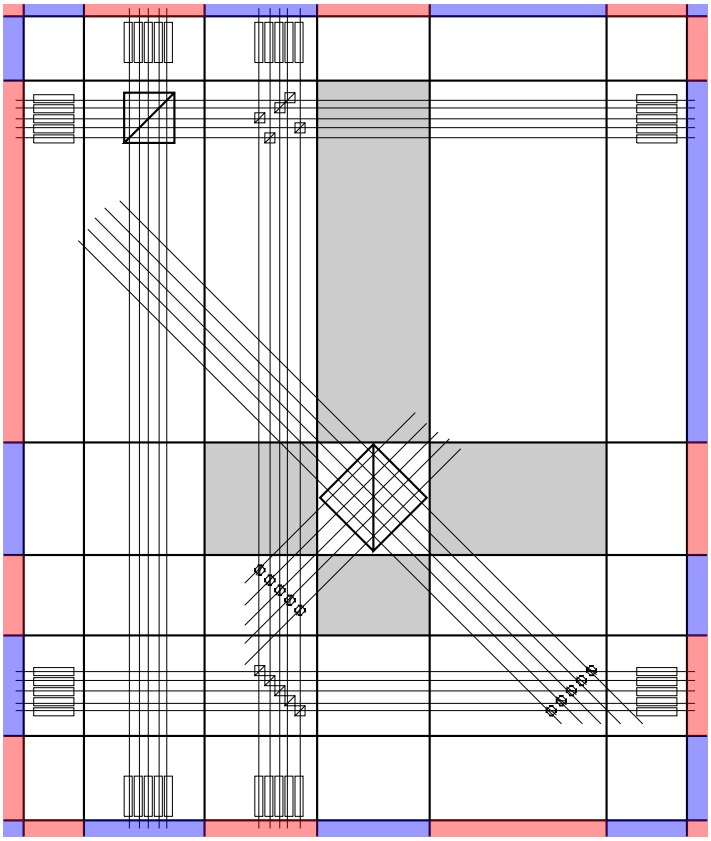
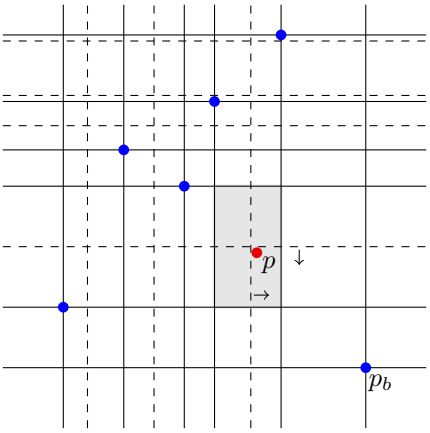


**Figure 5** The overall picture. The thin rectangles are long alleys, the bold large squares with a diagonal are the encoding of track A, in the top left corner, and track B, slanted by 45 degrees (for the sake of fitting the whole construction on one page; in reality the encoding of B is only inclined by 5 degrees). The smaller squares with a diagonal are simple interval gadgets and the small round gadgets are half-encodings of the permutations $\sigma_i$'s. The four super-cells filled with grey contain $4k$ long alleys slanted by 5 degrees. The (super-)cells filled with red and blue match their color, and are monochromatic once the 14 lines imposed by the outermost long alleys have been selected.

We first give a high-level description of the algorithm. It begins by subdividing the plane into at most $|\mathcal{B}| + 1$ vertical strips, each consisting of the area "between" two horizontally successive blue points, and at most $|\mathcal{B}| + 1$ horizontal strips, each consisting of the area "between" two vertically successive blue points (see Figure 6a). Since each strip can contain only red points in its interior, an optimal solution uses at most two lines inside a single strip (Lemma 5(a)). We can therefore guess (by exhaustive enumeration) the number of lines used in each strip in an optimal solution. This gives a running time of roughly $9^{|\mathcal{B}|}$. A second observation is that if an optimal solution uses two lines in a strip, these can be placed as far away from each other as possible (Lemma 5(b)). To complete the solution we must decide where to place the lines in strips that contain only one line of an optimal solution. We consider every pair of blue and red points whose separation may depend on the exact placement of these lines. The key idea is that the separation of two such points can be expressed as a 2-CNF constraint. If the upcoming formal exposition seems a bit more complicated than this informal idea, it is because we have to deal with points sharing the same x- or y-coordinates.




**(a)** The cell decomposition (solid lines), a guess of how $S$ intersects it (dashed lines), and an interesting cell (in gray) for a point $p_b$ (bottom-right corner). The red point $p$ cannot be in the south-east quadrant of this cell which translates to the 2-clause $y_p^2 \vee \neg x_p^4$. Indeed, it should be that the horizontal line of $S$ is below it or that the vertical line is to its right.


**(b)** Two consecutive red points in a horizontal strip $\mathcal{R}_h(i)$. If the corresponding line of $S$ is below $p$, then it is also below $p'$ which translates to $y_p^i \to y_{p'}^i$.


**(c)** Two consecutive red points in a vertical strip $\mathcal{R}_v(j)$. If the corresponding line of $S$ is to the left of $p$, then it is also to the left of $p'$ which translates to $x_p^i \to x_{p'}^i$.

**Figure 6** Illustration of the algorithm and the two kinds of clauses of the 2-SAT instance.

We now proceed to a formal description of our algorithm, beginning with some definitions. For a point $p \in \mathbb{R}^2$, let $p(x)$ and $p(y)$ be its $x$-coordinate and $y$-coordinate, respectively. Also, let $X, Y$ be the sets of $x$, $y$ coordinates of the points in $\mathcal{B}$. In order to ease presentation later on, with a slight terminology abuse, we add $-\infty, +\infty$ to both $X$ and $Y$. Let $X(i), Y(i)$ be the respective $i$-th elements of these sets in increasing order with $0 \leqslant i$, and let $k = |X| - 2$ and $l = |Y| - 2$; $k \leqslant |B|$ and $l \leqslant |B|$.

▶ **Definition 3.** The vertical strips are defined as $V_i = \{p \in \mathbb{R}^2 \mid X(i) \leqslant p(x) \leqslant X(i+1)\}$ for $i \in [0, k]$.

▶ **Definition 4.** The horizontal strips are defined as $H_i = \{p \in \mathbb{R}^2 \mid Y(i) \leqslant p(y) \leqslant Y(i+1)\}$ for $i \in [0, l]$.

The horizontal and vertical strips defined above essentially partition the plane into open monochromatic (red) or empty regions, while the boundaries of the strips may contain both red and blue points. As a result, we have the following properties of an optimal solution.

▶ **Lemma 5.** *(a) An optimal solution of* Axis-Parallel Red-Blue Separation *contains at most two lines in each horizontal or vertical strip. (b) In the case where a strip has two lines, these lines can be assumed to be placed in a way such that all red points in the interior of the strip lie between them.*

**Proof of Theorem 2.** We describe an FPT algorithm which first guesses how many lines an optimal solution uses in each strip and then produces a 2-SAT instance of size $O(|\mathcal{B}|n)$ in order to check if its guess is feasible. We assume that we have access to two lists containing the input points sorted lexicographically by their $(x, y)$ and $(y, x)$ coordinates.

Let $S$ be some optimal solution. We first guess how many lines of $S$ are in each horizontal and each vertical strip. Since, by Lemma 5, $S$ contains at most two lines per strip, and there

are $l+1 \leqslant |\mathcal{B}|+1$ horizontal strips and $k+1 \leqslant |\mathcal{B}|+1$ vertical strips, there are at most $3^{|\mathcal{B}|+1}$ possibilities to guess from for each direction thus, $O(9^{|\mathcal{B}|})$ in total.

In what follows, we assume that we have fixed how many lines of $S$ are in each strip. We give an algorithm deciding in polynomial time if such a specification gives a feasible solution. Since a specification fully determines the number of lines of a solution, the algorithm simply goes through all specifications and selects one with minimum cost among all feasible ones.

We produce a 2-SAT instance, which will be satisfiable if and only if a given specification is feasible. We first define the variables: for each horizontal strip $H_i$ that contains exactly one line from $S$ and for each red point $p \in H_i$, we define a variable $y^i_p$. Its informal meaning is "the line of $S$ in $H_i$ is below point $p$". When $p$ lies on the upper (lower) boundary of $H_i$, $y^i_p$ is set to true (false) by default. Similarly, for each vertical strip $V_j$ that contains exactly one line from $S$ and for each red point $p \in V_j$, we define a variable $x^j_p$. Its meaning is "the line of $S$ in $V_j$ is to the left of $p$". It is set to true (false) when $p$ lies on the right (left) boundary of $V_j$. We have constructed $O(n)$ variables (at most four for each point of $\mathcal{R}$).

Next, we construct 2-CNF clauses imposing the informal meaning described. For each strip $H_i$ that contains exactly one horizontal line from $S$ and each pair of red points $p, p' \in H_i$ that are consecutive in lexicographic $(y, x)$ order, we add the clause $(y^i_p \to y^i_{p'})$. We can skip pairs that have a point lying on the upper or lower boundary of $H_i$ as the corresponding variable has been already set to true or false respectively and the clause is satisfied; see the description in the previous paragraph. Similarly, for each strip $V_j$ that contains exactly one vertical line from $S$ and each pair of red points $p, p' \in V_j$ that are consecutive in lexicographic $(x, y)$ order, we add the clause $(x^j_p \to x^j_{p'})$; pairs that have a point lying on the left or right boundary of $V_j$ do not produce any clauses. Given any solution, we can construct from its lines an assignment following the informal meaning described above that satisfies all clauses added so far, while from any satisfying assignment we can find lines according to the informal meaning. We call the $O(n)$ clauses constructed so far the coherence part of our instance.

What remains is to add some further clauses to our instance to ensure also that the solution is feasible, that is, it separates all pairs of red and blue points.

Consider a cell $C_{ij} = H_i \cap V_j$, where $i \in [0, l]$ and $j \in [0, k]$. A red point $p \in C_{ij}$ is called *$C_{ij}$-separable* for a point $p_b \in \mathcal{B}$, if $p$ can be separated from $p_b$ by a vertical or horizontal line running through the interior of $C_{ij}$. We will sometimes call $p$ just separable when $C_{ij}$ and $p_b$ are obvious from the context. We say that $C_{ij}$ is *interesting* for a point $p_b \in \mathcal{B}$ if the following conditions hold: (i) $C_{ij}$ contains at least one red point that is $C_{ij}$-separable for $p_b$; (ii) at least one of $H_i$ or $V_j$ contains at most one horizontal or one vertical line from $S$ respectively; (iii) if $X(j+1) < p_b(x)$ or $p_b(x) < X(j)$, then there is no vertical line from $S$ in a strip between $p_b$ and $V_j$; and (iv) if $Y(i+1) < p_b(y)$ or $p_b(y) < Y(i)$, then there is no horizontal line from $S$ in a strip between $p_b$ and $H_i$. Note that even if $C_{ij}$ is interesting for $p_b$, it may contain a red point $p$ that is *already separated* from $p_b$ by a line going through $C_{ij}$: this happens exactly when $H_i$ or $V_j$ contains two horizontal or vertical lines from $S$ respectively and $p$ lies either in the interior of $C_{ij}$ or on its boundary but not on the same side of $H_i$ or $V_j$ as $p_b$. The motivation for these definitions is that the cells that are interesting for $p_b$ contain exactly the red points that need to be separated from $p_b$ by lines going through the cells and whose positions cannot be predetermined. We therefore have to add some clauses to express these constraints.

For each $p_b \in \mathcal{B}$ and each cell $C_{ij}$ that is interesting for $p_b$ we construct a clause for every red point $p \in C_{ij}$ that is separable and not already separated from $p_b$. Initially, the clause is empty. If the specification says that there is exactly one line from $S$ in $H_i$, we add to the clause a literal as follows: if $y(p_b) \geqslant Y(i+1)$, we add $\neg y^i_p$ (meaning that the horizontal line

is above $p$, and hence separates $p$ from $p_b$); if $y(p_b) \leqslant Y(i)$, we add $y_p^i$. Furthermore, if the specification says that there is exactly one line from $S$ in $V_j$, we add to the clause a literal as follows: if $x(p_b) \geqslant X(i+1)$, we add the literal $\neg x_p^j$; if $x(p_b) \leqslant X(i)$, we add $x_p^j$. Observe that this process produces clauses of size at most two. It may produce an empty clause, rendering the 2-SAT unsatisfiable, in the case where there is no line of $S$ in $H_i$ or $V_j$, but this is desirable since in this case no feasible solution matches the specification. Note that we have constructed $O(|\mathcal{B}||\mathcal{R}|)$ clauses in this way (at most four for each pair of a blue with a red point). Hence, the 2-SAT formula we have constructed has $O(n)$ variables and $O(|\mathcal{B}|n)$ clauses. Since 2-SAT can be solved in linear time, we obtain the promised running time.

To complete the proof we rely on the informal correspondence between assignments to the 2-SAT instance and Axis-Parallel Red-Blue Separation solutions. If there exists a solution that agrees with the guessed specification, this solution can easily be translated to an assignment that satisfies the coherence part of the 2-SAT formula. Furthermore, for any blue point $p_b$ and any separable and not already separated red point $p$ in a cell $C_{ij}$ that is interesting for $p_b$, the solution must be placing at least one line going through $C_{ij}$ in a way that separates $p_b$ from $p$ (this follows from the fact that the cell is interesting). Hence, the corresponding 2-SAT clauses are also satisfied. Conversely, given an assignment to the 2-SAT instance, we construct an Axis-Parallel Red-Blue Separation solution following the informal meaning of the variables. Note that for every blue point $p_b$, every red point is $C_{ij}$-separable for $p_b$ for at least one cell $C_{ij}$. For any cell $C_{ij}$ that is not interesting for $p_b$ and contains at least one separable point, we have that either all red points in the cell are separated from $p_b$ by lines outside the cell or all separable red points in the cell are separated from $p_b$ by the four lines running through the cell. If $C_{ij}$ is interesting for $p_b$, then all separable (and not already separated) red points in the cell are separated from $p_b$ because of the additional 2-SAT clauses we added in the second part of the construction. ◀

## 4 Open problems

The most intriguing open problem is settling the complexity of Axis-Parallel Red-Blue Separation w.r.t. the number of lines. We conjecture it to be FPT. Other problems include the complexity of Red-Blue Separation when the lines can have three different slopes and of Axis-Parallel Red-Blue Separation in 3-dimensions.

**Acknowledgements.** We thank Sergio Cabello and Christian Knauer for fruitful discussions.

## References

1 Édouard Bonnet and Tillmann Miltzow. Parameterized hardness of art gallery problems. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 19:1–19:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.19.

2 G. Calinescu, A. Dumitrescu, H.J. Karloff, and P. Wan. Separating points by axis-parallel lines. *Int. J. Comput. Geometry Appl.*, 15(6):575–590, 2005.

3 O. Devillers, F. Hurtado, M. Mora, and C. Seara. Separating several point sets in the plane. In *Proc. of the 13th Canad. Conf. Comput. Geom.*, pages 81–84, 2001.

4 U.M. Fayyad and K.B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. of 13th Int. Joint Conf. on Artificial Intelligence*, pages 1022–1029, 1993.

**5** R. Freimer, J.S.B. Mitchell, and C.D. Piatko. On the complexity of shattering using arrangements. TR 91-1197, Dept. Comput. Sci., Cornell Univ., NY, 1991.

**6** S. Har-Peled and M. Jones. On separating points by lines. arXiv:1706.02004v1, 2017.

**7** Ferran Hurtado, Mercè Mora, Pedro A. Ramos, and Carlos Seara. Separability by two lines and by nearly straight polygonal chains. *Discrete Applied Mathematics*, 144(1-2):110–122, 2004. doi:10.1016/j.dam.2003.11.014.

**8** J. Kujala and T. Elomaa. Improved algorithms for univariate discretization of continuous features. In *Proc. of the 11th PKDD*, volume 4702 of *LNCS*, pages 188–199, 2007.

**9** B. Lu, H. Du, X. Jia, Y. Xu, and B. Zhu. On a minimum linear classification problem. *J. of Global Optimization*, 35(1):103–109, 2006.

**10** N. Megiddo. On the complexity of polyhedral separability. *Discr. & Comput. Geom*, 3:325–337, 1988.

# Relativization and Interactive Proof Systems in Parameterized Complexity Theory*†

**Ralph Christian Bottesch**

**QuSoft, CWI, Amsterdam, The Netherlands**
bottesch@cwi.nl

## Abstract

We introduce some classical complexity-theoretic techniques to Parameterized Complexity. First, we study relativization for the machine models that were used by Chen, Flum, and Grohe (2005) to characterize a number of parameterized complexity classes. Here we obtain a new and non-trivial characterization of the **A**-Hierarchy in terms of oracle machines, and parameterize a famous result of Baker, Gill, and Solovay (1975), by proving that, relative to specific oracles, **FPT** and **A**[**1**] can either coincide or differ (a similar statement holds for **FPT** and **W**[**P**]). Second, we initiate the study of interactive proof systems in the parameterized setting, and show that every problem in the class **AW**[**SAT**] has a proof system with "short" interactions, in the sense that the number of rounds is upper-bounded in terms of the parameter value alone.

**1998 ACM Subject Classification** F.1.2 Modes of Computation, F.1.3 Complexity Measures and Classes

**Keywords and phrases** Parameterized complexity, Relativization, Interactive proofs

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.9

## 1 Introduction

In Parameterized Complexity Theory, the complexity of computational problems is measured not only in terms of the size of the input, $|x|$, but also in terms of a parameter $k$ which measures some additional structure of the input. The main advantage of this approach is that the class of problems which are considered computationally tractable can be expanded considerably by requiring that the running time of algorithms be polynomial only in $|x|$, while allowing some other dependence of the running time on the parameter value. Problems that can be solved by such algorithms are said to be *fixed-parameter tractable*. To this relaxed notion of computational tractability there corresponds a matching notion of intractability.

The complexity classes capturing parameterized intractability were originally defined as closures, under suitably defined parameterized reductions, of specific problems that were conjectured to not have fpt-algorithms (see [8], or the more recent [9]). This approach ensured that most of these "hard" classes contained an interesting or somewhat natural complete problem, and, in the case of **W**[1], produced a "web of reductions" similar to the one for **NP**-complete problems in classical complexity.

However, defining complexity classes only via reductions to specific problems means that the resulting classes may not have characterizations in terms of computing machines, or, indeed, any natural characterizations except the definition. This in turn can mean that many proof techniques from classical complexity are not usable in the parameterized setting,

* This work is supported by the ERC Consolidator Grant QPROGRESS 615307.
† A full version of the paper is available at https://arxiv.org/abs/1706.09391.


because they rely on different characterizations that do not apply to any one parameterized complexity class. To give an example, in the proof of **IP** = **PSPACE** ([15], see also [16]), both the definition of **PSPACE** in terms of space-bounded computation, and the characterization of this class in terms of alternating polynomial-time computation are used. In the parameterized world, this equivalence between space and alternating time seems to break down [6], and parameterized interactive proof systems do not appear to have been studied at all, so no similar theorem is known in this setting.

Surprisingly (given the way they were originally defined), many of the classes capturing parameterized intractability turned out to have characterizations in terms of computing machines: In three papers, Chen [5, 6, 7], Flum [5, 6, 7], and Grohe [6, 7] showed that certain kinds of nondeterministic random access machines (RAMs) exactly define some important parameterized classes:

- **W**[**P**] and **AW**[**P**] are characterized by RAMs that can nondeterministically [1] guess integers, but the number of guesses they can make throughout the computation is bounded by a computable function of the parameter value of the input instance. We refer to this as *parameter-bounded nondeterminism* (a term used similarly in [6]).
- The classes of the **A**-Hierarchy, as well as **AW**[$*$], are obtained by further restricting the (alternating) nondeterminism of the machines to *tail-nondeterminism*, meaning that the machines can only make nondeterministic guesses among the last $h(k)$ steps of a computation, where $h$ is a computable function and $k$ is the parameter.
- Finally, the classes of the **W**-Hierarchy are characterized by tail-nondeterministic machines which are not allowed to access the guessed integers directly (they can make nondeterministic decisions based on them, but not use them in arithmetic operations).

The main reason why the characterizations in [5, 6, 7] were given in terms of RAMs, rather than Turing machines (TMs), is that a TM may need to traverse the entire used portion of its tape in order to read a particular bit, so a tail-nondeterministic TM would not be able to make use of its entire memory during the nondeterministic phase of the computation. The classes **W**[**P**] and **AW**[**P**] also have characterizations in terms of TMs with restricted nondeterminism [6], but we consistently use random access machines throughout this work.

The machine characterizations of some of the above-mentioned classes can be rewritten in such a way that they strongly resemble definitions of some familiar classes from classical complexity. For example, **A**[1] can be defined as the class of parameterized problems that are decided by tail-nondeterministic RAMs in *fpt-time*, which at least formally looks like the definition of **NP**. Similarly, **W**[**P**] can also be defined in a way that is similar to **NP** (using parameter-bounded nondeterminism), the levels of the **A**-Hierarchy have characterizations that match the definitions of the $\Sigma$-levels of the Polynomial Hierarchy, and **AW**[**P**] and **AW**[$*$] both correspond to **AP** (the class of problems that are decidable in alternating polynomial-time). Given the similar definitions, it seems reasonable to expect that parameterized complexity classes also inherit some properties from their classical counterparts. However, replacing the machine model in a definition is a significant change, so it is by no means obvious which theorems will still hold for a parameterized version of a complexity class.

Our goal in this paper is to show that having machine-based characterizations of parameterized complexity classes opens up a largely unexplored, but possibly very fruitful, path toward understanding parameterized intractability. To that end we extend the work

[1] Throughout this paper, nondeterminism will mean alternating nondeterminism with a number of alternations that will be clear from the context. This should not cause any confusion, since simple nondeterminism is just 1-alternating nondeterminism.

of Chen, Flum, and Grohe [5, 6, 7] in two directions: relativization and interactive proofs. The key insight is that parameterized versions of these two concepts can be defined in such a way that some important classical theorems can be recovered in this setting. The proofs of our theorems follow along the same lines as their classical counterparts, with only some technical obstacles to be overcome, but it is a remarkable fact that parameterized versions of these proofs can be made to work at all: For example, it is not a priori clear whether parameterized oracle computation can be even in principle defined in a way that makes the **A**-Hierarchy have an oracle characterization that is similar to that of **PH**. We show, among other things, that this is indeed the case, and furthermore, that the restrictions that must be placed on the access to the oracle in order to obtain this result are quite natural (at least, in the context of the machine characterization of $\mathbf{A}[1]$ from [7]).

## 1.1 Our results

**Parameterized relativization.** Theorems involving oracles have been given before in Parameterized Complexity, but it is almost always Turing machines that are endowed with access to an oracle (see, for example, [13]). In order to relativize the hard parameterized complexity classes for which machine characterizations are known, we define oracle RAMs with the different forms of restricted nondeterminism mentioned above. It turns out that in order for oracle access and nondeterminism to interact in a useful way, both of these features must, roughly speaking, have the same restrictions (tail-nondeterministic machines should have tail-restricted oracle access, etc.)[2]. We show that these restrictions lead to a natural type of oracle access for each type of machine, by proving parameterized versions of two fundamental results from classical complexity, both for the tail-nondeterministic and the parameter-bounded version of nondeterministic RAMs.

First, we give a new characterization of the classes of the **A**-Hierarchy, in terms of oracle machines (resembling the oracle characterization of the levels of the Polynomial Hierarchy (see [3], Section 5.5)), by proving that

$$\forall t \geq 1 : \mathbf{A}[1]^{O_t} = \mathbf{A}[t+1],$$

but only for a *specific* oracle $O_t$ that is complete for $\mathbf{A}[t]$ (Theorem 13). We also explain why tail-nondeterminism appears to be too weak to allow for this theorem to be proved for an arbitrary $\mathbf{A}[t]$-complete problem. The situation is much better when the nondeterminism is only parameter-bounded, and we have (Theorem 16) that

$$\forall t \geq 1 : \mathbf{W}[\mathbf{P}]^{\mathbf{\Sigma}_t^{[P]}} = \mathbf{\Sigma}_{t+1}^{[P]},$$

where $\mathbf{\Sigma}_t^{[P]}$ $(t \geq 1)$ are the $\Sigma$-levels of the analogue of the Polynomial Hierarchy for the machine model with parameter-bounded nondeterminism (so $\mathbf{\Sigma}_1^{[P]} = \mathbf{W}[\mathbf{P}]$). We emphasize that both of these theorems seem to hold only if the oracle $\mathbf{A}[1]$- and $\mathbf{W}[\mathbf{P}]$-machines have exactly the right restrictions placed on their oracle access, and even then, tail-nondeterminism causes a number of non-trivial technical issues (see the proof of Theorem 13).

[2] Placing restrictions on the access to an oracle is a fairly common practice even in classical complexity. For example, the oracle tape of a **LOGSPACE**-machine is write-only, in order to allow the machine to make polynomial-sized queries while preventing it from using the tape for computations that avoid the space restriction. Another example can be found in [1], where, in order to prove that the statement **NEXP** $\subset$ **MIP** *algebrizes*, the authors restrict machines that run in exponential time so that they can only make oracle queries of polynomial size.

Second, we recover a parameterized version of a well-known oracle separation result of Baker, Gill, and Solovay [4], by showing (Theorem 14) that there exist parameterized oracles $A$ and $B$ such that

$$\mathbf{FPT}^A = \mathbf{A}[1]^A \quad \text{and} \quad \mathbf{FPT}^B \neq \mathbf{A}[1]^B.$$

It is worth noting that here the **FPT**-machine may be given completely unrestricted access to the oracle $B$, whereas the $\mathbf{A}[1]$-machine only has tail-restricted access (which is the most restricted form of oracle access we consider), so in some sense this separation is stronger than expected. A similar theorem holds when replacing $\mathbf{A}[1]$ with $\mathbf{W}[\mathbf{P}]$ (Theorem 18).

These results are, of course, only the first steps toward understanding relativization for parameterized complexity classes beyond **FPT**. To illustrate the importance of investigating relativization in this setting, let us briefly consider the long-standing open problem of proving a parameterized version of Toda's Theorem [17], which states that $\mathbf{PH} \subseteq \mathbf{P}^{\mathbf{PP}}$. It is not clear which parameterized classes would be involved in such a theorem, but, presumably, **P** would be replaced by **FPT**, which can easily be described in terms of Turing machines, so it should be possible to at least state the theorem without further considerations about the type of oracle access being used. Furthermore, it could be argued that since only the larger of the two classes in the theorem statement is obtained via relativization, placing no restrictions on the access to the oracle can only make the inclusion easier to prove. However, both Toda's original proof [17] and Fortnow's simplified version of it [12] make heavy use of relativized versions of classes such as **BPP** and **PH**, so following either one of these proofs would involve relativized versions of parameterized counterparts of such classes. Our Theorems 13 and 16 only deal with oracle access and alternating nondeterminism, but this already requires a careful balancing of the restrictions placed on both features. Toda's Theorem, on the other hand, involves an interplay between relativization, alternating nondeterminism, randomization, and counting complexity, so it seems unlikely that a parameterized version of it can be proved without a better understanding of parameterized relativization and its relation to other complexity-theoretic concepts.

**Interactive proof systems for parameterized complexity classes.** The levels of the **A**-Hierarchy were originally defined as fpt-closures of *model checking* problems, where a relational structure $\mathcal{A}$ and a first-order formula $\phi$ without free variables are given, and the task is to decide whether $\mathcal{A}$ satisfies $\phi$. In [7], model checking problems are used in a very interesting way in the proof of the machine characterization of the classes $\mathbf{A}[t]$: Specifically, a pair $(\mathcal{A}, \phi)$ is used to encode the computation of a tail-nondeterministic RAM, in a way that is strongly reminiscent of how the computation of a nondeterministic TM is encoded as a quantified Boolean formula in the proof of the Cook-Levin Theorem (see [3], Chap. 2). This suggests that by generalizing classical techniques that involve quantified Boolean formulas, it may be possible to apply them to parameterized complexity classes for which a model checking problem is complete. In Section 4 we continue this line of thought by generalizing *arithmetization* of quantified Boolean formulas (see [3], Section 8.3) to pairs of relational structures and first-order formulas.

We also initiate the study of interactive proof systems in this setting. Using generalized arithmetization, we show that all problems in $\mathbf{AW}[\mathbf{SAT}]$ have proof systems with a number of rounds depending only on the parameter value of the input instance (Theorem 19). The goal (which, unfortunately, is not achieved here) is to precisely characterize either $\mathbf{AW}[*]$ or $\mathbf{AW}[\mathbf{P}]$ in terms of IPs, as this would recover a parameterized version of the fact that $\mathbf{IP} = \mathbf{AP}$, even without a notion of space that corresponds to alternation in the parameterized setting. At the end of Section 4 we give a possible candidate for a characterization of $\mathbf{AW}[*]$.

Note that theorem proofs and other details can be found in the full version of paper (arXiv:1706.09391).

## 2 Preliminaries

We refer to [3] and to [11], respectively, for the necessary background in classical and Parameterized Complexity. By $\mathbb{N}$ we mean the set of non-negative integers, and by $\mathbb{N}^*$ the set of finite sequences of non-negative integers.

### 2.1 Random access machines and parameterized complexity classes

We give only a general overview of RAMs, and refer to Section 2.6 of [14] for the details. A random access machine is specified by its *program* (a finite sequence of instructions), which operates on an infinite sequence of *standard registers*, $r_0, r_1, \ldots$, that contain integers. Instructions access registers either directly, by referencing their numbers, or indirectly, by taking the number of a register to be the current content of another register (in other words, the machine can access $r_{r_i}$, $i \in \mathbb{N}$). We follow [6] in assuming that the registers store only non-negative integers. Except instructions which copy the contents of one register to another, a RAM also has conditional and unconditional jump instructions, as well as instructions which perform the operations addition, subtraction, and integer division by 2 (these suffice to efficiently perform all arithmetic operations on signed integers). The input of a RAM is a finite sequence of non-negative integers, each stored in a separate register, and we define the problems solved by such machines accordingly.

▶ **Definition 1.** A *parameterized problem* $Q$ is a subset of $\mathbb{N}^* \times \mathbb{N}$. When dealing with the problem of deciding whether $(x, k) \in \mathbb{N}^* \times \mathbb{N}$ is an element of $Q$, $(x, k)$ is referred to as an *instance*; the second element of such a pair is called the *parameter*.

▶ **Remark 2.** When an instance of a parameterized problem is given as input to a RAM, we assume that the parameter is given in unary encoding, meaning that if the parameter value is $k \in \mathbb{N}$, then $k$ registers, each containing the value 1, are used to encode the parameter value. The size of $x$, the main part of the input, is taken as the sum of the sizes of the binary encodings of the integers that make up $x$. A RAM can therefore efficiently convert between a reasonable encoding using integers, and any reasonable encoding using a finite alphabet.

▶ **Definition 3.** A random access machine $\mathbb{M}$ is *parameter-restricted* if there is a computable function $f$ and a polynomial function $p$, such that on any input $(x, k)$:

- $\mathbb{M}$ terminates after executing at most $f(k)p(|x|)$ instructions;
- throughout any computation, the registers contain only numbers that are $\leq f(k)p(|x|)$.

The above definition replaces the "polynomial-time" restriction on the running time in the classical setting, and is similar to the definition of "$\kappa$-restricted" in Chap. 6 of [11]. Note that the second condition is a bound on the numbers stored in the registers, not on the number of bits that would be needed for the binary encoding of these numbers.

The next definition is easily seen to be equivalent to the usual definition of **FPT** [11].

▶ **Definition 4.** We define **FPT** as the class of parameterized problems that are decidable by parameter-restricted (deterministic) RAMs.

An *alternating random access machine (ARAM)* is a RAM with additional *existential* and *universal guess instructions*, EXISTS and FORALL, both of which place a nondeterministically chosen integer from the interval $[0, r_0]$ into $r_0$ (the difference between the two

instructions is in how the acceptance of the input is defined). We may assume that the upper end of the range of each nondeterministic guess is the largest number that the machine can store in its registers, given the input, because the machine can first guess a number in the maximum range, and then trim the result by computing the remainder of a division by the size of the intended range. For ARAMs, the notions of computation (on an input), configuration, computation path, $t$-alternation, and acceptance/rejection of an input are defined in the standard way (see [11], section 8.1, pp. 168-170). Following [7], we mean by "$t$-alternating" that the first guess instruction is existential.

We give the definitions of some complexity classes in terms of nondeterministic RAMs. These are not the original definitions, but characterizations proved in [6] and [7].

▶ **Definition 5.** A parameterized problem $Q$ is in $\mathbf{AW}[\mathbf{P}]$ [in $\mathbf{W}[\mathbf{P}]$] if it is decided by an ARAM [a 1-alternating ARAM] $\mathbb{A}$ which, for some computable function $h$, on any input $(x, k)$, executes at most $h(k)$ nondeterministic instructions on any computation path.

▶ **Definition 6.** An ARAM $\mathbb{A}$ is *tail-nondeterministic* if there is a computable function $g$ such that, on any input $(x, k)$, $\mathbb{A}$ executes nondeterministic instructions only among the last $g(k)$ steps of any computation path. For every $t \geq 1$, $\mathbf{A}[t]$ denotes the class of parameterized problems that are decidable by parameter-restricted tail-nondeterministic $t$-alternating ARAMs. $\mathbf{AW}[*]$ denotes the class of parameterized problems that are decidable by parameter-restricted tail-nondeterministic ARAMs.

An *oracle (A)RAM* or *(A)RAM with access to an oracle* is a machine with an additional set of *oracle registers* that store non-negative integers, as well as instructions that copy the contents of $r_0$ to an arbitrary oracle register and vice-versa, and a QUERY instruction, which queries the oracle with the contents of the oracle registers, and causes the register $r_0$ to contain the values 1 or 0 (representing the oracle's answer). Note that we only work with oracles that decide parameterized problems, and that the parameter of a query instance must be encoded in unary (see Remark 2). Most previous results involving oracles in Parameterized Complexity place the following restriction on oracle machines. We will consider additional restrictions to oracle access in the next section.

▶ **Definition 7.** An oracle (A)RAM $\mathbb{A}$ has *balanced* access to an oracle if there is a computable function $g$ such that, on input $(x, k)$, any query $(y, k')$ made to the oracle, on any computation path, satisfies $k' \leq g(k)$.

## 2.2 Relational structures and first-order formulas

A *relational vocabulary* $\tau$ is a set of pairs of symbols and positive integers, called *relational symbols* and *arities*, respectively. A *relational structure* $\mathcal{A}$ *with vocabulary* $\tau$ is a set containing: a set $A$, called the *universe of* $\mathcal{A}$, and for each pair $(s, r) \in \tau$, a relation $R^s \subseteq A^r$. We only use relational structures with finite universes and finite vocabularies, so we assume that $A = \{0, \ldots, n\}$, $n \in \mathbb{N}$. A *first-order formula* $\phi$ *with vocabulary* $\tau$ is constructed in the same way as a quantified Boolean formula, except that the *atomic formulas* are not variables, but expressions of the form $x_1 = x_2$ or $R^s x_1 \ldots x_r$, where $x_1, \ldots, x_r$ are variables and $(s, r) \in \tau$.

Whenever a pair $(\mathcal{A}, \phi)$ is given, it is assumed implicitly that $\mathcal{A}$ and $\phi$ share the same relational vocabulary. We say that $\mathcal{A}$ *satisfies* $\phi$ if $\phi$ is true when all atomic formulas are evaluated based on the relations in $\mathcal{A}$ and all variables are taken as ranging over $A$.

We define some important classes of first-order formulas with relational vocabularies. For every $t \in \mathbb{N}$, let $\Sigma_t$ be the set of all first-order formulas of the form

$$\exists x_{1,1} \ldots \exists x_{1,k_1} \forall x_{2,1} \ldots \forall x_{2,k_2} \ldots\ldots Q x_{t,1} \ldots Q x_{t,k_t} : \psi(x_1, \ldots, x_t),$$

where $\psi(x_1, \ldots, x_t)$ is a quantifier-free formula ($Q$ means $\exists$ if $t$ is odd, $\forall$ if $t$ is even). For all $t, r \in \mathbb{N}$, let $\Sigma_t[r]$ be the set of all $\Sigma_t$-formulas with vocabularies in which all arities are $\leq r$. Finally, let PNF be the set of all first-order formulas in *prenex normal form*, meaning that they are of the form $Q_1 x_1 \ldots Q_t x_t : \psi(x_1, \ldots, x_t)$, where $\psi(x_1, \ldots, x_t)$ is a quantifier-free formula and $Q_1, \ldots, Q_t \in \{\exists, \forall\}$.

For certain classes of formulas $F$, the following parameterized *model checking* problems are complete for various important complexity classes.

| $p$-MC($F$) | |
|---|---|
| Input: | $(\mathcal{A}, \phi)$, where $\mathcal{A}$ is a relational structure, $\phi \in F$. |
| Parameter: | $\lvert\phi\rvert$. |
| Problem: | Decide whether $\mathcal{A}$ satisfies $\phi$. |

| $p$-*var*-MC($F$) | |
|---|---|
| Input: | $(\mathcal{A}, \phi)$, where $\mathcal{A}$ is a relational structure, $\phi \in F$. |
| Parameter: | The number of variables in $\phi$. |
| Problem: | Decide whether $\mathcal{A}$ satisfies $\phi$. |

▶ **Remark 8.** A relational structure can be represented by listing the elements of its universe, followed by the tuples in each relation. However, for a RAM to check whether some tuple $(a_1, \ldots, a_r)$ is an element of some $r$-ary relation $R^s$ may then take a number of steps that depends on $\|\mathcal{A}\| := |A| + |\tau| + \sum_{(s,r)\in\tau} |R^s| \cdot r$ (even if the elements of each relation are listed in lexicographic order, and binary search is used). To avoid this, we will assume, whenever $\mathcal{A}$ contains only relations of arity at most some fixed number $l$, that each $r$-ary relation ($r \leq l$) is stored as an $|A|^r$-size array of ones and zeroes, each number representing whether or not some element of $A^r$ is a member of the relation. Furthermore, we will assume that the location of every such array is stored in a look-up table. This way, checking whether $(a_1, \ldots, a_r) \in R^s$ only takes a *constant* number of operations for a RAM, at the cost of increasing the size of the representation of $\mathcal{A}$ in memory to $O(\text{poly}(\|\mathcal{A}\|))$ (since $l$ is constant). This also means that adding and removing elements requires only constant time.

▶ **Definition 9.** Let $Q$ and $Q'$ be parameterized problems. An algorithm $\mathbb{R}$ is an *fpt-reduction from $Q$ to $Q'$* if there exist computable functions $f$ and $g$, and a polynomial function $p$, such that for any instance $(x, k)$ of $Q$ we have a) $(y, k') := \mathbb{R}(x, k) \in Q'$ if and only if $(x, k) \in Q$; b) $\mathbb{R}$ runs in time $f(k)p(|x|)$; and c) $k' \leq h(k)$.

For any parameterized problem $Q$, we denote by $[Q]^{\text{fpt}}$ the set of parameterized problems that are $\leq^{\text{fpt}} Q$, meaning fpt-reducible to $Q$.

▶ **Fact 10** ([6, 10],[2])**.** For every $t \in \mathbb{N}$, $\mathbf{A}[t] = [p\text{-MC}(\Sigma_t)]^{\text{fpt}} = [p\text{-MC}(\Sigma_t[3])]^{\text{fpt}}$.
$\mathbf{AW}[\mathbf{SAT}] = [p\text{-}\mathit{var}\text{-MC}(\text{PNF})]^{\text{fpt}}$.

▶ **Remark 11.** In the proof of their machine-based characterization of $\mathbf{A}[t]$, Chen, Flum, and Grohe [7] show how the parameter-restricted computation of a $t$-alternating tail-nondeterministic RAM can be encoded as a pair $(\mathcal{A}, \phi)$. We refer the interested reader to [7] for the details, and recall only some facts about this reduction that we use here. Let $f(k)p(|x|)$ be an upper bound on the running time, the largest number of a register used, and the largest integer stored during the computation of the machine $\mathbb{A}$ on input $(x, k)$. The relational structure $\mathcal{A}$ has universe $\{0, \ldots, f(k)p(|x|)\}$ and contains relations representing the instructions of $\mathbb{A}$'s program and the contents of the accessed registers at the end of the deterministic part of the computation (a relation *Reg* is defined so that $(y, z) \in \mathit{Reg}$ if and

only if $r_y = z$ right before the first nondeterministic instruction is executed). All relations in $\mathcal{A}$ have arity $\leq 3$. The first-order formula $\phi$ has the same vocabulary as $\mathcal{A}$ and encodes the nondeterministic computation of $\mathbb{A}$ (the last $h(k)$ steps). The formula is constructed in such a way that changes to the contents of the registers are kept track of, and access to the contents of the registers at the start of the nondeterministic computation are encoded using the relation *Reg*. A close look at the construction in [7] reveals that part of it is oblivious to the input $x$, in the sense that computing the formula $\phi$ only requires knowledge of $k$, $\mathbb{A}$.

## 3 Parameterized relativization

The guiding principle in our approach to defining nondeterministic oracle RAMs will be that all of the special resources of a machine (nondeterminism, oracle queries, random guesses – everything beyond the basic deterministic operations) should be restricted in the same way, in order for these resources to interact well with each other.

▶ **Definition 12.** An oracle (A)RAM $\mathbb{A}$ has *parameter-bounded* access to an oracle if it has balanced access to the oracle, and there is a computable function $h$ such that, on input $(x, k)$, $\mathbb{A}$ makes at most $h(k)$ queries to the oracle on any computation path. $\mathbb{A}$ is said to have *tail-restricted* access to an oracle if it has balanced access to the oracle, and there is a computable function $h$ such that, on input $(x, k)$, $\mathbb{A}$ makes queries to the oracle only among the last $h(k)$ steps of any computation path.

Because we will use different kinds of oracle machines, and the exponent notation for the relativization of a complexity class is difficult to customize, we will also use the (older) parenthesis notation: If $C$ is a complexity class that is characterized by machines, we denote by $C(O)$ the class characterized by oracle machines of the same type as the ones characterizing $C$, with unrestricted access to the oracle $O$. Similarly, $C(O)_{bal}$ denotes the class defined by oracle machines with *balanced* access to the parameterized oracle, $C(O)_{para}$ denotes the class defined by oracle machines with *parameter-bounded* access to the oracle, and $C(O)_{tail}$ denotes the class defined by tail-nondeterministic oracle machines with the same restrictions as the machines that define $C$. The exponent notation is only used when the type of oracle access is the "natural" one for the type of machine being considered (so $\mathbf{A}[1]^O = \mathbf{A}[1](O)_{tail}$ and $\mathbf{W}[\mathbf{P}]^O = \mathbf{W}[\mathbf{P}](O)_{para}$). For **FPT** we always specify the type of oracle access.

**Relativization results for tail-nondeterministic random access machines.** We give an informal overview of the proof that $\mathbf{A}[1]^{p\text{-MC}(\Sigma_t[3])} = \mathbf{A}[t+1]$, to highlight the role played by the choice of the oracle and by the restrictions made to the tail-nondeterministic oracle machines (for a comparison with the proof that $\mathbf{NP}^{\Sigma_i \text{SAT}} = \mathbf{\Sigma}^P_{i+1}$, see [3], Section 5.5).

For the "$\supseteq$"-inclusion, we have that an $\mathbf{A}[1]$-machine with a $p$-MC$(\Sigma_t[3])$-oracle (which is complete for $\mathbf{A}[t]$) can first deterministically simulate the deterministic part of the computation of an $\mathbf{A}[t+1]$-machine on input $(x, k)$. The oracle $\mathbf{A}[1]$-machine then enters the nondeterministic phase of its computation, and uses its own nondeterministic guesses to simulate the first block of existential guesses of the simulated machine (until a universal instruction is encountered). The computation of the $\mathbf{A}[t+1]$-machine from this point onward (which starts with a universal guess instruction and has $\leq t-1$ alternations) can be encoded as an instance $((\mathcal{A}, \phi), |\phi|)$ of $p$-MC$(\Sigma_t[3])$ (see Remark 11), but the size of $\mathcal{A}$ depends on $|x|$. Therefore, $\mathcal{A}$ must (for the most part) be computed by the oracle $\mathbf{A}[1]$-machine and written to the oracle registers ahead of time, during the deterministic phase of the computation, with only the formula $\phi$ left to be computed during the nondeterministic phase. This is why it is necessary to allow tail-nondeterministic oracle machines access to their oracle registers throughout the entire computation.

For the reverse inclusion, we have that an $\mathbf{A}[t+1]$-machine can simulate an oracle $\mathbf{A}[1]$-machine on input $(x,k)$, by first simulating the deterministic part of the computation deterministically, and then using $(t+1)$-alternating nondeterminism to simulate both the oracle $\mathbf{A}[1]$-machine's existential guesses, as well as all of the $p$-MC$(\Sigma_t[3])$-queries (this is accomplished in the same way as in the classical proof). In order to evaluate the queried instances, however, the $\mathbf{A}[t+1]$-machine's computation must be in its nondeterministic phase, so it is essential that:

- the simulated oracle machine can not make queries outside of the last $h(k)$ steps of its computation, for some computable function $h$;
- the size of the formulas in the queried instances is $\leq g(k)$, for some computable function $g$ (balanced oracle access);
- the quantifier-free part of a formula can be evaluated efficiently (relational structures must be encoded in such a way that expressions involving relations can be evaluated by a RAM in time independent of the size of the relational structure; see Remark 8).

▶ **Theorem 13.** *For every $t \geq 1$, $\mathbf{A}[1]^{p\text{-MC}(\Sigma_t[3])} = \mathbf{A}[t+1]$.*

Since, for every $t \geq 1$, the problem used as an oracle in Theorem 13 is complete for $\mathbf{A}[t]$, it would be tempting to now state that $\mathbf{A}[1]^{\mathbf{A}[t]} = \mathbf{A}[t+1]$, because this would imply a "collapse theorem" for this hierarchy, namely that $\forall t \geq 1 : \mathbf{A}[t] = \mathbf{A}[t+1] \Rightarrow (\forall t' \geq t : \mathbf{A}[t] = \mathbf{A}[t'])$. Unfortunately, tail-nondeterminism appears to be too weak for such a collapse theorem to be proved in this fashion. In fact, it is not even certain whether $\mathbf{A}[1]^{\mathbf{FPT}} \subseteq \mathbf{A}[2]$: This is because an $\mathbf{A}[2]$-machine trying to simulate an $\mathbf{A}[1]$-machine that has oracle access to some non-trivial problem in $\mathbf{FPT}$, on some input $(x,k)$, may have to enter the nondeterministic phase of its computation before it even knows the instance to be queried (the simulated machine may write a large instance to its oracle registers, and then nondeterministically make some changes to it before querying the oracle). The size of this instance may depend on $|x|$, and although it can be decided in fpt-time, it may not be possible to decide it in time $h(k)$, for some computable function $h$, even with 2-alternating nondeterminism. Thus, the property of $p$-MC$(\Sigma_t[3])$ that, with the right encoding, an instance $((\mathcal{A},\phi),|\phi|)$ can be decided by a $t$-alternating tail-nondeterministic ARAM in time depending computably only on $|\phi|$, appears to have been crucial for our oracle characterization of the $\mathbf{A}$-Hierarchy.

The next theorem is the parameterized analogue of a famous classical result of Baker, Gill, and Solovay [4]. The construction of a parameterized oracle $B$ relative to which $\mathbf{FPT}$ and $\mathbf{A}[1]$ differ, is done via diagonalization and uses similar ideas as the classical proof in [4], but with two noteworthy differences:

First, when diagonalizing against all $\mathbf{FPT}$-machines, we can not computably list all such machines, because the $f(k)$-term in their running times can be any computable function. We must therefore proceed more carefully with the construction in order to obtain an oracle which is computable.

Second, when running each RAM on larger and larger inputs for an increasing number of steps while constructing the oracle, we are free to increase *both* the size of the main part of the input *and* the parameter value. Having this additional dimension of the input works in our favor, and allows us to "kill" the $f(k)$-term in the running time of any $\mathbf{FPT}$-machine by increasing $|x|$ so that $|x| > f(k)$, at which point we can treat $f(k)|x|^c$ as a polynomial in $|x|$.

▶ **Theorem 14.** *There exist parameterized oracles $A$ and $B$ such that*

$$\mathbf{FPT}(A)_{tail} = \mathbf{A}[1]^A \quad \textit{and} \quad \mathbf{FPT}(B)_{tail} \subsetneq \mathbf{A}[1]^B \ \textit{(and even } \mathbf{A}[1]^B \setminus \mathbf{FPT}(B) \neq \emptyset\textit{)}.$$

**Relativization results for RAMs with parameter-bounded nondeterminism.** For this machine model, we first need to define the analogue of the Polynomial Hierarchy.

▶ **Definition 15.** For each $t \geq 1$, let $\mathbf{\Sigma}_t^{[P]}$ be the class of parameterized problems that can be decided by a parameter-restricted $t$-alternating ARAM $\mathbb{A}$ such that, for some computable function $h$, on any input $(x, k)$, $\mathbb{A}$ executes at most $h(k)$ nondeterministic instructions on any computation path. Furthermore, we define $\mathbf{W}[\mathbf{P}]\mathbf{H} := \bigcup_{t=1}^{\infty} \mathbf{\Sigma}_t^{[P]}$.

Clearly, $\mathbf{W}[\mathbf{P}] = \mathbf{\Sigma}_1^{[P]} \subseteq \mathbf{W}[\mathbf{P}]\mathbf{H} \subseteq \mathbf{AW}[\mathbf{P}]$. For $t \geq 2$, $\mathbf{\Sigma}_t^{[P]}$-complete problems can be obtained by modifying known $\mathbf{W}[\mathbf{P}]$- or $\mathbf{AW}[\mathbf{P}]$-complete problems appropriately (see [6, 11]).

We turn to the oracle characterization of this hierarchy. Since a $\mathbf{W}[\mathbf{P}]$-machine can compute fpt-reductions at any point in the computation, the choice of the complete problem given as an oracle is no longer important. Now the proof of the theorem proceeds in the same way as the characterization of $\mathbf{PH}$ in terms of oracle machines (see [3], Section 5.5), but note that for the "$\subseteq$"-inclusion, the restrictions on the oracle access are nevertheless essential: balanced access ensures that the $\mathbf{\Sigma}_{t+1}^{[P]}$-machine can nondeterministically decide the instances queried by the oracle machine, and parameter-bounded access ensures that the number of queries made by the oracle machine is not too large for a $\mathbf{\Sigma}_{t+1}^{[P]}$-machine to simulate.

▶ **Theorem 16.** *For each $t \geq 1$, we have $\mathbf{W}[\mathbf{P}]^{\mathbf{\Sigma}_t^{[P]}} = \mathbf{\Sigma}_{t+1}^{[P]}$.*

▶ **Corollary 17.** *For any $t, u \geq 1$, if $\mathbf{\Sigma}_t^{[P]} = \mathbf{\Sigma}_{t+u}^{[P]}$, then $\mathbf{W}[\mathbf{P}]\mathbf{H} = \mathbf{\Sigma}_t^{[P]}$.*

Finally, we have the oracle separation result for this machine model, as in [4]:

▶ **Theorem 18.** *There exist parameterized oracles $A$ and $B$ such that*

$$\mathbf{FPT}(A)_{para} = \mathbf{W}[\mathbf{P}]^A \quad \textit{and} \quad \mathbf{FPT}(B)_{para} \subsetneq \mathbf{W}[\mathbf{P}]^B \ \textit{(and even } \mathbf{W}[\mathbf{P}]^B \setminus \mathbf{FPT}(B) \neq \emptyset\textit{)}.$$

For the proof, it suffices to use the same two oracles as in the proof of Theorem 14.

## 4 Interactive proof systems for parameterized complexity classes

A classical interactive proof system consists of a verifier and a prover who exchange messages in order for the verifier to decide whether a given input is a 'yes'-instance of a problem. The verifier is a probabilistic TM, meaning that he can guess random bits, but his computation throughout the entire interaction is time-bounded polynomially in terms of the size of the input instance (and therefore so is the length of the messages he can send or receive). The prover is computationally all-powerful, but he only sees the input and the messages sent by the verifier (not the verifier's random bits), and his goal is to convince the verifier to accept. A proof system is said to decide a problem $Q$ if every $x \in Q$ is accepted by the verifier with probability (over the verifier's random bits) $\geq 2/3$ for some prover, and every $x \notin Q$ is accepted by the verifier with probability $\leq 1/3$ for any prover (see [3], Chap. 8).

Here we make a slight change to this definition, in order to apply the concept to parameterized complexity classes, by letting the verifier be a probabilistic RAM (meaning that he can guess non-negative integers of bounded size in a single step), and allowing the messages between verifier and prover to be strings of non-negative integers of bounded size. This change does not affect the (classical) class $\mathbf{IP}$ (see Remark 2), but allows us to apply separate bounds to different aspects of the proof systems.

**Arithmetization of first-order formulas with relational vocabularies.** Before we can give interactive proof systems for parameterized complexity classes, we need to adapt the main technical tool used in such results, namely arithmetization. The main idea behind the original version of this technique is that a quantified Boolean formula can be replaced by a multivariate polynomial which coincides with the formula on all assignments of values to the non-quantified variables, if the Boolean truth values are identified with the elements of $GF(2)$ (in other words, the Boolean formula is encoded as a polynomial). Once this is accomplished, the polynomial can also be evaluated over some larger field, which is a key ingredient of the proof that **PSPACE** $\subseteq$ **IP** [15].

We wish to encode a pair $(\phi, \mathcal{A})$ as a polynomial, where $\phi$ is an FO formula, $\mathcal{A}$ is a relational structure with the same vocabulary as $\phi$, and the universe $A$ of $\mathcal{A}$ is $\{0, \dots, n\}$, $n \in \mathbb{N} \setminus \{0\}$. The main obstacle here is that the atomic formulas in $\phi$ are not Boolean variables, but relational expressions of the form $Rx_1 \dots x_l$, which evaluate to Boolean values whenever the variables are assigned values from $A$. We need a way to encode such a relational expression as a polynomial $P_R$ that takes the values 0 or 1 whenever $x_1, \dots, x_l \in A$, in accordance with the relation in $\mathcal{A}$ corresponding to $R$. To do this, we first choose a prime $q > n + 1$ and identify $A$ with a subset of $\{0, \dots, q-1\}$. We then take $P_R$ as the sum over all terms of the form $(1 - (X - a_1)^{q-1}) \cdot \ldots \cdot (1 - (X - a_l)^{q-1})$, where $(a_1, \dots, a_l)$ is in the relation corresponding to $R$ in $\mathcal{A}$, and argue via Fermat's Little Theorem that whenever $P_R$ is evaluated over values from $GF(q)$, at most one such term is 1, the rest being 0, and that $P_R$ therefore encodes the expression $Rx_1 \dots x_l$. (See the full version of the paper for details.)

With arithmetization generalized in this way, we are now in a position to construct an IP similar to the one used in [16] to show that **PSPACE** $\subseteq$ **IP**, and prove the following:

▶ **Theorem 19.** *For every problem $Q \in$ **AW**[**SAT**], there is an interactive proof system deciding $Q$ such that, for some computable functions $f$ and $h$, and a polynomial $p$, on any input $(x, k)$, the verifier runs in time $f(k)p(|x|)$, guesses at most $h(k)$ random numbers, and the interaction has at most $h(k)$ rounds.*

The IP in Theorem 19 has both the number of rounds and the number of random guesses made by the verifier bounded computably in terms of the parameter, but the length of the prover's messages and of the verifier's computations between rounds are "fpt-bounded". In order for an **AW**[$*$]-machine to simulate an interactive proof, it would presumably need to nondeterministically guess the prover's messages, as well as the random guesses made by the verifier, so the entire interaction would have to be simulated in the last $h(k)$ steps of the computation (due to tail-nondeterminism). In other words, the proof system would have to be such that the verifier only performs an fpt-bounded pre-computation, followed by an interaction that is entirely bounded in the parameter alone. We conjecture that the class of problems with such IPs, which we call $\mathbf{IP}^{tail}$, is precisely **AW**[$*$]. The evidence for this conjecture is that when the size of the FO formula is bounded in terms of the parameter, it seems that the IP from Theorem 19 can be improved so that at least the length of the prover's messages depends only on the parameter, by using only symbols for the polynomials representing the atomic relations, rather than expanding them into algebraic expressions. Getting the same bound for the verifier's computations between rounds is more challenging.

## 5 Conclusions

We have shown that, with some degree of effort, certain classical methods can be put to use in the parameterized setting, although some theorems only partially transfer over. The

fact that different aspects of the computation of a RAM are bounded differently, and that some computational resources can be tail-restricted, ensures that the machine-based theory of parameterized intractability is by no means just "complexity theory with RAMs".

One can now attempt to make some progress on the problem of separating matching levels of the **A**- and the **W**-Hierarchy by proving oracle separations when reasonable restrictions are placed on the oracle access of the respective machines. Another question is related to the fact that the implication $\mathbf{NP} \neq \mathbf{P} \Rightarrow \mathbf{A}[1] \neq \mathbf{FPT}$ is not known to hold: It would be interesting to show that this implication fails to hold relative to some oracle.

**Acknowledgments** The author is grateful to Yijia Chen and Sándor Kisfaludi-Bak for helpful discussions, to Martin Bottesch, Sándor Kisfaludi-Bak, and Ronald de Wolf for comments on a draft of this paper, and to two anonymous reviewers for their detailed suggestions on improvements to the version submitted to IPEC 2017.

## References

1 S. Aaronson and A. Wigderson. Algebrization: A new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1(1), 2009.
2 K.A. Abrahamson, R.G. Downey, and M.R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogs. *Annals of Pure and Applied Logic*, 73:235–276, 1995.
3 S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge, 2009.
4 T. Baker, J. Gill, and R. Solovay. Relativizations of the P=?NP question. *SIAM J. Comput.*, 4(4):431–442, 1975.
5 Y. Chen and J. Flum. Machine characterizations of the classes of the W-hierarchy. *Proceedings of the 17th International Workshop on Computer Science Logic, Lecture Notes in Computer Science*, 2803:114–127, 2003.
6 Y. Chen, J. Flum, and M. Grohe. Bounded nondeterminism and alternation in parameterized complexity theory. *Proceedings of the 18th IEEE Conference on Computational Complexity*, pages 13–29, 2003.
7 Y. Chen, J. Flum, and M. Grohe. Machine-based methods in parameterized complexity theory. *Theor. Comput. Sci.*, 339:167–199, 2005.
8 R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer, Berlin, 1999.
9 R.G. Downey and M.R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.
10 J. Flum and M. Grohe. Fixed-parameter tractability, definability, and model checking. *SIAM J. Comput.*, 31(1):113–145, 2001.
11 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, Berlin, 2006.
12 L. Fortnow. A simple proof of Toda's theorem. *Theory of Computing*, 5:135–140, 2009.
13 J.A. Montoya and M. M´uller. Parameterized random complexity. *Theory. Comput. Syst.*, 52:221–270, 2013.
14 C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
15 A. Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.
16 A. Shen. IP = PSPACE: simplified proof. *J. ACM*, 39(4):878–880, 1992.
17 S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

# How Much Does a Treedepth Modulator Help to Obtain Polynomial Kernels Beyond Sparse Graphs?*†

**Marin Bougeret[1] and Ignasi Sau[2]**

1 **LIRMM, Montpellier, France**
bougeret@lirmm.fr
2 **CNRS, LIRMM, Montpellier, France, and**
**Departamento de Matemática, UFC, Fortaleza, Brazil**
ignasi.sau@lirmm.fr

## Abstract

In the last years, kernelization with structural parameters has been an active area of research within the field of parameterized complexity. As a relevant example, Gajarskỳ *et al.* [ESA 2013] proved that every graph problem satisfying a property called finite integer index admits a linear kernel on graphs of bounded expansion and an almost linear kernel on nowhere dense graphs, parameterized by the size of a $c$-treedepth modulator, which is a vertex set whose removal results in a graph of treedepth at most $c$ for a fixed integer $c \geq 1$. The authors left as further research to investigate this parameter on general graphs, and in particular to find problems that, while admitting polynomial kernels on sparse graphs, behave differently on general graphs.

In this article we answer this question by finding two very natural such problems: we prove that Vertex Cover admits a polynomial kernel on general graphs for any integer $c \geq 1$, and that Dominating Set does *not* for any integer $c \geq 2$ even on degenerate graphs, unless NP $\subseteq$ coNP/poly. For the positive result, we build on the techniques of Jansen and Bodlaender [STACS 2011], and for the negative result we use a polynomial parameter transformation for $c \geq 3$ and an or-cross-composition for $c = 2$. As existing results imply that Dominating Set admits a polynomial kernel on degenerate graphs for $c = 1$, our result provides a dichotomy about the existence of polynomial problems for Dominating Set on degenerate graphs with this parameter.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

**Keywords and phrases** parameterized complexity, polynomial kernels, structural parameters, treedepth, treewidth, sparse graphs

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.10

## 1 Introduction

**Motivation.** There is a whole area of parameterized algorithms and kernelization investigating the *complexity ecology* (see for example [18]), where the objective is to consider a *structural* parameter measuring how "complex" is the input, rather than the size of the solution. For instance, parameterizing a problem by the treewidth of its input graph has been a great success for FPT algorithms, triggered by Courcelle's theorem [4] stating that

* This work has been supported by project DEMOGRAPH (ANR-16-CE40-0028).
† A full version of this article is permanently available at https://arxiv.org/abs/1609.08095.


any problem expressible in MSO logic is FPT parameterized by treewidth. However, the situation is not as good for kernelization, as many problems do not admit polynomial kernels when parameterized by treewidth unless NP $\subseteq$ coNP/poly [2].

Of fundamental importance within structural parameters are parameters measuring the so-called "distance from triviality" of the input graphs (a term that was first coined by Guo *et al.* [13]), like the size of a vertex cover (distance to an independent set) or of a feedback vertex set (distance to a forest). Unlike treewidth, these parameters may lead to both positive and negative results for polynomial kernelization. An elegant way to generalize these parameters is to consider a parameter allowing to *quantify* the triviality of the resulting instance, measured in terms of its treewidth. More precisely, for a positive integer $c$, a *$c$-treewidth modulator* of a graph $G$ is a set of vertices $X$ such that the treewidth of $G - X$ is at most $c$. Note that for $c = 0$ (resp. $c = 1$), a $c$-treewidth modulator corresponds to a vertex cover (resp. feedback vertex set).

Treewidth modulators have been extensively studied in kernelization, especially on classes of *sparse* graphs, where they have been at the heart of the recent developments of meta-theorems for obtaining linear and polynomial kernels on graphs on surfaces [3], minor-free graphs [8], and topological-minor-free graphs [12, 15], all based in a generic technique known as *protrusion replacement*. However, as observed in [11, 15], if one tries to move further in the families of sparse graphs by considering, for instance, graphs of bounded expansion, for several natural problems such as Treewidth-$t$ Vertex Deletion (minimizing the number of vertices to be removed to get a graph of treewidth at most $t$), parameterizing by a treewidth modulator is as hard as on general graphs.

This observation led Gajarskỳ *et al.* [11] to consider another type of modulators, namely *$c$-treedepth modulators* (defined analogously to $c$-treewidth modulators), where treedepth is a graph invariant – which we define in Section 2 – that plays a crucial structural role on graphs of bounded expansion and nowhere dense graphs [17]. Gajarskỳ *et al.* [11] proved that any graph problem satisfying a property called *finite integer index* admits a linear kernel on graphs of bounded expansion and an almost linear kernel on nowhere dense graphs when parameterized by the size of a $c$-treedepth modulator. Shortly afterwards this result was obtained, the authors asked [5] to investigate this parameter on *general graphs*, namely to find natural problems that admit and that do not admit polynomial kernels parameterized by the size of a $c$-treedepth modulator. More precisely, are there natural problems $\Pi_1$ and $\Pi_2$ fitting into the framework of [11] such that $\Pi_1/c$-tdmod admits a polynomial kernel on general graphs, but $\Pi_2/c$-tdmod does not? (As defined in Section 2, "/$c$-tdmod" means "parameterized by the size of a $c$-treedepth modulator".)

**Our results.** In this article we answer the above question by proving that Vertex Cover and Dominating Set are such problems $\Pi_1$ and $\Pi_2$, respectively. Let us now elaborate a bit more on our results, the techniques we use to prove them, and how do they compare to previous work in the area (see the preliminaries of Section 2 for any undefined terminology).

Note first that both VC/$c$-tdmod and DS/$c$-tdmod (where VC and DS stand for Vertex Cover and Dominating Set, respectively) are FPT on general graphs, as they are FPT by treewidth [4], which is a smaller parameter than $c$-tdmod, as for any graph $G$ and any integer $c \geq 0$, it holds that $\mathsf{tw}(G) \leq \mathsf{td}(G) - 1 \leq \text{c-tdmod}(G) + c - 1$. Thus, asking for polynomial kernels is a pertinent question.

In Section 3 we prove that VC/$c$-tdmod admits a polynomial kernel on general graphs. Our approach is based on the techniques introduced by Jansen and Bodlaender [14] to prove that VC/1-twmod (or equivalently, VC/FVS, where FVS stands for Feedback Vertex Set) admits a polynomial kernel. More precisely, we use three reduction rules inspired from

the rules given in [14], and we present a recursive algorithm that, starting from a $c$-treedepth modulator, constructs an appropriate $(c-1)$-treedepth modulator and calls itself inductively. The kernel obtained in this manner has $x^{2^{\mathcal{O}(c^2)}}$ vertices, where $x$ is the size of the $c$-treedepth modulator. This result completes the following panorama of structural parameterization for Vertex Cover, which has been a testbed for structural parameterizations in the last years:

- VC/1-twmod (or equivalently, VC/FVS) admits a polynomial kernel [14].
- VC/$c$-twmod for $c \geq 2$ does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly [6].
- VC/2-degmod (distance to a graph of maximum degree 2) and VC/$c$-CVD (distance to a disjoint collection of cliques of size at most $c$) admit a polynomial kernel [16]. Note that our result generalizes the latter kernel, as a disjoint collection of cliques of size at most $c$ is a particular case of a graph having treedepth at most $c$.
- VC/pfm (distance to a pseudoforest, a graph in which every connected component has at most one cycle) admits a polynomial kernel [9].

In Section 4 we turn to negative results for Dominating Set. We provide a characterization, according to the value of $c$, of the existence of polynomial kernels for DS/$c$-tdmod on degenerate graphs. Indeed, using the results of Philip *et al.* [19] it is almost immediate to prove that DS/1-tdmod (or equivalently, DS/VC) admits a polynomial kernel on degenerate graphs. For $c \geq 3$, we rule out the existence of polynomial kernels for DS/$c$-tdmod on 2-degenerate graphs by a simple polynomial parameter transformation from DS/1-tdmod on general graphs, which does not admit polynomial kernels unless NP $\subseteq$ coNP/poly [7]. The remaining case, namely DS/2-tdmod, turns out to be more interesting, and we rule out the existence of polynomial kernels on 4-degenerate graphs by providing an or-cross-composition from 3-Sat. This dichotomy for the existence of polynomial kernels for DS/$c$-tdmod on degenerate graphs is to be compared with the dichotomy for VC/$c$-twmod on general graphs discussed above [14, 6].

As mentioned before, it is commonly admitted that almost no natural problem admits a polynomial kernel parameterized by tw, or even with td. However, to the best of our knowledge the only published negative results are those in [2], which together with [10] imply that IS/tw and DS/tw do not admit a polynomial kernel unless NP $\subseteq$ coNP/poly. As this result only holds for general graphs, for the sake of completeness we complete it in the full version, by showing that a large majority of the problems considered in [11] having an almost linear kernel parameterized by $c$-tdmod on nowhere dense graphs do not admit polynomial kernels parameterized by td, even on planar graphs of bounded maximum degree.

Due to space limitations, the proofs of the results marked with '($\star$)' have been moved to the full version. We also refer the reader to the full version for the definition and acronyms of problems considered in the paper.

## 2 Preliminaries

We present here just some preliminaries about graphs. The basic definitions about parameterized complexity can be found in the full version.

Unless explicitly mentioned, all graphs considered here are simple and undirected. Given a graph $G = (V, E)$ and $X \subseteq V$, we denote $N_X(v) = N(v) \cap X$, where $N(v) = \{u \in V \mid \{u, v\} \in E\}$. We denote by $\alpha(G)$ the size of a maximum independent set of $G$. For any function $f$ defined on any induced subgraph of a given graph $G$, given a subset of vertices $V'$ of $G$, we denote $f(V') = f(G[V'])$ (for example, $\alpha(V') = \alpha(G[V'])$). For any integer $n$, we denote $[n] = \{i \in \mathbb{N} \mid 1 \leq i \leq n\}$.

For the following definitions related to treedepth, bounded expansion, and nowhere dense graph classes, we refer the reader to [17] for more details, and we only recall here some basic

notations and facts. The *treedepth* of a graph $G$ (denoted $\mathsf{td}(G)$) is the minimum height of a rooted forest $F$ (called a *treedepth decomposition*) such that $G$ is a subgraph of the closure of $F$, where the closure of a rooted tree is the graph obtained by adding an edge between any internal vertex and all its ancestors, and the height of a rooted tree is the number of vertices in a longest path from the root to a leaf. Let $c \geq 1$ be an integer. A *$c$-treedepth modulator* is a subset of vertices $X \subseteq V$ such that $\mathsf{td}(G[V \setminus X]) \leq c$, and we denote by $c\text{-}\mathsf{tdmod}(G)$ the size of a smallest $c$-treedepth modulator of $G$. A $c$-treewidth modulator is defined in the same way. Recall that as these parameters are greater than their associated measure (i.e., $\mathsf{tw}(G) \leq c\text{-}\mathsf{twmod}(G) + c$) the negative results for kernelization by treewidth and treedepth do not immediately apply, but the positive FPT results do.

Concerning graph classes, we recall that in the sparse graph hierarchy, graphs of bounded expansion (BE) and nowhere dense graphs (ND) are related to classic sparse families as follows (see [17] for the definitions): planar graphs $\subseteq$ minor-free graphs $\subseteq$ BE $\subseteq$ ND. Note also that the class of graphs of bounded degeneracy is a natural superclass of BE (intuitively, BE also requires the shallow minors to be degenerate), and is incomparable with ND.

## 3 A polynomial kernel for VC/$c$-tdmod on general graphs

In this section we prove that for any positive integer $c$, VC/$c$-tdmod admits a polynomial kernel on general graphs. Recall that this was only known for VC/1-tdmod and VC/2-tdmod, as for $c = 1$ this corresponds to the standard parameterization and we can use the linear kernel of [1], and for $c = 2$ we have 1-twmod $\leq$ 2-tdmod (as a 1-twmod corresponds to the distance to a forest, while 2-tdmod corresponds to the distance to a star forest), and thus we can use the polynomial kernel of [14] for VC/1-twmod. We also recall that we cannot expect to extend our result to VC/$c$-twmod for any $c \geq 2$ [6].

As VC/$c$-tdmod and IS/$c$-tdmod are clearly equivalent for this parameterization, we provide the result for IS/$c$-tdmod. More specifically, in Subsection 3.1 we provide a polynomial kernel for a-$c$-tdmod-IS, an annotated version of our problem on hypergraphs defined below, and in Subsection 3.2 we derive a polynomial kernel for IS/$c$-tdmod.

### 3.1 A polynomial kernel for a-$c$-tdmod-IS/$(|X| + |\mathcal{H}|)$

Working with hypergraphs is useful because we will use a reduction rule identifying a subset $X'$ of the modulator that cannot be entirely contained in a solution; this will be modeled by adding a hyperedge on the set $X'$.

| Annotated $c$-treedepth modulator Independent Set (a-$c$-tdmod-IS) | |
|---|---|
| **Instance:** | $(G, X, k)$ where<br>• $G = (V, E, \mathcal{H})$ is a hypergraph structured as follows: $V = X \uplus R$, $E = E_{X,R} \uplus E_{R,R}$ is a set of edges where edges in $E_{A,B}$ have one endpoint in $A$ and the other in $B$, and $\mathcal{H} \subseteq 2^X$ is a set of hyperedges where each $H \in \mathcal{H}$ is entirely contained in $X$.<br>• $X$ is a $c$-treedepth modulator (as $G[V \setminus X]$ is no longer a hypergraph, its treedepth is correctly defined and we have $\mathsf{td}(V \setminus X) \leq c$).<br>• $k$ is a positive integer. |
| **Question:** | Decide whether $\alpha(G) \geq k$ (where an independent set in a hypergraph is a subset of vertices that does not contain any hyperedge, corresponding here to a subset $S \subseteq V$ such that for every $h \in E \cup H$, $h \not\subseteq S$). |

Throughout this subsection $I = (G, X, k)$ denotes the input of a-$c$-tdmod-IS with $G = (V, E, \mathcal{H})$ and $V = X \uplus R$. Note that $G[X]$ is a hypergraph and that $G[R]$ is a graph, and that the parameter we consider here is $|X| + |\mathcal{H}|$. For any $X' \subseteq X$ and $R' \subseteq R$, observe that the notation $N_{R'}(X')$ is not ambiguous and denotes $\{v \in R' \mid \exists x \in X' \text{ with } \{x, v\} \in E\}$.

We use the following definition that was introduced in [14] for VC/1-twmod.

▶ **Definition 1 (**[14]**).** Given $X' \subseteq X$ and $R' \subseteq R$, let $\mathsf{conf}_{R'}(X') = \alpha(R') - \alpha(R' \setminus N_{R'}(X'))$ be the *conflicts* induced by $X'$ on $R'$.

Intuitively, $\mathsf{conf}_{R'}(X')$ measures the loss in the size of a maximum independent set of $R'$ due to $X'$. We extend the previous definition in the following way: for any $R' \subseteq R$ and any $Y' \subseteq R'$, let $\mathsf{conf}_{R'}(Y') = \alpha(R') - \alpha(R' \setminus Y')$. We can see that $\mathsf{conf}_{R'}(Y') = 0$ is equivalent to the existence of an independent set $S^* \subseteq R'$ such that $|S^*| = \alpha(R')$ and $S^* \cap Y' = \emptyset$.

▶ **Lemma 2.** *Let $R' \subseteq R$ be a connected component of $R$ and let $Y' \subseteq R'$. If $\mathsf{conf}_{R'}(Y') > 0$, there exists $\bar{Y}' \subseteq Y'$ such that $\mathsf{conf}_{R'}(\bar{Y}') > 0$ and $|\bar{Y}'| \leq f(c)$ with $f(c) = 2^c$.*

**Proof.** As it holds that $\mathsf{td}(R') \leq c$, let us consider a treedepth decomposition of $R'$ with root $r$ and $t \geq 1$ subtrees, where $A_i$, $i \in [t]$ is the vertex set of subtree $i$. We can partition $Y' = \bigcup_{i \in [t+1]} Y'_i$ with $Y'_i \subseteq A_i$ for $i \in [t]$, $Y'_{t+1} \subseteq \{r\}$, where the $Y'_i$'s are possibly empty. We will prove the lemma by induction on $c$. Observe that $\sum_{i \in [t]} \alpha(A_i) \leq \alpha(R') \leq 1 + \sum_{i \in [t]} \alpha(A_i)$, and thus we distinguish two cases according to the value of $\alpha(R')$.

**Case 1.** $\alpha(R') = 1 + \sum_{i \in [t]} \alpha(A_i)$. In this case any maximum independent set $S^*$ of $R'$ contains $r$. Hence for every $i \in [t]$, $S^* \cap A_i$ is a maximum independent set in $A_i \setminus N_{A_i}(r)$, and thus $\alpha(A_i \setminus N_{A_i}(r)) = \alpha(A_i)$. Indeed, if we had $\alpha(A_i \setminus N_{A_i}(r)) < \alpha(A_i)$ for some $i$, then $|S^*|$ would be strictly smaller than $1 + \sum_{i \in [t]} \alpha(A_i)$.

If $r \in Y'$ (i.e., if $Y'_{t+1} \neq \emptyset$) then we can take $\bar{Y}' = \{r\}$ (as any optimal solution of $R'$ must contain $r$ we get $\alpha(R' \setminus \{r\}) < \alpha(R')$, and $|\bar{Y}'| = 1 \leq 2^c$), and thus we suppose henceforth that $Y'_{t+1} = \emptyset$.

We claim that there exists $i_0 \in [t]$ such that $\mathsf{conf}_{A_{i_0} \setminus N_{A_{i_0}}(r)}(Y'_{i_0}) > 0$. Indeed, otherwise we could define for any $i \in [t]$ an independent set $S_i \subseteq A_i \setminus N_{A_i}(r)$ with $|S_i| = \alpha(A_i \setminus N_{A_i}(r)) = \alpha(A_i)$ and $S_i \cap Y'_i = \emptyset$. Thus, $S^* = \{r\} \cup_{i \in [t]} S_i$ would be an independent set of size $\alpha(R')$, and as $Y'_{t+1} = \emptyset$ we would have $S^* \cap Y' = \emptyset$, a contradiction to the hypothesis that $\mathsf{conf}_{R'}(Y') > 0$. Thus, there exists $i_0 \in [t]$ such that $\mathsf{conf}_{A_{i_0} \setminus N_{A_{i_0}}(r)}(Y'_{i_0}) > 0$, and as $\mathsf{td}(A_{i_0} \setminus N_{A_{i_0}}(r)) < c$, by induction hypothesis there exists $\bar{Y}'_{i_0} \subseteq Y'_{i_0}$ such that $\mathsf{conf}_{A_{i_0} \setminus N_{A_{i_0}}(r)}(\bar{Y}'_{i_0}) > 0$ and $|\bar{Y}'_{i_0}| \leq 2^{c-1}$. Let us verify that $\bar{Y}' = \bar{Y}'_{i_0}$ satisfies $\mathsf{conf}_{R'}(\bar{Y}') > 0$. Let $S^*$ be an independent set of $R'$ with $S^* \cap \bar{Y}' = \emptyset$. If $r \notin S^*$ then clearly $|S^*| < \alpha(R')$. Otherwise, $|S^*| = (\sum_{i \in [t]} |S^* \cap (A_i \setminus N_{A_i}(r))|) + 1 \leq \alpha(A_{i_0} \setminus N_{A_{i_0}}(r)) - 1 + (\sum_{i \in [t], i \neq i_0} \alpha(A_i \setminus N_{A_i}(r))) + 1 < \alpha(R')$.

**Case 2.** $\alpha(R') = \sum_{i \in [t]} \alpha(A_i)$. In this case there exists $i_0 \in [t]$ such that $\mathsf{conf}_{A_{i_0}}(Y'_{i_0}) > 0$. Indeed, otherwise we could define for any $i \in [t]$ an independent set $S_i \subseteq A_i$ with $|S_i| = \alpha(A_i)$ and $S_i \cap Y'_i = \emptyset$, and the existence of $S^* = \cup_{i \in [t]} S_i$ would be a contradiction to the hypothesis that $\mathsf{conf}_{R'}(Y') > 0$. Thus, by the induction hypothesis there exists $\bar{Y}'_{i_0} \subseteq Y'_{i_0}$ such that $\mathsf{conf}_{A_{i_0}}(\bar{Y}'_{i_0}) > 0$ and $|\bar{Y}'_{i_0}| \leq 2^{c-1}$.

If $r \in Y'$ (i.e., if $Y'_{t+1} \neq \emptyset$) then we can take $\bar{Y}' = \bar{Y}'_{i_0} \cup \{r\}$. Let us verify that $\mathsf{conf}_{R'}(\bar{Y}') > 0$. Let $S^*$ be an independent set of $R'$ with $S^* \cap \bar{Y}' = \emptyset$. As $S^*$ cannot contain $r$ we have $|S^*| = \sum_{i \in [t]} |S^* \cap A_i| < \alpha(A_{i_0}) + \sum_{i \in [t], i \neq i_0} |S^* \cap A_i| = \alpha(R')$. Thus, we suppose from now on property $\mathbf{p_1} : Y'_{t+1} = \emptyset$.

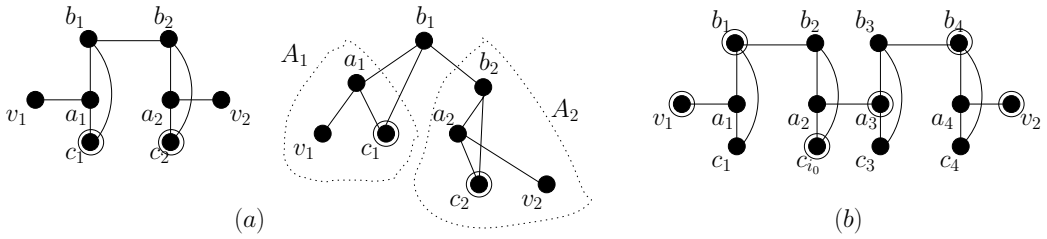



**Figure 1** (*a*) Example of a graph $G[R']$ (left) with an associated treedepth decomposition (right) as used in Lemma 2, with $Y' = \{c_1, c_2\}$. This case corresponds to one of the subcases treated in Case 2 of Lemma 2, as $\alpha(R') = \alpha(A_1) + \alpha(A_2) = 4$, $\mathsf{conf}_{A_1}(Y'_1) > 0$, $\mathsf{conf}_{A_2}(Y'_2) = 0$. Moreover, $\mathbf{p_2}$ and $\mathbf{p'_2}$ are true, while $\mathbf{p_3}$ is false (but $\mathbf{p'_3}$ is true). (*b*) Example for $t = 2$ of the construction of Lemma 3, where the circled vertices belong to $S$.

Note that in this case (when $\mathbf{p_1}$ is true) we cannot simply set $\bar{Y}' = Y'_{i_0}$, as shown in the example depicted in Figure 1. Indeed, in this example we would have $\bar{Y}' = Y'_{i_0} = \{c_1\}$, however $\mathsf{conf}_{R'}(\{c_1\}) = 0$ as $S^* = \{b_1, v_1, c_2, v_2\}$ verifies $|S^*| = \alpha(R')$ and $S^* \cap \{c_1\} = \emptyset$.

*Properties related to $\alpha$.* We claim that we can assume property $\mathbf{p_2}$ : for every $i \neq i_0$, $\alpha(A_i \setminus N_{A_i}(r)) = \alpha(A_i)$. Indeed, if $\mathbf{p_2}$ is not true, then there exists $i_1 \neq i_0$, $i_1 \in [t]$ such that $\alpha(A_{i_1} \setminus N_{A_{i_1}}(r)) < \alpha(A_{i_1})$, and we set $\bar{Y}' = Y'_{i_0}$. Let $S^*$ be an independent set of $R'$ with $S^* \cap \bar{Y}' = \emptyset$. If $r \notin S^*$ then as previously $|S^*| < \alpha(R')$, otherwise we get $|S^*| \leq \alpha(A_{i_0}) - 1 + \alpha(A_{i_1}) - 1 + (\sum_{i\in[t], i\neq i_0, i\neq i_1} \alpha(A_i)) + 1 < \alpha(R')$. Thus, we now assume $\mathbf{p_2}$.

Let us now prove the following property $\mathbf{p'_2} : \alpha(A_{i_0} \cup \{r\}) = \alpha(A_{i_0})$. By contradiction, suppose that there exists an independent set $S^*_1$ of $A_{i_0} \cup \{r\}$ containing $r$ such that $|S^*_1| = \alpha(A_{i_0}) + 1$. According to $\mathbf{p_2}$, for every $i \neq i_0$ there exists an independent set $S_i$ of $A_i \setminus N_{A_i}(r)$ of size $\alpha(A_i)$, and thus $\alpha(R') > \sum_{i\in[t]} \alpha(A_i)$, a contradiction. Thus, we now assume $\mathbf{p'_2}$.

*Properties related to $\mathsf{conf}_{A_i}(Y'_i)$.* Let us prove than we can assume the following property $\mathbf{p_3}$ : for every $i \neq i_0, \mathsf{conf}_{A_i \setminus N_{A_i}(r)}(Y'_i) = 0$. Indeed, if $\mathbf{p_3}$ is not true we can get the desired result as follows. Let $i_1 \neq i_0$, $i_1 \in [t]$ such that $\mathsf{conf}_{A_{i_1} \setminus N_{A_{i_1}}(r)}(Y'_{i_1}) > 0$. We use the same arguments as in the previous paragraph and define $\bar{Y}' = Y'_{i_0} \cup Y'_{i_1}$. Note that $|\bar{Y}'| \leq |Y'_{i_0}| + |Y'_{i_1}| \leq 2^c$. Using the same notation, if $r \notin S^*$ then $|S^*| = (\sum_{i\in[t]} |S^* \cap A_i|) \leq \alpha(A_{i_0}) - 1 + (\sum_{i\in[t], i\neq i_0} \alpha(A_i)) < \alpha(R')$, and otherwise $|S^*| = (\sum_{i\in[t]} |S^* \cap (A_i \setminus N_{A_i}(r))|) + 1 \leq \alpha(A_{i_0}) - 1 + \alpha(A_{i_1}) - 1 + (\sum_{i\in[t], i\neq i_0, i\neq i_1} \alpha(A_i)) + 1 < \alpha(R')$. Thus, we now assume $\mathbf{p_3}$. Note that $\mathbf{p_2}$ and $\mathbf{p_3}$ imply property $\mathbf{p'_3}$ : for every $i \neq i_0, \mathsf{conf}_{A_i}(Y'_i) = 0$.

**Case 2a.** $\nexists S^*$ maximum independent set of $R'$ such that $r \in S^*$. In this case, we set $\bar{Y}' = Y'_{i_0}$. Let us prove that $\mathsf{conf}_{R'}(\bar{Y}') > 0$. Let $S^*$ be a maximum independent set of $R'$ with $S^* \cap \bar{Y}' = \emptyset$. As $r \notin S^*$, we get $|S^*| = \sum_{i\in[t]} |S^* \cap A_i| \leq \alpha(A_{i_0}) - 1 + \sum_{i\in[t], i\neq i_0} \alpha(A_i) < \alpha(R')$.

**Case 2b.** $\exists S^*$ maximum independent set of $R'$ such that $r \in S^*$. This implies that $\alpha(A_{i_0} \setminus N_{A_{i_0}}(r)) = \alpha(A_{i_0}) - 1$. Let us prove that $\mathsf{conf}_{A_{i_0} \setminus N_{A_{i_0}}(r)}(Y'_{i_0}) > 0$. If it was not the case, there would exist an independent set $S^*_{i_0}$ of $A_{i_0} \setminus N_{A_{i_0}}(r)$ of size $\alpha(A_{i_0} \setminus N_{A_{i_0}}(r)) = \alpha(A_{i_0}) - 1$ such that $S^*_{i_0} \cap Y'_{i_0} = \emptyset$. By $\mathbf{p_3}$, there would exist, for every $i \neq i_0$, an independent set $S^*_i$ of $A_i \setminus N_{A_i}(r)$ of size $\alpha(A_i \setminus N_{A_i}(r)) = \alpha(A_i)$ (by $\mathbf{p_2}$) such that $S^*_i \cap Y'_i = \emptyset$. Thus, $S^* = \{r\} \cup (\bigcup_{i\in[t]} S^*_i)$ would be an independent set of size $\alpha(R')$ such that $S^* \cap Y' = \emptyset$ (recall that by $\mathbf{p_1}$, $r \notin Y'$), a contradiction. Thus, we know that both $\mathsf{conf}_{A_{i_0} \setminus N_{A_{i_0}}(r)}(Y'_{i_0}) > 0$

and $\mathsf{conf}_{A_{i_0}}(Y'_{i_0}) > 0$ (which was established at the beginning of Case 2). Using twice the induction hypothesis we get that there exists $\bar{Y'_{i_0}}^1 \subseteq Y'_{i_0}$ such that $\mathsf{conf}_{A_{i_0} \setminus N_{A_{i_0}}(r)}(\bar{Y'_{i_0}}^1) > 0$ and there exists $\bar{Y'_{i_0}}^2 \subseteq Y'_{i_0}$ such that $\mathsf{conf}_{A_{i_0}}(\bar{Y'_{i_0}}^2) > 0$, with both $|\bar{Y'_{i_0}}^1|$ and $|\bar{Y'_{i_0}}^2|$ bounded by $2^{c-1}$. Thus, we set $\bar{Y'} = \bar{Y'_{i_0}}^1 \cup \bar{Y'_{i_0}}^2$. Let us verify that $\mathsf{conf}_{R'}(\bar{Y'}) > 0$. Let $S^*$ be an independent set of $R'$ with $S^* \cap \bar{Y'} = \emptyset$. If $r \in S^*$, then $|S^*| = \sum_{i \in [t]} |S^* \cap (A_i \setminus N_{A_i}(r))| + 1 = \alpha(A_{i_0} \setminus N_{A_{i_0}}(r)) - 1 + \sum_{i \in [t], i \neq i_0} \alpha(A_i) + 1 = \alpha(A_{i_0}) - 2 + \sum_{i \in [t], i \neq i_0} \alpha(A_i) + 1 < \alpha(R')$. Otherwise, $|S^*| = \sum_{i \in [t]} |S^* \cap A_i| = \alpha(A_{i_0}) - 1 + \sum_{i \in [t], i \neq i_0} \alpha(A_i) < \alpha(R')$. ◀

A first lower bound on the function $f$ of Lemma 2 can be obtained by considering a clique $R'$ on $c$ vertices (hence, with $\mathsf{td}(R') = c$) and $Y' = R'$, as any $\bar{Y'} \subsetneq Y'$ satisfies $\mathsf{conf}_{R'}(\bar{Y'}) = 0$. However, as shown in Lemma 3 below, we can even obtain an exponential lower bound, showing that the function $f(c) = 2^c$ of Lemma 2 is almost tight.

▶ **Lemma 3** (⋆). *There exists a constant $\lambda$ such that for any $c \geq \lambda$ there exists a graph $G = (R, E)$ and $Y \subseteq R$ such that $\mathsf{td}(G) = c$, $|Y| \geq 2^{c-3}$, $\mathsf{conf}_R(Y) > 0$, and for every $\bar{Y} \subsetneq Y$, $\mathsf{conf}_R(\bar{Y}) = 0$.*

▶ Remark. Lemma 2 was proven in [14] when $R'$ is a forest and with $|\bar{Y'}| \leq 2$. Even if we already know that IS/2-twmod does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly [6], it remains interesting to observe that, in particular, this lemma becomes false for 2-twmod, as the graph of Lemma 3 has treewidth 2. This points out one crucial difference between $c$-treewidth and $c$-treedepth modulators.

Let us now start the description of the kernel for a-$c$-tdmod-IS/$(|X| + |\mathcal{H}|)$. Given an input $(G, X, k)$ of a-$c$-tdmod-IS, we define the following three rules. Note that these rules and definitions (and the associated safeness proofs) correspond to Rules 1, 2, and 3 of [14], except that we now bound the sizes of the subsets by a function $f(c)$ instead of by 2.

▶ **Definition 4.** Given an input $(G, X, k)$ of a-$c$-tdmod-IS (with $\mathsf{td}(G[R]) \leq c$ where $R = V \setminus X$), the *chunks* of the input are defined by $\mathcal{X} = \{X' \subseteq X \mid$ there is no $H \in \mathcal{H}$ such that $H \subseteq X'$, and $0 < |X'| \leq f(c)\}$, where $f(c) = 2^c$.

Intuitively, the chunks correspond to all possible small traces of an independent set of $G$ in $X$. We are now ready to define the first two rules.

**Reduction Rule 1:** If there exists $u \in X$ such that $\mathsf{conf}_R(\{u\}) > |X|$, remove $u$ from $X$.

**Reduction Rule 2:** If there exists $X' \in \mathcal{X}$ such that $\mathsf{conf}_R(X') > |X|$, add $X'$ to $\mathcal{H}$.

▶ **Lemma 5** (⋆). *Rule 1 and Rule 2 are safe: if $I = (G, X, k)$ is the original input of a-$c$-tdmod-IS and $I^1 = (G^1, X^1, k)$ is the input after the application of Rule 1 or Rule 2, then $I$ and $I^1$ are equivalent.*

**Reduction Rule 3:** If $R$ contains a connected component $R'$ such that for every $X' \in \mathcal{X}$, $\mathsf{conf}_{R'}(X') = 0$, delete $R'$ from the graph and decrease $k$ by $\alpha(R')$.

To prove that Rule 3 is safe we need the following lemma. Recall that we say that $X' \subseteq X$ is an independent set if and only if there is no $H \in \mathcal{H}$ such that $H \subseteq X'$.

▶ **Lemma 6** (⋆). *Let $I = (G, X, k)$ be an instance of a-$c$-tdmod-IS. Let $R'$ be a connected component of $R$. If there exists an independent set $X' \subseteq X$ such that $\mathsf{conf}_{R'}(X') > 0$, then there exists $\bar{X'} \in \mathcal{X}$ such that $\mathsf{conf}_{R'}(\bar{X'}) > 0$.*

▶ **Lemma 7** (⋆). *Rule 3 is safe: if $I = (G, X, k)$ is the original input of a-$c$-tdmod-IS and $I' = (G', X', k')$ is the input after the application of Rule 3, then $I$ and $I'$ are equivalent.*

▶ **Lemma 8** (⋆). *Let $I = (G, X, k)$ be an instance of a-$c$-tdmod-IS, and let $s$ be the number of connected components of $R = V \setminus X$. If none of Rule 1, Rule 2, or Rule 3 can be applied, then $s = \mathcal{O}(|X|^{f(c)+2})$, where $f$ is the function of Lemma 2.*

We are now ready to present our polynomial kernel for a-$c$-tdmod-IS in Algorithm A below, which receives as input $(I, c)$, where $I = (G, X, k)$ and $X$ is a $c$-treedepth modulator.

$A(I, c)$:
1. If $c = 0$, return $X$. Otherwise:
2. While it is possible, apply Rule 1 (this rule suppresses vertices of $X$).
3. While it is possible, apply Rule 2 (this rule adds hyperedges of size at most $f(c)$ to $\mathcal{H}$).
4. Define the set $\mathcal{X}$, and while it is possible, apply Rule 3 (this rule suppresses some connected components of $R$ and decreases $k$ accordingly). Let $I_3 = (G_3, X_3, k_3)$ be the obtained instance, where $G_3 = (V_3, E_3)$ and $R_3 = V_3 \setminus X_3$.
5. For every connected component $R' \subseteq R_3$, compute an optimal treedepth decomposition of root $r_{R'}$. Let $X_r = \cup_{R' \subseteq R_3, R' \text{ connected}} \{r_{R'}\}$ be the set of roots.
6. Let $I' = (G' = (V', E', H'), X', k')$ be defined as follows. Let $V' = V_3$, $X' = X_3 \cup X_r$, and $Z = \{e \in E_3 \mid e \cap X_r \neq \emptyset \text{ and } e \cap X_3 \neq \emptyset\}$. Let $E' = E_3 \setminus Z, \mathcal{H}' = \mathcal{H}_3 \cup Z$ and $k' = k_3$ ($I'$ corresponds to $I_3$ where we added $X_r$ to the modulator, and consequently removed edges $Z$ from $E_3$ and added them as hyperedges included in $X'$. Note that $X'$ is now a $(c-1)$-treedepth modulator).
7. Return $A(I', c-1)$.

▶ **Theorem 9.** *For any fixed $c \geq 0$, Algorithm $A$ is a polynomial kernel for a-$c$-tdmod-IS$/(|X| + |\mathcal{H}|)$. More precisely, for any input $I = (G, X, k)$ (with $G = (V, E, \mathcal{H})$, $R = V \setminus X$) where $X$ is a $c$-treedepth modulator, Algorithm $A$ produces an equivalent instance $\tilde{I} = (\tilde{G}, \tilde{X}, \tilde{k})$ (with $\tilde{G} = (\tilde{V}, \tilde{E}, \tilde{H})$, $\tilde{R} = \tilde{V} \setminus \tilde{X}$) where $|\tilde{X}| \leq \mathcal{O}(|X|^{2^{(c+1)(c+2)/2}})$, $|\tilde{\mathcal{H}}| \leq |\mathcal{H}| + \mathcal{O}(|X|^{2^{(c+1)(c+2)/2}})$, and $\tilde{R} = \emptyset$.*

**Proof.** Observe first that Algorithm $A$ is polynomial for fixed $c$. Indeed, computing $\mathsf{conf}_{R'}(X')$ is polynomial (as $\mathsf{tw}(R') \leq \mathsf{td}(R')$ and it is well-known that IS/tw is FPT [4]) and there are at most $\mathcal{O}(|X|^c)$ applications of Rules 1 and 2, and $\mathcal{O}(s|X|^c)$ applications of Rule 3. Moreover, an optimal treedepth decomposition of each connected component can be computed in FPT time parameterized by $c$, using [17] or [20]. Let us prove the result by induction on $c$. The result is trivially true for $c = 0$. Let us suppose that the result holds for $c-1$ and prove it for $c$. Observe that $X'$ is now a $(c-1)$-treedepth modulator, and thus we can apply the induction hypothesis on $A(I', c-1)$. For any $\ell \in [3]$, let $I_\ell = (G_\ell, X_\ell, k_\ell)$ with $G_\ell = (V_\ell, E_\ell, \mathcal{H}_\ell)$ and $R_\ell = V_\ell \setminus X_\ell$ denote the instance after exhaustive application of Rule $\ell$, respectively.

*Equivalence of the output.* By Lemma 5 and Lemma 7, we know that Rules 1, 2, and 3 are safe, and thus that $I$ and $I_3$ are equivalent. Note that $I_3$ is equivalent to $I'$ as the underlying input is the same (except that some vertices were added to the modulator). As using induction hypothesis $A(I', c-1)$ outputs an instance $\tilde{I}$ equivalent to $I'$, we get the desired result.

*Size of the output.* We have $|X_1| \leq |X|$, $|\mathcal{H}_1| = |\mathcal{H}|$, $|X_2| = |X_1|$, $|\mathcal{H}_2| \leq |\mathcal{H}_1| + |X_1|^{f(c)}$, $|X_3| = |X_2|$, $|\mathcal{H}_3| = |\mathcal{H}_2|$ (by Lemma 8, $s$, the number of connected components of $R_3$, verifies $s = \mathcal{O}(|X_3|^{f(c)+2})$), and $|X'| \leq |X_3| + s$, and $|\mathcal{H}'| \leq |\mathcal{H}_3| + s|X_3|$.

Thus we get $|X'| = \mathcal{O}(|X|^{f(c)+2}) = \mathcal{O}(|X|^{2^{c+1}})$ and $|\mathcal{H}'| = |\mathcal{H}| + \mathcal{O}(|X|^{f(c)+3})$. Using induction hypothesis we get that $|\tilde{X}| = \mathcal{O}(|X'|^{2^{c(c+1)/2}}) = \mathcal{O}(|X|^{2^{(c+1)(c+2)/2}})$, and that $|\tilde{\mathcal{H}}| = |\mathcal{H}'| + \mathcal{O}(|X'|^{2^{c(c+1)/2}}) = |\mathcal{H}| + \mathcal{O}(|X|^{2^c+3}) + \mathcal{O}(|X|^{2^{(c+1)(c+2)/2}}) = |H| + \mathcal{O}(|X|^{2^{(c+1)(c+2)/2}})$. ◀

## 3.2 Deducing a polynomial kernel for IS/$c$-tdmod

Observe first that we can suppose that the modulator is given in the input, i.e., that IS/$c$-tdmod $\leq_{\text{PPT}}$ $c$-tdmod-IS/$|X|$ ($\leq_{\text{PPT}}$ is defined in the full version). Indeed, given an input $(G, x, k)$ of IS/$c$-tdmod (where $x$ denotes the size of a $c$-treedepth modulator), using the $2^c$-approximation algorithm of [11] for computing a $c$-treedepth modulator, wet get in polynomial time a set $X$ such that $|X| \leq 2^c \cdot x$ and $\mathsf{td}(R) \leq c$, where $R = V \setminus X$.

Observe also that IS/$|X|$ $\leq_{\text{PPT}}$ a-$c$-tdmod-IS/$(|X| + |\mathcal{H}|)$ using the same set $X$ and with $|\mathcal{H}| \leq |X|^2$. Now, as usual when using bikernels, we could claim that as IS is Karp NP-hard and as a-$c$-tdmod-IS is in NP, there exists a polynomial reduction from a-$c$-tdmod-IS, implying the existence of a polynomial kernel for IS/$c$-tdmod. However, let us make such a reduction explicit to provide an explicit bound on the size of the kernel.

▶ **Lemma 10** (⋆). *Let $I = (G, k)$ with $G = (X, \mathcal{H})$ be an instance of a-$c$-tdmod-IS as produced by Theorem 9 (as $R = \emptyset$ the set of vertices is reduced to $X$, and $\mathcal{H}$ is a set of hyperedges on $X$). We can build in polynomial time an equivalent instance $I' = (G', k')$ of* IS *with $G' = (V', E')$ where $|V'| \leq \mathcal{O}(|X| \cdot |\mathcal{H}|)$.*

Putting pieces together we immediately get the main theorem of this section.

▶ **Theorem 11.** *For every integer $c \geq 1$,* IS/$c$-tdmod *(or equivalently,* VC/$c$-tdmod*) admits a polynomial kernel on general graphs with $\mathcal{O}(x^{2^{\frac{1}{2}(c+1)(c+2)+1}})$ vertices, where $x$ is the size of a $c$-treedepth modulator.*

# 4 Excluding polynomial kernels for DS/$c$-tdmod on degenerate graphs

Given a graph $G$, we define $G^{c\text{-sub}}$ as the graph obtained from $G$ by subdividing each edge $c$ times. In other words, we add a set $X_e = \{x_e^\ell \mid \ell \in [c]\}$ of $c$ vertices of degree 2 for every edge $e \in E$ of $G$.

▶ **Observation 12** (⋆). *For any $c \geq 0$ and any $k \geq 0$, $G$ has a dominating set of size $k$ if and only if $G^{3c\text{-sub}}$ has a dominating set of size $k + mc$, where $m$ is the number of edges of $G$.*

Let us start with the following proposition, which follows from existing negative results for Dominating Set parameterized by the size of a vertex cover [7].

▶ **Proposition 13** (⋆). DS/$c$-tdmod *does not admit a polynomial kernel on 2-degenerate graphs for any $c \geq 3$ unless NP $\subseteq$ coNP/poly.*

▶ **Observation 14.** DS/1-tdmod *(or equivalently* DS/VC*) admits a polynomial kernel on degenerate graphs. Indeed, given an instance $(G, k)$ of* DS/VC*, we compute in polynomial time a 2-approximate vertex cover $X$ of $G$. If $|X| \leq k$ then we output a trivial* Yes*-instance, otherwise $VC(G) \geq \frac{k}{2}$ and we can apply the polynomial kernel for* DS/$k$ *on degenerate graphs of Philip* et al. *[19].*

Thus, by Proposition 13 and Observation 14, the only remaining case for degenerate graphs is DS/2-tdmod. We would like to point out that the composition of [7] for DS/($k$+VC)

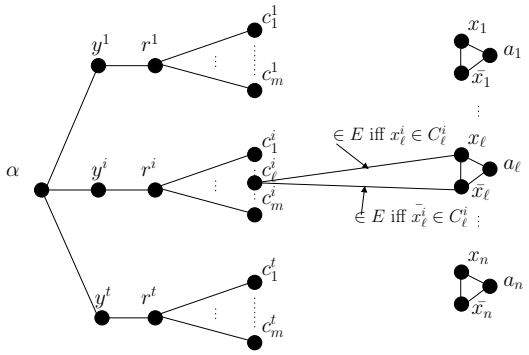



**Figure 2** Example of the OR-cross-composition of Theorem 15.

on general graphs cannot be easily adapted to DS/2-tdmod on degenerate graphs, as for example subdividing each edge also leads to a result for DS/3-tdmod. Thus, we treat the case DS/2-tdmod on degenerate graphs using an ad-hoc reduction.

▶ **Theorem 15.** DS/2-tdmod *does not admit a polynomial kernel on* 4*-degenerate graphs unless* NP $\subseteq$ coNP/poly.

**Proof.** We use an OR-cross-composition (see the full version for the definition) from 3-SAT. We consider $t$ instances of 3-SAT, where for every $i \in [t]$, instance $I^i$ has $m_i$ clauses $\{C^i_j \mid j \in [m_i]\}$ and $n_i$ variables $X^i = \{x^i_\ell \mid \ell \in [n_i]\}$, each clause containing three variables. We can safely assume that for every $i \in [t]$, we have $m_i = m$ and $n_i = n$.

Let us now construct a graph $G = (V, E)$ as follows; see Figure 2 for an illustration. We start by adding to $V$ the set of vertices $\mathcal{X} = \bigcup_{\ell \in [n]} \{x_\ell, \bar{x_\ell}\}$ (and thus $|\mathcal{X}| = 2n$) and $C^i = \{c^i_\ell \mid \ell \in [m]\}$ for every $i \in [t]$. Let $C = \bigcup_{i \in [t]} C^i$. For every $i \in [t]$, $\ell \in [n]$, $j \in [m]$, we set $\{x_\ell, c^i_j\} \in E^i$ (resp. $\{\bar{x_\ell}, c^i_j\} \in E^i$) if and only if $C^i_j$ contains $x^i_\ell$ (resp. $\bar{x}^i_\ell$). We add to $E$ the set $\bigcup_{i \in [t]} E^i$. Then, we add to $V$ the set $A = \{a_\ell \mid \ell \in [n]\}$, and create $n$ triangles by adding to $E$ edges $\{x_\ell, \bar{x_\ell}\}$, $\{a_\ell, x_\ell\}$, and $\{a_\ell, \bar{x_\ell}\}$ for every $\ell \in [n]$. Finally, we add to $V$ the set $Y = \{y^i \mid i \in [t]\}$, $R = \{r^i \mid i \in [t]\}$, and a vertex $\alpha$. Then, for every $i \in [t]$, we add to $E$ edges $\{r^i, c^i_\ell\}$ for every $\ell \in [m]$, edges $\{r^i, y^i\}$, and edges $\{y^i, \alpha\}$. This concludes the construction of $G$. To summarize, $G$ has $3n + t(m + 2) + 1$ vertices (vertices are partitioned into $V = (\mathcal{X} \cup A) \cup (C \cup Y \cup R) \cup \{\alpha\}$) and, in particular, for every $i \in [t]$, $G[\{r^i\} \cup C^i \cup y^i]$ is a star, and $G[\{\alpha\} \cup Y]$ is also a star.

*The* OR*-equivalence.* Let us prove that there exists $i \in [t]$ such that $I^i$ is satisfiable if and only if $G$ has a dominating set of size at most $k = n + t$. Suppose first, without loss of generality, that $I^1$ is satisfiable, and let $S_\mathcal{X} \subseteq \mathcal{X}$ be the set of $n$ literals corresponding to this assignment (thus for every $\ell \in [n]$ we have $|S_\mathcal{X} \cap \{x_\ell, \bar{x_\ell}\}| = 1$). Let $S = S_\mathcal{X} \cup y^1 \cup (R \setminus \{r^1\})$. We have $|S| = n + t$, and $S$ is a dominating set of $G$ as

- $\mathcal{X} \cup A$ is dominated by $S_\mathcal{X}$,
- $C^1$ is dominated by $S_\mathcal{X}$ as it corresponds to an assignment satisfying $I^1$, and for every $i \in [t], i \geq 2$, $C^i$ is dominated by $r^i$,
- $y^1 \in S$, and for every $i \in [t], i \geq 2$, $y^i$ is dominated by $r^i$,
- $r^1$ is dominated by $y^1$, and for any $i \in [t], r \geq 2$, $r^i \in S$, and
- $\alpha$ is dominated by $y^1$.

For the other direction, let $S = S_1 \cup S_2$, with $S_1 = S \cap (\mathcal{X} \cup A)$, be a dominating set of $G$ of size at most $k = n + t$. Without loss of generality, we can always suppose that $S_1 \subseteq \mathcal{X}$, as if $a_\ell \in S$ we can always remove $a_\ell$ from $S$ and add (arbitrarily) $x_\ell$ or $\bar{x_\ell}$.

Let us first prove that $|S_1| = n$. Observe first that $|S_1| \geq n$ as dominating $A$ requires at least $n$ vertices. Suppose now by contradiction that $|S_1| > n$. Then, there would remain at most $t-1$ vertices to dominate $R$, which is not possible. Note that we even have that for any $\ell \in [n]$, $|S_1 \cap \{x_\ell, \bar{x_\ell}\}| = 1$, as every $a^\ell$ must be dominated and $|S_2| = t$.

Let us now analyze $S_2$ (recall that, by definition, $S_2 \subseteq (C \cup Y \cup R) \cup \{\alpha\}$). We cannot have that for every $i \in [t]$, $|S_2 \cap (C^i \cup r^i)| \geq 1$, as otherwise there would be no remaining vertex to dominate $\alpha$. Thus, there exists $i_0$ such that $|S_2 \cap (C^{i_0} \cup r^{i_0})| = 0$. This implies that $C^{i_0}$ is dominated by $S_1$. As for every $\ell \in [n]$, $|S_1 \cap \{x_\ell, \bar{x_\ell}\}| = 1$, $S_1$ corresponds to a valid truth assignment that satisfies all the $C^i_\ell$'s, $\ell \in [m]$, and the instance $I^{i_0}$ is satisfiable.

*Size of the parameter.* Let $M = \mathcal{X} \cup A \cup \{\alpha\}$. As $G[V \setminus M]$ contains $t$ disjoint stars, we have that 2-tdmod $(G) \leq |M| \leq \text{poly}(n)$, as required.

*Degeneracy.* Let us prove that $G$ is 4-degenerate. Observe that any vertex in $C$ has degree at most 4 (three neighbors in $\mathcal{X}$ and one in $R$). Thus, any ordering of $V(G)$ of the form $(C, R, Y, \alpha, \mathcal{X}, A)$ (with arbitrary order within each set) is a 4-elimination order of $G$. ◀

Note that for DS/$c$-tdmod with $c \geq 3$, the bound in the degeneracy given by Proposition 13 is best possible, as DS can be easily solved in polynomial time on 1-degenerate graphs, i.e., forests. On the other hand, for $c = 2$, in view of Theorem 15 only the existence of polynomial kernels for DS/2-tdmod on 2-degenerate and 3-degenerate graphs remains open.

## 5 Concluding remarks and further research

In this article we studied the existence of polynomial kernels for problems parameterized by the size of a $c$-treedepth modulator, on graphs that are not sparse. On the positive side, we proved that Vertex Cover (or equivalently, Independent Set) parameterized by the size $x$ of a $c$-treedepth modulator admits a polynomial kernel on general graphs with $x^{2^{\mathcal{O}(c^2)}}$ vertices, for every $c \geq 1$. A natural direction is to improve the size of this kernel. Since Vertex Cover parameterized by the distance to a disjoint collection of cliques of size at most $c$ does not admit a kernel with $\mathcal{O}(x^{c-\epsilon})$ vertices unless NP $\subseteq$ coNP/poly [16], and since a clique of size $c$ has treedepth $c$, the same lower bound applies to our parameterization; in particular, this rules out the existence of a *uniform* kernel. However, there is still a large gap between both bounds, hence there should be some room for improvement.

On the negative side, we proved that Dominating Set parameterized by the size of a $c$-treedepth modulator does not admit a polynomial kernel on degenerate graphs for any $c \geq 2$. As Dominating Set with this parameterization admits a polynomial kernel on nowhere dense graphs [11], it follows that sparse graphs constitute the border for the existence of polynomial kernels. This leads us to the following natural question: are there smaller parameters for which Dominating Set still admits polynomial kernels on sparse graphs? Since considering as parameter the treedepth of the input graph does not allow for polynomial kernels (see the full version), we may consider as parameter the size $x$ of a vertex set whose removal results in a graph of treedepth at most $b(x)$, for a function $b$ that is not necessarily constant. We prove in the full version that Dominating Set does *not* admit polynomial kernels on graphs of bounded expansion for $b(x) = \Omega(\log x)$, unless NP $\subseteq$ coNP/poly. On the other hand, by combining the approach of Garnero *et al.* [12] to obtain explicit kernels via dynamic programming with the techniques of Gajarskỳ *et al.* [11] on graphs of bounded expansion, it can be shown – we omit the details here – that Dominating Set admits a polynomial kernel for $b(x) = \mathcal{O}(\log \log \log x)$ on graphs of bounded expansion whose expansion function $f$ is not too "large" (that is, the function $F$ that bounds the grad with rank $d$ of the graphs

in the family, see [17]), namely $f(d) = 2^{\mathcal{O}(d)}$. While this result is somehow anecdotal, we think that it may be the starting point for a systematic study of this topic.

## References


1 Faisal N Abu-Khzam, Michael R Fellows, Michael A Langston, and W Henry Suters. Crown structures for vertex cover kernelization. *Theory of Computing Systems*, 41(3):411–430, 2007.

2 Hans L Bodlaender, Rodney G Downey, Michael R Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009.

3 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. In *Proc. of the 50th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 629–638. IEEE Computer Society, 2009.

4 B. Courcelle. The monadic second-order theory of graphs I: recognisable sets of finite graphs. *Information and Computation*, 85(12-75):663, 1990.

5 Marek Cygan, Lukasz Kowalik, and Marcin Pilipczuk. Open problem session of the Workshop on Kernelization (WorKer), Warsaw, Poland, 2013. Summary available at `http://worker2013.mimuw.edu.pl/slides/worker-opl.pdf`.

6 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. On the hardness of losing width. *Theory of Computing Systems*, 54(1):73–82, 2014.

7 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and ids. *ACM Transactions on Algorithms*, 11(2):13, 2014.

8 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. In *Proc. of the 21st ACM-SIAM Symp. on Disc. Algorithms (SODA)*, pages 503–510, 2010.

9 Fedor V. Fomin and Torstein J. F. Strømme. Vertex cover structural parameterization revisited. *CoRR*, abs/1603.00770, 2016. `arXiv:1603.00770`.

10 L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *Journal of Computer and System Sciences*, 77(1):91–106, 2011.

11 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. *Journal of Computer and System Sciences*, 84:219–242, 2017.

12 Valentin Garnero, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. Explicit linear kernels via dynamic programming. *SIAM Journal on Discrete Mathematics*, 29(4):1864–1894, 2015.

13 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A Structural View on Parameterizing Problems: Distance from Triviality. In *Proc. of the 1st International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 3162 of *LNCS*, pages 162–173, 2004.

14 Bart Jansen and Hans Bodlaender. Vertex cover kernelization revisited: Upper and lower bounds for a refined parameter. In *Proc. of the 28th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 9 of *LIPIcs*, pages 177–188, 2011.

15 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Transactions on Algorithms*, 12(2):21, 2016.

16 Diptapriyo Majumdar, Venkatesh Raman, and Saket Saurabh. Kernels for structural parameterizations of vertex cover - case of small degree modulators. In *Proc. of the 10th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 43 of *LIPIcs*, pages 331–342, 2015.

**17** Jaroslav Nesetril and Patrice Ossona De Mendez. *Sparsity: Graphs, Structures, and Algorithms*. Algorithms and Combinatorics. Springer, 2012. URL: `https://hal.archives-ouvertes.fr/hal-00768681`.

**18** Rolf Niedermeier. Reflections on multivariate algorithmics and problem parameterization. In *Proc. of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 5 of *LIPIcs*, pages 17–32, 2010.

**19** Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Solving dominating set in larger classes of graphs: FPT algorithms and polynomial kernels. In *Proc. of the 17th Annual European Symposium on Algorithms (ESA)*, volume 5757 of *LNCS*, pages 694–705, 2009.

**20** Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In *Proc. of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942, 2014.

# Solving and Sampling with Many Solutions: Satisfiability and Other Hard Problems*†

**Jean Cardinal[1], Jerri Nummenpalo[2], and Emo Welzl[3]**

1 **Université Libre de Bruxelles (ULB), Computer Science Department, Brussels, Belgium**
jcardin@ulb.ac.be
2 **ETH Zürich, Department of Computer Science, Zürich, Switzerland**
njerri@inf.ethz.ch
3 **ETH Zürich, Department of Computer Science, Zürich, Switzerland**
emo@inf.ethz.ch

## Abstract

We investigate parameterizing hard combinatorial problems by the size of the solution set compared to all solution candidates. Our main result is a uniform sampling algorithm for satisfying assignments of 2-CNF formulas that runs in expected time $O^*(\varepsilon^{-0.617})$ where $\varepsilon$ is the fraction of assignments that are satisfying. This improves significantly over the trivial sampling bound of expected $\Theta^*(\varepsilon^{-1})$, and on all previous algorithms whenever $\varepsilon = \Omega(0.708^n)$. We also consider algorithms for 3-SAT with an $\varepsilon$ fraction of satisfying assignments, and prove that it can be solved in $O^*(\varepsilon^{-2.27})$ deterministic time, and in $O^*(\varepsilon^{-0.936})$ randomized time. Finally, to further demonstrate the applicability of this framework, we also explore how similar techniques can be used for vertex cover problems.

**1998 ACM Subject Classification** F.2.2 Theory of Computation, Nonnumerical Algorithms and Problems, Computations on discrete structures, G.2.1. Discrete Mathematics, Combinatorics, Combinatorial algorithms

**Keywords and phrases** Satisfiability, Sampling, Parameterized complexity

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.11

## 1 Introduction

In order to cope with the computational complexity of combinatorial optimization and satisfiability problems without sacrificing correctness guarantees, one can consider a family of instances for which a certain parameter is bounded, and analyze the complexity of algorithms as a function of this parameter. While it is now commonplace in combinatorial optimization to define the parameter as the *size* of a solution, we here consider computationally hard problems parameterized by the *number* of solutions. More precisely, we will consider satisfiability problems in which we are promised that a fraction at least $\varepsilon$ of all possible assignments are satisfying, and graph covering problems in which a fraction at least $\varepsilon$ of all vertex subsets of a certain size are solutions.

Counting and sampling solutions to CNF formulas and more generally to CSP formulas has important practical applications. For example, in verification and artificial intelligence [12];

* This work started at the 2016 Gremo Workshop on Open Problems (GWOP), on June 6-10 at St. Niklausen, OW, Switzerland.
† A full version of this paper is available on arXiv [2], https://arxiv.org/abs/1708.01122.


and Bayesian inference [13]. Recent algorithmic developments have made possible practical algorithms that can tackle industrial scale problems [10].

In contrast to that line of work we focus on the exact complexity of sampling, in particular to sampling solutions for 2-CNF formulas, and show that we can significantly improve on the *trivial sampling algorithm* that repeatedly samples uniformly in the search space and terminates after $\varepsilon^{-1}$ steps on average. A few previous works have also considered satisfiability problems under the promise that there are many solutions, most notably from Hirsch [6], and more recently from Kane and Watanabe [9]. Their focus has been on deterministic algorithms and we extend their work while also adding the consideration of randomized algorithms for $k$-SAT.

Before detailing our contributions more precisely, we briefly summarize the current state of knowledge regarding this family of questions.

## 1.1 Background and previous work on satisfiability

Hirsch [6] developed a deterministic algorithm that finds a satisfying assignment for a $k$-CNF formula $F$ with an $\varepsilon$ fraction of satisfying assignments in time $O^*(\varepsilon^{-\delta_k})$ where $(\delta_k)_{k=2}^{\infty}$ is a positive increasing sequence defined by the roots of the characteristic polynomials of certain recurrence relations. The constant obtained for $k = 3$ is $\delta_3 \approx 7.27$. The main idea in his algorithm is that such formulas $F$ have *short implicants* which are satisfying assignments that need to fix only few variables – in this case only $O(\log \varepsilon^{-1})$ many – and such assignments can be found relatively fast with a branching algorithm. Trevisan [17] proposed a similar algorithm to that of Hirsch but with an explicit running time of $O^*(\varepsilon^{-(\ln 4)k2^k})$. Although his algorithm is slightly simpler, the performance guarantees, at least for small $k$, are worse.

Kane and Watanabe [9] looked at general CNF formulas in a similar setting. They assume that $\varepsilon \geq 2^{-n^{\delta}}$, that the number of clauses is bounded by $n^{1+\delta'}$ and that $\delta + \delta' < 1$. Under these conditions they show that the formula has a short implicant that only fixes a linear fraction of the variables and they provide a $O^*(2^{n^{\beta}})$ time algorithm for finding a solution with $\beta < 1$.

Classical derandomization tools naturally apply in this context. For arbitrary CNF formulas on $n$ variables with $\varepsilon 2^n$ satisfying assignments, one can obtain a deterministic algorithm by using a pseudorandom generator that $\varepsilon$-fools depth-2 circuits. A result by De et al. [3] provides such pseudorandom generators with seed length $O\left(\log n + \log^2 \frac{m}{\varepsilon} \log\log \frac{m}{\varepsilon}\right)$. By enumerating over all seeds, we obtain a running time of $O^*\left(\left(\frac{n}{\varepsilon}\right)^{c \cdot \log \frac{n}{\varepsilon}}\right)$ for some constant $c$ (assuming there are $\mathrm{poly}(n)$ clauses). A recent result of Servedio and Tan improves this running time to $n^{\tilde{O}(\log\log n)^2}$ for any $\varepsilon \geq 1/\mathrm{poly}\log(n)$ [16].

We let *Sample-2-SAT* denote the problem of sampling exactly and uniformly a satisfying assignment. Due to self-reducibility of satisfiability, any algorithm for the counting problem #2-SAT can be used to solve Sample-2-SAT with only a multiplicative polynomial loss in runtime. In fact, so far the best algorithm for Sample-2-SAT is Wahlström's #2-SAT algorithm [18] that runs in time $O(1.238^n)$. In contrast to the exponential time algorithms, 2-SAT can be solved in linear time with the classical algorithm of Aspvall et al. [1]. We note that while Sample-2-SAT is between 2-SAT and #2-SAT in complexity, under the assumption $RP \neq NP$ it is not possible to uniformly or even almost uniformly sample satisfying assignments in polynomial time. We can use a simple threefold reduction to prove this:

- The constraints for an independent set in a graph can be modeled as a 2-SAT formula. Therefore a polynomial time algorithm for Sample-2-SAT would give a polynomial time

algorithm for *Sample-IS*. (sampling uniformly among independent sets of any size). The same holds for approximate versions of the problems.
- Such sampling algorithms would yield a fully polynomial randomized approximation scheme (FPRAS) for #IS. See for example the article of Jerrum et al. [8].
- Lastly, such an FPRAS exists only if $RP = NP$. For details see for example the book by Jerrum [7, Chapter 7, Proposition 7.7].

Even when relaxing Sample-2-SAT to almost uniform sampling, the best algorithm is still the one based on Wahlström's counting algorithm. This is in contrast to $k$-CNF formulas with $k \geq 3$ which have an exponential gap between exact and almost uniform sampling. More precisely, the gap is between exact and approximate counting. See Schmitt and Wanka [14] for a table of the best algorithms.

## 1.2 Our results

In Section 2 we recall Hirsch's [6] algorithm for finding a satisfying assignment for a $k$-CNF $F$ with a fraction $\varepsilon$ of satisfying assignments. We slightly generalize his analysis to also cover improved branching rules for $k$-SAT. The resulting deterministic algorithms have running times of $O^*(\varepsilon^{-\lambda_k})$ for some positive increasing sequence $(\lambda_k)_{k=2}^{\infty}$, where for instance $\lambda_3 \leq 2.27$. We demonstrate how similar techniques can be used for finding vertex covers and we give a deterministic algorithm running in time sublinear in $\varepsilon^{-1}$ for instances of $k$-vertex cover with at least $\varepsilon\binom{n}{k}$ solutions and $k$ bounded by some fraction of $n$.

In Section 3 we prove our main result, Theorem 7, which describes an algorithm for Sample-2-SAT that runs in expected time $O^*(\varepsilon^{-0.617})$. It therefore improves on the algorithm based on Wahlström's algorithm [18] when $\varepsilon = \Omega(0.708^n)$, or equivalently when $F$ has $\Omega(1.415^n)$ satisfying assignments. We leave it as an open problem to decide whether sampling solutions to 3-CNF formulas can be done in time $O^*(\varepsilon^{-\delta})$ with $\delta < 1$ and discuss why the 2-CNF case does not generalize. In Proposition 8 we show how to solve 3-SAT in time $O(\varepsilon^{-0.936}(m+n))$ using similar ideas.

## 1.3 Notation

For a Boolean variable $x$ we denote its *negation* by $\bar{x}$ and for a set $V$ of Boolean variables let $\overline{V}$ be the set of negated variables. A *literal* is either a Boolean variable or its negation and in the former case we call the literal *positive* and in the latter we call it *negative*. We think of a *CNF formula*, or simply a *formula*, $F$ over a variable set $V$ as a set $F = \{C_1, C_2, \ldots, C_m\}$ of *clauses* where each clause $C_i \subset V \cup \overline{V}$ is a set of literals without both $x$ and $\bar{x}$ in the same clause for any variable $x \in V$. By a $k$-CNF formula and by a $(\leq k)$-CNF we denote CNF formulas in which every clause has cardinality exactly $k$ or at most $k$, respectively. We let $\mathrm{vbl}(F) \subseteq V$ denote the set of variables that appear in $F$ either as a positive or negative literal. The *empty formula* is denoted by $\{\}$ and the *empty clause* by $\square$. An *assignment* to the variables in the formula $F$ is a function $\alpha : V \to \{0,1\}$ and it is said to *satisfy* $F$ if every clause $C \in F$ is satisfied, namely, if the clause contains a literal whose value is set to 1 under the assignment. A satisfying assignment is also called a *solution*. The empty formula is satisfied by any assignment to the variables and the empty clause by none. The set of all satisfying assignments of a formula $F$ over $V$ is denoted $\mathrm{sat}_V(F)$, and we omit the subscript $V$ when it is clear from the context. A *partial assignment* to $F$ is a function $\beta : W \to \{0,1\}$ with $W \subseteq V$ and we let $F^{[\beta]}$ be the formula over the variables $V \setminus W$ which is attained from $F$ by removing each clause of $F$ that is satisfied under $\beta$ and then removing all literals assigned to 0 from the remaining clauses. If $u \in V \cup \overline{V}$ is a literal and $i \in \{0,1\}$

we let $F^{[u \mapsto i]}$ denote $F^{[\beta]}$ where $\beta$ is the partial assignment that maps only $u$ to $i$. By *unit clause reduction* we refer to the process of repeatedly setting variables to satisfy the unit clauses until finishing the process by exhausting the unit clauses or finding the empty clause.

All the logarithms are in base 2 unless noted otherwise.

## 2 Deterministic algorithms and Hirsch's method

In this section we consider Hirsch's method [6] for finding a satisfying assignment to a $k$-CNF formula, and extend the analysis to accommodate any branching rule.

We first briefly recall basic definitions on branching algorithms. A *complexity measure* $\mu$ is a function that assigns a nonnegative value $\mu(F)$ to every instance $F$ of some particular problem. Given a problem and a complexity measure $\mu$ for it, we say that an algorithm correctly solving the problem is a *branching algorithm* (with respect to $\mu$) if for every instance $F$ the algorithm computes a list $(F_1, \ldots, F_t)$ of instances of the same problem, recursively solves the $F_i$'s, and finally combines the results to solve $F$. Finding the list $(F_1, \ldots, F_t)$ and recursively solving each of them is called a *branching*. Letting $b_i = \mu(F) - \mu(F_i)$ we call the vector $(b_1, \ldots, b_t)$ the *branching vector* associated to the branching. Lastly, the *branching number* $\tau(b_1, \ldots, b_t)$ is defined as the smallest positive solution of the equation $\sum_{i=1}^{t} x^{-b_i} = 1$. If $\lambda$ is the largest branching number of any possible branching in the algorithm and $T(F)$ is the time used to find the branching and to combine the results after the recursive calls, then the running time of the algorithm can be bounded by $O(T(F)\lambda^{\mu(F)})$.

Following Hirsch [6], we consider a *breadth-first* version of such a branching algorithm, taking a $k$-CNF Boolean formula $F$ as input. We use the number of variables as a measure, and branch on partial assignments $\beta_i$, each fixing exactly $b_i$ variables. The set $\Phi_\ell$ in the algorithm below eventually contains the formulas constructed from input $F$ after fixing exactly $\ell$ variables.

1. set $\ell \leftarrow 0$, $\Phi_0 \leftarrow \{F\}$, and $\Phi_\ell \leftarrow \emptyset$ for all $\ell > 0$.
2. if $\{\} \in \Phi_\ell$, then stop and return the so far fixed variables
3. for each $F \in \Phi_\ell$ such that $\square \notin F$:
   a. find a collection of $t$ partial assignments of the form $\beta_i : W_i \to \{0,1\}$, where $W_i \subseteq \mathrm{vbl}(F)$
   b. for each $i \in [t]$:
      i. $\Phi_{\ell+b_i} \leftarrow \Phi_{\ell+b_i} \cup \{F^{[\beta_i]}\}$
4. $\ell \leftarrow \ell + 1$; if $\ell \leq n$ then go to step 2

For this algorithm to be correct, the partial assignments in 3a have to of course be chosen according to a correct branching rule. The complete collection $\Phi_\ell$ can be seen as a collection of nodes of the search tree of the recursive algorithm, and is referred to as the $\ell$th *floor* of the tree. The following lemma holds [6].

▶ **Lemma 1.** *$|\Phi_\ell| \leq \lambda^\ell$ where $\lambda$ is the maximum branching number of the recursion tree.*

The following result was proved by Hirsch in the special case of the simple Monien-Speckenmeyer algorithm [11], in which the branching vector was $(1, 2, \ldots, k)$. We generalize it to arbitrary branching vectors. The proof is left for the full version of this paper [2].

▶ **Theorem 2.** *Consider a $k$-CNF formula $F$ with $n$ variables and $m$ clauses, and suppose it has at least $\varepsilon 2^n$ satisfying assignments. Then any breadth-first branching algorithm for $k$-SAT with maximum branching number $\lambda_k < 2$ runs in time $O^*(\varepsilon^{-B})$ on this instance, where $B := 1/(\log_{\lambda_k} 2 - 1)$.*

To get concrete bounds from Theorem 2 it remains to find good branching rules for $k$-SAT. The improved algorithm by Monien and Speckenmeyer [11] for $k$-SAT uses the notion of autarkies and the branching vectors appearing in the algorithm are (1) and $(1, 2, \ldots, k-1)$ of which the latter has the worse branching number. This directly yields the following result for $k = 3$.

▶ **Theorem 3.** *Given a 3-CNF formula $F$ on $n$ variables and an $\varepsilon > 0$ with the guarantee that $|\mathrm{sat}(F)| \geq \varepsilon 2^n$, one can find a satisfying assignment for $F$ in deterministic time $O^*\left(\varepsilon^{-2.27}\right)$.*

## 2.1 Vertex cover

The technique we have seen is not unique to satisfiability but extend easily to known graph problems. As an example, we now consider the *vertex cover problem*: given a graph $G$ and an integer $k$, does there exist a subset $S \in \binom{V(G)}{k}$ such that $\forall e \in E(G), e \cap S \neq \emptyset$? The optimization version consists of finding a smallest subset $S$ satisfying the condition. We consider exact algorithms, hence the problem is equivalent to the maximum independent set problem (consider $V(G) \setminus S$). This is naturally related to the previous results on 2-SAT: the vertex cover problem can be cast as finding a minimum-weight satisfying assignment for a monotone 2-CNF formula.

We first briefly recall a standard algorithm for finding a minimum vertex cover in a graph $G$ on $n$ vertices, if one exists, in time $O^*(1.3803^n)$. First note that if the maximum degree of the graph is 2, then the problem can be solved in polynomial time. Otherwise, pick a vertex $v$ of degree at least 3, and return the minimum of $1 + VC(G - v)$ and $VC(G - v - N(v))$, where $VC$ are recursive calls, and $N(v)$ is the set of neighbors of $v$ in $G$. The running time $T(n)$ obeys the recurrence $T(n) = T(n-1) + T(n-4)$, solving to the claimed bound. We can also analyze it with respect to the size $k$ of the sought cover, yielding $T(k) = T(k-1) + T(k-3)$, solving to $1.4656^k$. In the latter, we do not count the total number of vertices that are processed, but only those that are part of the solution. Hence we can distinguish the branching number $\lambda$ related to the number of vertices processed and the branching number $\rho$ related to the number of vertices included in the vertex cover (equivalently, the weight of the current partial assignment). In our case, we have $\rho < 1.4656$.

We now consider instances of the vertex cover problem in which we are promised that there are at least $\varepsilon\binom{n}{k}$ vertex covers. Given a branching algorithm, we can parse its search tree in breadth-first order, by associating with each node the number of vertices included in $S$ so far (that is, the weight of the partial assignment). We define $\Phi_\ell$ as the set of nodes with such value $\ell$, and call it the $\ell$th floor. The following lemma is similar to Lemma 1.

▶ **Lemma 4.** $|\Phi_\ell| \leq \rho^\ell$.

After generating the $\ell$th floor $\Phi_\ell$, there are at most $\rho^\ell\binom{n-\ell}{k-\ell}$ remaining covers to check. If this is less than the total number of solutions of size $k$, we are done. The following statement gives an upper bound on the number of levels of the tree we need to parse. We leave the proof to the full version of this paper [2].

▶ **Lemma 5.** *Let $\ell^* := \ln(\frac{1}{\varepsilon})/\ln(\frac{n}{\rho k})$. Then for $k, n >> \ell^*$ and $k \leq n/\rho$, we have*

$$\rho^\ell \binom{n-\ell}{k-\ell} \geq \varepsilon \binom{n}{k} \Rightarrow \ell \leq \ell^*.$$

For $n$ large enough, Lemma 5 implies that if $\ell > \ell^*$ then the number of remaining solutions is smaller than the promised number $\varepsilon\binom{n}{k}$, and either we have found one already, or greedily

completing any partial solution leads to a solution. Hence the running time is within a linear factor of $\rho^{\ell^*}$, which simplifies as follows.

▶ **Theorem 6.** *Given a Vertex Cover instance composed of a graph $G$ on $n$ vertices, a number $k < n/\rho$, and an $\varepsilon > 0$ with the guarantee that $G$ has at least $\varepsilon\binom{n}{k}$ vertex covers of size $k$, one can find such a vertex cover in deterministic time*

$$O^*\left(\varepsilon^{-\frac{\log\rho}{\log(\frac{n}{\rho k})}}\right),$$

*where $\rho$ is the branching number of an exact branching algorithm for $k$-vertex cover. In particular, this holds for $\rho = 1.4656$.*

Note that the running time remains sublinear in $1/\varepsilon$ for all values of $k$ such that $\frac{\log\rho}{\log(\frac{n}{\rho k})} < 1 \Leftrightarrow k < n/\rho^2$. Hence for those values of $k$, and in particular when $k = o(n)$, we have a deterministic algorithm for $k$-vertex cover whose complexity improves on the trivial sampling algorithm.

## 3 Randomized algorithms for Sample-2-SAT and for 3-SAT

In this section we present our algorithm for Sample-2-SAT with an expected running time of $O\left(\varepsilon^{-0.617}(m+n)\right)$ on 2-CNF formulas with more than $\varepsilon$ fraction of satisfying assignments. The parameter $\varepsilon$ does not need to be a constant and the algorithms can be easily modified so that they do not need to know $\varepsilon$ in advance. Before stating and proving our main result we consider a warm-up algorithm that gives a weaker bound but already highlights some of the main ideas. In the end we discuss the complications of generalizing our method to Sample-3-SAT and see how to solve 3-SAT in expected time $O\left(\varepsilon^{-0.940}(m+n)\right)$ using similar techniques as for Sample-2-SAT.

Schmitt and Wanka [14] have used analogous ideas to approximately count the number of solutions in $k$-CNF formulas.

### 3.1 A warm-up algorithm for Sample-2-SAT

We will start with a warm-up algorithm that we then improve. Let $F$ be a 2-CNF formula over the variable set $V$ with $n := |V|$ and with $m$ clauses. Let $S \subseteq F$ be a greedily chosen maximal set of variable disjoint clauses. We make the following remarks.

- Any satisfying full assignment for $F$ must in particular satisfy $S$ and is therefore an extension of one of the $3^{|S|}$ partial assignments to $\mathrm{vbl}(S)$ that satisfy all clauses in $S$.
- Because of maximality any partial assignment of the form $\alpha : \mathrm{vbl}(S) \to \{0,1\}$ has the property that $F^{[\alpha]}$ is a $(\leq 1)$-CNF.
- Counting and sampling of solutions of a $(\leq 1)$-CNF is easily done in linear time.

The set $S$ allows us on one hand to do improved rejection sampling and on the other hand to device a branching based sampling. More concretely, consider the following two algorithms that use $S$.

1. Sample uniformly among all full assignments for $F$ that satisfy all the clauses in $S$ until finding one that satisfies $F$.
2. Go through all $3^{|S|}$ partial assignments $\alpha : \mathrm{vbl}(S) \to \{0,1\}$ that satisfy $S$ and for each $\alpha$ compute $A_\alpha := |\mathrm{sat}_{V\setminus\mathrm{vbl}(S)}(F^{[\alpha]})|$, i.e., the number of satisfying assignments in $F^{[\alpha]}$. Then $A := \sum_\alpha A_\alpha$ is the number of satisfying assignments in $F$. Draw one partial

assignment $\alpha^*$ at random so that $\Pr(\alpha^* = \alpha) = A_\alpha/A$. For the remaining variables choose an assignment $\beta^* : V \setminus \mathrm{vbl}(S) \to \{0,1\}$ uniformly among all assignments satisfying $F^{[\alpha^*]}$. Output the full assignment which when restricted to $\mathrm{vbl}(S)$ is $\alpha^*$ and when restricted to $V \setminus \mathrm{vbl}(S)$ is $\beta^*$.

The correctness of the first algorithm is clear since any assignment satisfying $F$ must also satisfy $S$. One sample can also be drawn in linear time. Because the clauses of $S$ are variable disjoint, the pool of assignments we are sampling from has $(\frac{3}{4})^{|S|}2^n$ assignments and it contains all the at least $\varepsilon 2^n$ satisfying assignments. Therefore the probability of one sample being satisfying is at least $(\frac{4}{3})^{|S|}\varepsilon$, implying an expected runtime of $O\left(\varepsilon^{-1}(\frac{3}{4})^{|S|}(m+n)\right)$ for the first algorithm.

We need the second algorithm to balance the first one when $|S|$ is small. For the correctness we observe that the partial assignments $\alpha$ partition the solution space in the sense that $A = \sum_\alpha A_\alpha = |\mathrm{sat}_V(F)|$ and a simple calculation shows that the output distribution is uniform over $\mathrm{sat}_V(F)$. With the remarks made before the algorithm description we conclude that the runtime of the second algorithm is $O(3^{|S|}(m+n))$. If space is a concern, the sampling of $\alpha^*$ can be done in linear space without storing the numbers $A_\alpha$ as follows: Sample a uniform number $r$ from $\{1,\dots,A\}$ and go through the partial assignments $\alpha$ again in the same order and output the first $\alpha$ for which the total number of assignments counted up to that point reaches at least $r$.

For any given $S$ we can choose the better of the two algorithms which gives an expected runtime guarantee of

$$O\left(\max_{|S|}\left\{3^{|S|}, \varepsilon^{-1}\left(\frac{3}{4}\right)^{|S|}\right\}\cdot(m+n)\right) = O\left(\varepsilon^{-\log_4 3}(m+n)\right) \tag{1}$$

where $\log_4 3 < 0.793$. Note that we do not need to know $\varepsilon$ in advance to get the same runtime guarantee as we can simulate running both of the algorithms in parallel until one finishes.

## 3.2 A faster algorithm for Sample-2-SAT

In the warm-up algorithm we used the set $S$ on the one hand to reduce the size of the set of assignments we are sampling from and on the other hand we used it as a small size *hitting set* for the clauses in $F$: every clause in $F$ contained at least one variable from $\mathrm{vbl}(S)$. To improve we will do two things. Firstly, we will consider more complicated independent structures that improve on both aspects above, giving us both a smaller size sampling pool and a better hitting set. Secondly, we notice that it is not necessary to always use an exact hitting set in the counting procedure but an "almost hitting set" is enough. Namely, if some small set of variables hits almost all clauses we can count the number of solutions to the remaining relatively small ($\leq 2$)-SAT with a good exponential time algorithm for #2-SAT.

We introduce first some notation. For $i \in \mathbb{N}$ we call a set of clauses $S$ an *$i$-star* if $|S| = i$ and if there exists a variable $x$ such that for any pair of distinct clauses $C, D \in S$ we have $\{x\} = \mathrm{vbl}(C) \cap \mathrm{vbl}(D)$. A *star* is an $i$-star for some $i$. For $i \geq 2$ we call the variable $x$ the *center* of the star and any other variable is called a *leaf*. For 1-stars we consider both of the variables as centers and neither of them as leaves. A star is called *monotone* if the center appears as the same literal in every clause of the star. We call a set $T$ of exactly three clauses a *triangle* if every 2-element subset of $T$ is a star and $T$ is not itself a star. Finally, we call a family $\mathcal{M}$ of CNF formulas *independent* if no two formulas in $\mathcal{M}$ share common variables.

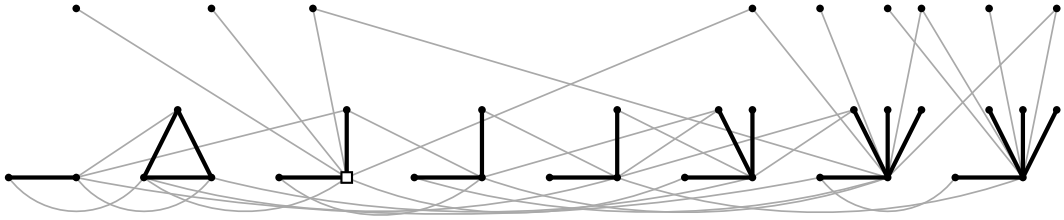


**Figure 1** A possible construction of $\mathcal{M}_4$ for a formula $F$ that is displayed as a graph with the variables as vertices and edges between variables appearing in the same clause. The subformulas of $F$ that make up $\mathcal{M}_4$ are given by the components defined by the black bold edges. The edges that form up $\mathcal{M}_0$ are the horizontal black bold edges. There is one non-monotone 2-star in $\mathcal{M}_4$ and it is denoted by the square center vertex.

▶ **Theorem 7.** *Let $F$ be a 2-CNF formula on $n$ variables and $m$ clauses and let $\varepsilon > 0$ be such that $|\mathrm{sat}(F)| \geq \varepsilon 2^n$. A uniformly random satisfying assignment for $F$ can be found in expected time $O\left(\varepsilon^{-\delta}(m+n)\right)$ where $\delta < 0.617$.*

**Proof.** Let $V$ be the variable set of $F$ and let $k \geq 2$ be a constant independent of $\varepsilon$ that we fix later. We start by constructing a sequence $(\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_k)$ of $k+1$ independent families of formulas where every family consists of subformulas of $F$.

Let $\mathcal{M}_0$ be any independent 1-maximal family of 1-stars (clauses) in $F$. That is, in addition to maximality we require further that there is no clause in the family whose removal would allow the addition of two clauses in its place. We can find $\mathcal{M}_0$ with a greedy algorithm in linear time[1].

To construct $\mathcal{M}_1$ from $\mathcal{M}_0$, we add clauses of $F$ to the 1-stars of $\mathcal{M}_0$ greedily to update them into non-monotone 2-stars or triangles while maintaining independence. As a result $\mathcal{M}_1$ is an independent family of subformulas of $F$ that consists of 1-stars, non-monotone 2-stars, and triangles and no 1-star can be turned into the other two types by adding clauses of $F$ to it without revoking independence.

For $i = 2, \ldots, k$ we construct $\mathcal{M}_i$ from $\mathcal{M}_{i-1}$ by greedily adding clauses of $F$ to the monotone $(i-1)$-stars to turn them into monotone $i$-stars while ensuring independence. Since $k$ is a constant, and all since greedily adding clauses can be done in linear time, the total time taken to construct the families is $O(m+n)$. An example of $\mathcal{M}_4$ can be seen in Figure 1. We describe the structural properties of the families later in the proof.

Analogously to the warm-up algorithm in the previous section we describe two different algorithms that both make use of the independent families we have constructed and that complement each other in terms of their running times. The second algorithm describes in fact $k$ different algorithms, determined by the choice of a parameter $\ell \in \{1, \ldots, k\}$. For each $i = 1, \ldots, k$ we let $s_i$ denote the number of monotone $i$-stars in $\mathcal{M}_k$. By construction the parameter $r_i := \sum_{j=i}^{k} s_j$ then denotes the number of monotone $i$-stars in $M_i$. We further let $t$ be the number of triangles and $q$ be the number of non-monotone 2-stars in $\mathcal{M}_k$, and therefore in every $\mathcal{M}_i$ with $i = 1, \ldots, k$. The two algorithms we consider are:

1. Sample uniformly among all full assignments for $F$ that satisfy all the clauses in $\mathcal{M}_k$ until finding one that satisfies $F$.

[1] This is equivalent to finding a 1-maximal matching in a graph: first find a maximal matching and then find a maximal set of independent augmenting paths of length 3 and augment them.

2. Fix $\ell \in \{1, \dots k\}$. Define further the variable set $W := \mathrm{vbl}(\mathcal{M}_\ell)$ and let $W' \subseteq W$ be the set of variables of $\mathcal{M}_\ell$ that appear in a clause of $F$ that has exactly one variable of $\mathcal{M}_\ell$ in them. Go through all $2^{|W'|}$ partial assignments $\alpha : W' \to \{0,1\}$ and compute $A_\alpha := |\mathrm{sat}_{V \setminus W'}(F^{[\alpha]})|$ by using Wahlström's #2-SAT algorithm [18]. Let $A := \sum_\alpha A_\alpha$ and choose one partial assignment $\alpha^*$ at random so that $\Pr(\alpha^* = \alpha) = A_\alpha / A$. For the remaining variables choose an assignment $\beta^* : V \setminus W' \to \{0,1\}$ uniformly among all assignments satisfying $F^{[\alpha^*]}$. This can be done by branching on a variable, using Wahlström's algorithm to count the number of assignments in the two branches, flipping a biased coin weighed by the counts to decide on the branch and repeating the same on the resulting formula until all variables have been set. Output the full assignment which when restricted to $W'$ is $\alpha^*$ and when restricted to $V \setminus W'$ is $\beta^*$.

The correctness analysis for both of these two algorithms is essentially the same as in our warm-up in Section 3.1 and it remains to discuss the running times.

Starting with the first algorithm we note that the stars and triangles in $\mathcal{M}_k$ have constant size so the sampling of an assignment can be done in linear time in each iteration. Out of the $2^{i+1}$ possible assignments to the variables in any monotone $i$-star it can be easily checked that $2^i + 1$ satisfy all the clauses in the star. Both for a triangle or for a non-monotone 2-star there are 8 possible assignments out of which at most 4 are satisfying. Therefore from the independence of $\mathcal{M}_k$ we know that there are at most

$$2^{-t-q} \prod_{i=1}^{k} \left( \frac{2^i + 1}{2^{i+1}} \right)^{s_i} 2^n \tag{2}$$

full assignments to the variables in $F$ that satisfy everything in $\mathcal{M}_k$. Since $F$ has at least $\varepsilon 2^n$ satisfying assignments and the size of the universe we are sampling from is given by (2) we conclude that the first algorithm takes expected time

$$O\left( \varepsilon^{-1} 2^{-t-q} \prod_{i=1}^{k} \left( \frac{2^i + 1}{2^{i+1}} \right)^{s_i} (m+n) \right) \tag{3}$$

until returning a uniform satisfying assignment.

Consider now the runtime of the second algorithm. This is the more intricate part of the analysis and we will make use of the structure of the families that we have set up. It may be helpful to consider Figure 1. Let $F' \in \mathcal{M}_\ell$ be one of the subformulas in the family $\mathcal{M}_\ell$. We claim that $|\mathrm{vbl}(F') \cap W'| \leq 1$ and that if $\mathrm{vbl}(F') \cap W' = \{x\}$, then $F'$ is either an $\ell$-star or a non-monotone 2-star and $x$ is the center of the star. Towards showing the claim let $\{u, v\}$ be a clause with $\mathrm{vbl}(u) \in W$ and $\mathrm{vbl}(v) \in V \setminus W$ so that $\{u, v\}$ is a witness for $\mathrm{vbl}(u) \in W'$. If $\mathrm{vbl}(u)$ was a leaf of a star of $\mathcal{M}_\ell$, then we could have made $\mathcal{M}_0$ larger which would contradict the 1-maximality when $\mathrm{vbl}(u) \in \mathrm{vbl}(\mathcal{M}_0)$ or just maximality in the case of $\mathrm{vbl}(u) \notin \mathrm{vbl}(\mathcal{M}_0)$. For the same reasons the variable $\mathrm{vbl}(u)$ can not appear in any triangle. For any $j < \ell$ the variable $\mathrm{vbl}(u)$ can also not be the center of a $j$-star as otherwise we would have updated that star into a monotone $(j+1)$-star when constructing $\mathcal{M}_{j+1}$ or we would have created a non-monotone 2-star already in the beginning while constructing $\mathcal{M}_1$. The options for $\mathrm{vbl}(u)$ that remain are the centers of $\ell$-stars and the centers of the non-monotone 2-stars. In the case of $\ell = 1$ we still have to argue that at most one center may appear in $W'$. If both of the centers appeared in $W'$, it would either violate the 1-maximality of $\mathcal{M}_0$ or we could have turned the 1-star into a triangle which proves the claim. Therefore we have the bound $|W'| \leq r_\ell + q$.

We can observe from the argumentation above that if $\alpha : W' \to \{0,1\}$ is a partial assignment for $F$, then doing unit clause reduction on the formula $F^{[\alpha]}$ results in a 2-CNF formula over some variable set $W_\alpha \subseteq W \setminus W'$. Computing $A_\alpha$ with Wahlström's algorithm takes time $O(c^{|W_\alpha|})$ [18]. Therefore we want to bound $|W_\alpha|$ as tightly as possible. If the assignment $\alpha$ sets the center literal of a monotone $\ell$-star to 0, then the values of the $\ell$ remaining variables in the star are determined and will be set to their required values with unit clause reduction. For a non-monotone 2-star either assignment of the center will force the value of one of the leaves and one leaf stays undetermined. If $\alpha$ sets $i$ of the $r_\ell$ literals in the centers of the monotone $\ell$-stars to 0 we get the bound

$$|W_\alpha| \leq q + 3t + \ell(r_\ell - i) + \sum_{j=1}^{\ell-1}(j+1)s_j. \tag{4}$$

Among the assignments $\alpha$ that we consider there are $\binom{r_\ell}{i}2^q$ different ones that set $i$ of the central literals of the monotone $\ell$-stars to 0. Using formula (4) we conclude that the runtime cost of going over the assignments $\alpha$ and computing the numbers $A_\alpha$ is

$$\begin{aligned} &O\left(\sum_{i=0}^{r_\ell}\binom{r_\ell}{i}2^q \cdot c^{q+3t+\ell(r_\ell-i)+\sum_{j=1}^{\ell-1}(j+1)s_j} \cdot (m+n)\right) \\ &= O\left(c^{3t}(2c)^q\left(1+c^\ell\right)^{r_\ell}\left[\prod_{j=1}^{\ell-1}c^{(j+1)s_j}\right]\cdot(m+n)\right) \end{aligned} \tag{5}$$

where we used the binomial theorem. We can again use the same trick as in the warm-up algorithm to sample $\alpha^*$ without storing all the values of $A_\alpha$ to keep the space requirement linear. The running time of finding $\beta^*$ with the branching procedure takes time $O(c^{|W_{\alpha^*}|}|W_{\alpha^*}| + (m+n))$ which is subsumed by (5).

We have now one algorithm with running time given by (3) and for any $\ell \in \{1,\ldots,k\}$ we have an algorithm with running time given by (5). Given the sequence $(\mathcal{M}_1,\ldots,\mathcal{M}_k)$ we choose the algorithm with the best runtime. To find a worst case upper bound on the runtime we look for the runtime in the form

$$O\left(\varepsilon^{-\delta}(m+n)\right) \tag{6}$$

and compute the nonnegative parameters $s_1,\ldots,s_k;t$ and $q$ that maximize the minimum of the different runtimes. Write $\sigma_i := s_i/\log\frac{1}{\varepsilon}, \tau := t/\log\frac{1}{\varepsilon}, \rho := q/\log\frac{1}{\varepsilon}$. By taking logarithms of the runtimes (3), (5) and (6) we can write the problem of finding $\delta$ and the worst case parameters $\sigma_i, \tau, \rho$ as the linear program

$$\begin{array}{lrcl} & \max\limits_{\delta,\sigma_i,\tau,\rho} \quad \delta & & \\ \text{s.t.} \quad -\tau-\rho+\sum_{i=1}^{k}\sigma_i\log\left(\frac{2^i+1}{2^{i+1}}\right) & \geq & \delta-1 & \\ 3\tau\log c+\rho\log(2c)+\sum_{i=1}^{\ell-1}\sigma_i\log\left(c^{i+1}\right)+\sum_{i=\ell}^{k}\sigma_i\log\left(1+c^\ell\right) & \geq & \delta & \text{for all } \ell=1,\ldots,k \\ \delta,\sigma_i,\tau,\rho & \geq & 0 & \text{for all } i=1,\ldots,k\ . \end{array}$$

It turns out that we only need to consider $k = 7$ due to the fact that $c^{j+1} > 1 + c^j$ in the integers when $j \geq 7$ which implies that the running time for higher values of $k$ no longer improves. For $k = 7$ the linear program has in the optimum $\delta < 0.61618$. The approximate values of the other variables in the optimum are $\sigma_1 \approx 0.131, \sigma_2 \approx 0.127, , \sigma_3 \approx 0.111, \sigma_4 \approx 0.084, \sigma_5 \approx 0.051, \sigma_6 \approx 0.022, \sigma_7 \approx 0.004$ and exact values of $\tau = 0$ and $\rho = 0$. This finishes the proof. ◀

We attempted to improve the analysis by constructing families that do not consist only of stars and triangles but the runtimes we achieved were not better. In some sense stars seem particularly good for the efficient use of Wahlström's #2-SAT algorithm as a subroutine because the set $W'$ is not too big. We also note that while we could consider adding the option of choosing $\ell = 0$ in the second algorithm, it is easily verified that choosing $\ell = 1$ instead gives a better performance.

## 3.3 A randomized algorithm for 3-SAT

One could say that our Sample-2-SAT algorithm works because counting and sampling solutions for a $(\leq 1)$-CNF is trivial. Direct generalizations of our method to Sample-3-SAT do not work because the same is not true for $(\leq 2)$-CNF formulas. Instead of solving Sample-3-SAT we apply our method for 3-SAT.

▶ **Proposition 8.** *Let $F$ be a 3-CNF formula on $n$ variables and $m$ clauses and let $\varepsilon > 0$ be such that $|\mathrm{sat}(F)| \geq \varepsilon 2^n$. A satisfying assignment for $F$ can be found in expected time $O\left(\varepsilon^{-\log_8 7}(m+n)\right)$.*

**Proof.** Let $S$ be a maximal set of variable disjoint clauses in $F$. Either sample among those assignments that satisfy $S$ until finding a satisfying assignment or go through all the $7^{|S|}$ partial assignments to $\mathrm{vbl}(S)$ and check the satisfiability of the resulting $(\leq 2)$-CNF.

Checking through the partial assignments takes time $O(7^{|S|} \cdot (m+n))$ because each of the $7^{|S|}$ instances of $(\leq 2)$-SAT can be solved in linear time [1]. The rejections sampling takes expected time $O\left(\varepsilon^{-1}\left(\frac{7}{8}\right)^{|S|}(m+n)\right)$ because we are sampling from a pool of $\left(\frac{7}{8}\right)^{|S|}2^n$ assignments that contain all the at least $\varepsilon 2^n$ many satisfying assignments. Choosing always the better of the two methods, depending on $|S|$, gives a worst case running time of $O\left(\varepsilon^{-\log_8 7}(m+n)\right)$. ◀

Proposition 8 gives an algorithm that works for any $\varepsilon$, but there exist better algorithms for certain ranges of $\varepsilon$. The PPSZ algorithm for 3-SAT runs in expected time $O(1.308^n)$ [5] which is faster in the case that $\varepsilon = O(0.750^n)$. It is also possible to analyze Schöning's algorithm [15] for 3-SAT to get a dependence on $\varepsilon$ by using an isoperimetric inequality for the hypercube by Frankl and Füredi [4]. The runtime guarantee that results is $O\left(\left(\frac{4}{3} \cdot 2^{-H^{-1}(\delta)}\right)^n\right)$ in expectation where $\delta$ is the solution to $\varepsilon = 2^{(\delta-1)n}$ and where $H : (0, 1/2] \to (0, 1]$ is the bijective *binary entropy function* defined by $H(x) = -x\log_2(x) - (1-x)\log_2(1-x)$. We will include a proof in the full version of this paper [2]. The range where Schöning's algorithm is better than Proposition 8 is when $\varepsilon = O(0.929^n)$.

# 4 Conclusion

An interesting open problem is whether Sample-3-SAT can be solved time $O^*\left(\varepsilon^{-\delta}\right)$ for some $\delta < 1$. Similarly, can we achieve such a running time for 3-SAT with a deterministic algorithm?

We also believe that parameterizing by the number of solutions should be a fruitful approach to other problems besides satisfiability or vertex cover.

**Acknowledgments.** We would like to thank Noga Alon and József Solymosi for discussions on the problem. We also thank the reviewers of IPEC 2017 for valuable remarks that improved the exposition.

## References


1. Bengt Aspvall, Michael F Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.
2. Jean Cardinal, Jerri Nummenpalo, and Emo Welzl. Solving and Sampling with Many Solutions: Satisfiability and Other Hard Problems. *ArXiv e-prints*, 2017. `arXiv:1708.01122`.
3. Anindya De, Omid Etesami, Luca Trevisan, and Madhur Tulsiani. Improved pseudorandom generators for depth 2 circuits. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX, and 14th International Workshop, RANDOM*, pages 504–517, 2010.
4. Peter Frankl and Zoltán Füredi. A short proof for a theorem of Harper about hamming-spheres. *Discrete Mathematics*, 34(3):311–313, 1981.
5. Timon Hertli. 3-SAT faster and simpler—unique-SAT bounds for PPSZ hold in general. *SIAM Journal on Computing*, 43(2):718–729, 2014.
6. Edward A Hirsch. A fast deterministic algorithm for formulas that have many satisfying assignments. *Logic Journal of IGPL*, 6(1):59–71, 1998.
7. Mark R Jerrum. *Counting, sampling and integrating: algorithms and complexity*. Springer Science & Business Media, 2003.
8. Mark R Jerrum, Leslie G Valiant, and Vijay V Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.
9. Daniel M Kane and Osamu Watanabe. A short implicant of cnfs with relatively many satisfying assignments. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 20, page 176, 2013.
10. Kuldeep S Meel, Moshe Y Vardi, Supratik Chakraborty, Daniel J Fremont, Sanjit A Seshia, Dror Fried, Alexander Ivrii, and Sharad Malik. Constrained sampling and counting: Universal hashing meets sat solving. In *AAAI Workshop: Beyond NP*, 2016.
11. Burkhard Monien and Ewald Speckenmeyer. Solving satisfiability in less than $2^n$ steps. *Discrete Applied Mathematics*, 10(3):287–295, 1985.
12. Yehuda Naveh, Michal Rimon, Itai Jaeger, Yoav Katz, Michael Vinov, Eitan s Marcu, and Gil Shurek. Constraint-based random stimuli generation for hardware verification. *AI magazine*, 28(3):13, 2007.
13. Tian Sang, Paul Beame, and Henry A Kautz. Performing bayesian inference by weighted model counting. In *AAAI*, volume 5, pages 475–481, 2005.
14. Manuel Schmitt and Rolf Wanka. Exploiting independent subformulas: A faster approximation scheme for #k-SAT. *Information Processing Letters*, 113(9):337–344, 2013.
15. Uwe Schöning. A probabilistic algorithm for k-SAT based on limited local search and restart. *Algorithmica*, 32(4):615–623, 2002.
16. Rocco Servedio and Li-Yang Tan. Deterministic search for CNF satisfying assignments in almost polynomial time. Unpublished manuscript, 2016.
17. Luca Trevisan. A note on approximate counting for k-DNF. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 417–425. Springer, 2004.
18. Magnus Wahlström. A tighter bound for counting max-weight solutions to 2SAT instances. In *International Workshop on Parameterized and Exact Computation*, pages 202–213. Springer, 2008.

# Odd Multiway Cut in Directed Acyclic Graphs*

**Karthekeyan Chandrasekaran[1] and Sahand Mozaffari[2]**

1 **University of Illinois, Urbana-Champaign, USA**
karthe@illinois.edu
2 **University of Illinois, Urbana-Champaign, USA**
sahandm2@illinois.edu

## Abstract

We investigate the odd multiway node (edge) cut problem where the input is a graph with a specified collection of terminal nodes and the goal is to find a smallest subset of non-terminal nodes (edges) to delete so that the terminal nodes do not have an odd length path between them. In an earlier work, Lokshtanov and Ramanujan showed that both odd multiway node cut and odd multiway edge cut are fixed-parameter tractable (FPT) when parameterized by the size of the solution in *undirected graphs*. In this work, we focus on directed acyclic graphs (DAGs) and design a fixed-parameter algorithm. Our main contribution is an extension of the shadow-removal framework for *parity problems in DAGs*. We complement our FPT results with tight approximability as well as polyhedral results for 2 terminals in DAGs. Additionally, we show inapproximability results for odd multiway *edge* cut in undirected graphs even for 2 terminals.

**1998 ACM Subject Classification** G.2.2 Graph Theory, I.1.2 Algorithms

**Keywords and phrases** Odd Multiway Cut, Fixed-Parameter Tractability, Approximation Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.12

## 1 Introduction

In the classic $\{s,t\}$-cut problem, the goal is to delete the smallest number of edges so that the resulting graph has no path between $s$ and $t$. A natural generalization of this problem is the multiway cut, where the input is a graph with a specified set of terminal nodes and the goal is to delete the smallest number of non-terminal nodes/edges so that the terminals cannot reach each other in the resulting graph. In this work, we consider a parity variant of this problem. A path[1] is an odd-path (even-path) if the number of edges in the path is odd (even). In the OddMultiwayNodeCut (similarly, OddMultiwayEdgeCut), the input is a graph with a collection of terminal nodes and the goal is to delete the smallest number of non-terminal nodes (edges) so that the resulting graph has no odd-path between the terminals. This is a generalization of $\{s,t\}$-OddPathNodeBlocker (and similarly, $\{s,t\}$-OddPathEdgeBlocker), which is the problem of finding a minimum number of nodes that are disjoint from $s$ and $t$ (edges) that cover all $s-t$ odd-paths.

Covering and packing paths has been a topic of intensive investigation in graph theory as well as polyhedral theory. Menger's theorem gives a perfect duality relation for min

* A full version of the paper is available at https://arxiv.org/abs/1708.02323.
[1] We emphasize that the term *paths* refers to simple paths and not walks. This distinction is particularly important in parity-constrained settings, because the existence of a walk with an odd number of edges between two nodes $s$ and $t$ does not imply the existence of an odd-path between $s$ and $t$. This is in contrast to the non-parity-constrained settings where the existence of a walk between $s$ and $t$ implies the existence of a path between $s$ and $t$.


$\{s,t\}$-cut: the minimum number of nodes (edges) that cover all $s-t$ paths is equal to the maximum number of node-disjoint (edge-disjoint) $s-t$ paths. However, packing paths of restricted kinds is a difficult problem. One special case is when the paths are required to be of odd-length for which many deep results exist [4, 17, 5]. In this work, we study the problem of covering $s-t$ odd-paths and more generally all odd-paths between a given collection of terminals.

Covering $s-t$ odd-paths in undirected graphs has been explored in the literature from the perspective of polyhedral theory—we refer to Chapter 29 in Schrijver's book [17]. Given an undirected graph $G=(V,E)$ with distinct nodes $s,t \in V$ and non-negative edge-lengths, we may find a shortest length $s-t$ odd-path in polynomial time. Edmonds gave a polynomial-time algorithm for this by reducing the shortest $s-t$ odd-path problem to the minimum-weight perfect matching problem [13, 7, 8]. However, as observed by Schrijver and Seymour [18], his approach of reducing to a matching problem does not extend to address other fundamental problems about $s-t$ odd-paths. One such fundamental problem is the $\{s,t\}$-OddPathEdgeBlocker problem. Towards investigating $\{s,t\}$-OddPathEdgeBlocker, Schrijver and Seymour [18] considered the following polyhedron:

$$\mathcal{P}^{\text{odd-cover}} := \left\{ x \in \mathbb{R}^E_+ : \sum_{e \in P} x_e \geq 1 \ \forall \ s-t \text{ odd-path } P \text{ in } G \right\}.$$

This leads to a natural integer programming formulation of $\{s,t\}$-OddPathEdgeBlocker: $\min \left\{ \sum_{e \in E} x_e : x \in \mathcal{P}^{\text{odd-cover}} \cap \mathbb{Z}^E \right\}$. By Edmonds' algorithm, we have an efficient separation oracle for $\mathcal{P}^{\text{odd-cover}}$ and hence there exists an efficient algorithm to optimize over $\mathcal{P}^{\text{odd-cover}}$ using the Ellipsoid algorithm [10]. It was known that the extreme points of $\mathcal{P}^{\text{odd-cover}}$ are not integral. Cook and Sebö conjectured that all extreme points of $\mathcal{P}^{\text{odd-cover}}$ are half-integral which was later shown by Schrijver and Seymour [18]. Schrijver and Seymour's work also gave a min-max relation for the max fractional packing of $s-t$ odd-paths. However their work does not provide algorithms or address the computational complexity of $\{s,t\}$-OddPathEdgeBlocker. In this work, we show NP-hardness and an inapproximability result for $\{s,t\}$-OddPathEdgeBlocker in undirected graphs.

The main focus of this work is OddMultiwayNodeCut in directed acyclic graphs (DAGs). Before describing the reason for focusing on the subfamily of DAGs among directed graphs, we mention that OddMultiwayNodeCut and OddMultiwayEdgeCut are equivalent in directed graphs by standard reductions. The reason we focus on the subfamily of DAGs and not all directed graphs is the following: consider the $(s \to t)$-OddPathEdgeBlocker problem where the input is a directed graph with nodes $s,t$ and the goal is to find a minimum number of edges to delete so that the resulting graph has no odd-path from $s$ to $t$. There is a stark contrast in the complexity of $\{s,t\}$-OddPathEdgeBlocker in undirected graphs and $(s \to t)$-OddPathEdgeBlocker in directed graphs: while there exists a polynomial time algorithm to verify if a given undirected graph has an $s-t$ odd-path (e.g., by Edmonds' reduction to a matching problem), it is NP-complete to verify if a given directed graph has an $s \to t$ odd-path (e.g., see Lapaugh-Papadimitriou [13]). Thus verifying feasibility of a solution to OddMultiwayEdgeCut is already NP-complete in directed graphs. However, there exists a polynomial time algorithm to verify if a given directed *acyclic* graph (DAG) has an $s \to t$ odd-path. For this reason, we restrict our focus to DAGs.

Our main contribution is a fixed-parameter algorithm for OddMultiwayNodeCut in DAGs. We complement the fixed-parameter algorithm by showing NP-hardness and tight approximability results for the two terminal variant, namely $(s \to t)$-OddPathNodeBlocker, in DAGs.

In addition to approximation algorithms, fixed-parameter algorithms have served as an alternative approach to address NP-hard problems [9]. A problem is said to be fixed-parameter tractable (FPT), if it can be solved in time $f(k)n^c$, where $k$ is the parameter, $f$ is a computable function, $n$ is the size of the input and $c$ is a universal constant. Fixed-parameter algorithms for cut problems have provided novel insights into the connectivity structure of graphs [6]. The notion of *important separators* and the *shadow-removal technique* have served as the main ingredients in the design of fixed-parameter algorithms for numerous cut problems [6]. Our work also builds upon the *shadow-removal technique* to design fixed-parameter algorithms but differs from known applications substantially owing to the parity constraint.

**Related Work.** We are not aware of any prior work on this problem in directed graphs. We describe the known results in undirected graphs. A simple reduction from vertex cover[2] shows that $\{s,t\}$-OddPathNodeBlocker in undirected graphs is NP-hard and does not admit a $(2-\epsilon)$-approximation for $\epsilon > 0$ assuming the Unique Games Conjecture [11]. These hardness results also hold for OddMultiwayNodeCut. The most relevant results to this work are that of Lokshtanov and Ramanujan [15, 16]. They showed a parameter-preserving reduction from OddMultiwayEdgeCut to OddMultiwayNodeCut and designed a fixed-parameter algorithm for OddMultiwayNodeCut. However, their algorithmic techniques work only for undirected graphs and do not extend immediately for OddMultiwayNodeCut in directed acyclic graphs.

Lokshtanov and Ramanujan also showed that OddMultiwayEdgeCut is NP-hard in undirected graphs for three terminals. However, their reduction is not an approximation-preserving reduction. Hence the approximability of OddMultiwayEdgeCut in undirected graphs merits careful investigation. In particular, the complexity of OddMultiwayEdgeCut in undirected graphs even for the case of two terminals is open in spite of existing polyhedral work in the literature [18] for this problem.

## 1.1 Results

**Directed acyclic graphs.** We recall that OddMultiwayNodeCut and OddMultiwayEdgeCut are equivalent in DAGs by standard reductions. Hence, all of the following results for DAGs hold for both problems. The following is our main result.

▶ **Theorem 1.** OddMultiwayNodeCut *in DAGs can be solved in* $2^{O(k^2)}poly(n)$ *time, where* $k$ *is the size of the optimal solution and* $n$ *is the number of nodes in the input graph.*

We briefly remark on the known techniques to illustrate the challenges in designing the fixed-parameter algorithm for OddMultiwayNodeCut in DAGs. To highlight the challenges, we will focus on the case of 2 terminals, namely $(s \to t)$-OddPathNodeBlocker in DAGs.

**Remark 1.** It is tempting to design a fixed-parameter algorithm for $(s \to t)$-OddPathNodeBlocker by suitably modifying the definition of *important separators* to account for parity and using the shadow-removal technique for directed graphs [3]. However, the main technical challenge lies in understanding and exploiting the acyclic property of the input directed graph.

[2] Given an instance $G$ of vertex cover, introduce two new nodes $s$ and $t$ that are adjacent to all nodes in $G$ to obtain a graph $H$. A set $S \subseteq V(G)$ is a feasible vertex cover in $G$ if and only if $S$ is a feasible solution to $\{s,t\}$-OddPathNodeBlocker in $H$.

**Remark 2.** The next natural attempt is to rely on the fixed-parameter algorithm for multicut in DAGs by Kratsch et al. [12]. However, their technique crucially relies on reducing the degrees of the source terminals by suitably branching to create a small number of instances. On the one hand, applying their branching rule directly to reduce the degree of $s$ in $(s \to t)$-OddPathNodeBlocker will blow up the number of instances in the branching. On the other hand, it is unclear how to modify their branching rule to account for parity.

Given the difficulties mentioned in the above two remarks, our algorithm builds upon the shadow-removal technique and exploits the acyclic property of the input directed graph to reduce the instance to minimum odd cycle transversal (remove the smallest number of nodes to make an undirected graph bipartite) which in turn, has a fixed-parameter algorithm when parameterized by the number of removed nodes. Our technique is yet another illustration of the broad-applicability of the shadow-removal framework.

We complement our fixed-parameter algorithm in Theorem 1 with tight approximability results for 2 terminals. We refer the reader to Table 1 for a summary of the complexity and approximability results. Unlike the case of undirected graphs where there is still a gap in the approximability of both $\{s,t\}$-OddPathEdgeBlocker and $\{s,t\}$-OddPathNodeBlocker, we present tight approximability results for both $(s \to t)$-OddPathEdgeBlocker and $(s \to t)$-OddPathNodeBlocker.

▶ **Theorem 2.** *We have the following inapproximability and approximability results:*

**(i)** $(s \to t)$-OddPathNodeBlocker *in DAGs is NP-hard, and has no efficient* $(2-\epsilon)$*-approximation for any* $\epsilon > 0$ *assuming the Unique Games Conjecture.*

**(ii)** *There exists an efficient* 2*-approximation algorithm for* $(s \to t)$-OddPathNodeBlocker *in DAGs.*

We emphasize that our 2-approximation for $(s \to t)$-OddPathEdgeBlocker mentioned in Theorem 2 is a combinatorial algorithm and not LP-based. We note that Schrijver and Seymour's result [18] that all extreme points of $\mathcal{P}^{\text{odd-cover}}$ are half-integral holds only in undirected graphs and fails in DAGs—see Theorem 3 below. Consequently, we are unable to design a 2-approximation algorithm using the extreme point structure of the natural LP-relaxation of the path-blocking integer program. Instead, our approximation algorithm is combinatorial in nature. The correctness argument of our algorithm also shows that the integrality gap of the LP-relaxation of the path-blocking integer program is at most 2 in DAGs.

▶ **Theorem 3.** *The following odd path cover polyhedron is not necessarily half-integral:* $\mathcal{P}^{\text{odd-cover-dir}} := \left\{x \in \mathbb{R}_+^E : \sum_{e \in P} x_e \geq 1 \ \forall \ s \to t \text{ odd-path } P \text{ in } D\right\}$.

**Undirected graphs.** We next turn our attention to undirected graphs. As mentioned earlier, $\{s,t\}$-OddPathNodeBlocker is NP-hard and does not admit a $(2-\epsilon)$-approximation assuming the Unique Games Conjecture. We are unaware of a constant factor approximation for $\{s,t\}$-OddPathNodeBlocker. For $\{s,t\}$-OddPathEdgeBlocker, the results of Schrijver and Seymour [18] show that the LP-relaxation of a natural integer programming formulation of $\{s,t\}$-OddPathEdgeBlocker is half-integral and thus leads to an efficient 2-approximation for $\{s,t\}$-OddPathEdgeBlocker. However, the complexity of $\{s,t\}$-OddPathEdgeBlocker was open. We address this gap in complexity by showing the following NP-hardness and inapproximability results.

▶ **Theorem 4.** $\{s,t\}$-OddPathEdgeBlocker *is NP-hard and has no efficient* $(6/5-\epsilon)$*-approximation assuming the Unique Games Conjecture.*

**Table 1** Complexity and Approximability. Text in gray refers to known results while text in black refers to the results from this work.

| Problem | Undirected graphs | DAGs |
|---|---|---|
| $\{s,t\}$-OddPathNodeBlocker | $(2-\epsilon)$-inapprox | [Equiv. to edge-deletion] |
| $\{s,t\}$-OddPathEdgeBlocker | LP is half-integral [18]<br>2-approx [18]<br>$(\frac{6}{5}-\epsilon)$-inapprox (Thm 4) | LP is NOT half-integral (Thm 3)<br>2-approx (Thm 2)<br>$(2-\epsilon)$-inapprox (Thm 2) |
| OddMultiwayEdgeCut | NP-hard for 3 terminals [15]<br>$(\frac{6}{5}-\epsilon)$-inapprox for 2 terminals (Thm 4) | |

**Organization.** We summarize the preliminaries in Section 1.2. We prove the FPT for DAGs (Theorem 1) in Section 2. We refer the reader to the full version of the paper [1] for all missing proofs.

## 1.2 Preliminaries

For ease of notation, we will frequently use $v$ instead of $\{v\}$. Let $G$ be a (directed) graph and $W$ be a subset of $V(G)$. A $W$-path in $G$ is a path with both of its end-nodes in $W$. We restate the problem of OddMultiwayNodeCut in DAGs to set the notation.

▶ **Problem 5** (Minimum Odd Multiway Cut in DAGs). *Given a directed acyclic graph $G=(V,E)$ with sets $T$, $V^\infty \subseteq V$ where $T \subseteq V^\infty$, an* odd multiway cut *in $G$ is a set $M \subseteq V(G) \setminus V^\infty$ of nodes that intersects every odd $T$-path in $G$. We refer to $T$ as terminals, $V \setminus T$ as non-terminals and $V^\infty$ as protected nodes. In* DagOddMultiwayNodeCut*, the input is specified as $(G, V^\infty, T, k)$, where $k \in \mathbb{Z}_+$ and the goal is to verify if there exists an odd multiway cut in $G$ of size at most $k$.*

For subsets $X$ and $Y$ of $V(G)$ we say that $M \subseteq V(G) \setminus V^\infty$ is an $X \to Y$ *separator* in $G$ when $G \setminus M$ has no path from $X$ to $Y$. The set of nodes that can be reached from a node set $X$ in $G$ is denoted by $\mathcal{R}_G(X)$. We note that $\mathcal{R}_G(X)$ always includes $X$. We define the *forward shadow* of a node set $M$ to be $f_G(M) := V(G \setminus M) \setminus \mathcal{R}_{G\setminus M}(T)$, i.e., the set of nodes $v$ such that there is no $T \to v$ path in $G$ disjoint from $M$. Similarly, the *reverse shadow* of $M$, denoted $r_G(M)$, is the set of nodes $v$ from which there is no path to $T$ in $G \setminus M$. Equivalently, the reverse shadow is $f_{G^{\text{rev}}}(M)$, where $G^{\text{rev}}$ is the graph obtained from $G$ by reversing all the edge orientations. We refer to the union of the forward and the reverse shadow of $M$ in $G$, as *shadow* of $M$ in $G$ and denote it by $s_G(M)$. An $X \to Y$ separator $M'$ is said to *dominate* another $X \to Y$ separator $M$, if $|M'| \leq |M|$ and $\mathcal{R}_{G\setminus M}(X) \subsetneq \mathcal{R}_{G\setminus M'}(X)$. A minimal $X \to Y$ separator that is not dominated by any other separator is called an *important $X \to Y$ separator*. A set $M \subseteq V(G)$ is *thin*, if every node $v \in M$ is not in $r_G(M \setminus \{v\})$.

For a directed graph $G$, we define the *underlying undirected graph* of $G$, denoted by $G^*$, as the undirected graph obtained from $G$ by dropping the orientations. In an undirected graph $H$ with protected nodes $V^\infty$, an *odd cycle transversal* is a set $U \subseteq V(H) \setminus V^\infty$ of nodes such that $H \setminus U$ is bipartite. The problem of finding the minimum such set in a given instance of the problem is called the *minimum odd cycle transversal problem* and is denoted by MinOddCycleTransversal. Although this problem is NP-hard, it is fixed-parameter tractable when parameterized by the size of the solution. The current-best fixed-parameter algorithm

for MinOddCycleTransversal runs in time $O(2.32^k \text{poly}(n))$ [14]. The problem addressed in [14] does not allow for protected nodes, but MinOddCycleTransversal with protected nodes, can easily be reduced to MinOddCycleTransversal without protected nodes by iteratively replacing each protected node with $k+1$ nodes and connecting them to the same set of neighbors as the original node. We will use MinOddCycleTransversal$(H, V^\infty, k)$ to denote the procedure that implements this fixed-parameter algorithm for the input graph $H$ with protected nodes $V^\infty$ and parameter $k$.

## 2 FPT of OddMultiwayNodeCut in DAGs

We will use the shadow-removal technique introduced in [3]. We will reduce the problem to MinOddCycleTransversal problem in an undirected graph, which is a fixed-parameter tractable problem when parameterized by the solution size.

### 2.1 Easy instances

▶ **Theorem 6.** *Suppose the instance $(G, V^\infty, T, k)$ of* DagOddMultiwayNodeCut *has a solution $M$ of size at most $k$ with the following property: every node $v \in s_G(M)$ has total degree at most one in $G \setminus M$. There exists an algorithm that given one such instance $(G, V^\infty, T, k)$ as input, finds a solution of size at most $k$ in time $O(2.32^k \text{poly}(n))$, where $n$ is the number of nodes in the input graph $G$.*

**Proof.** Let $(G, V^\infty, T, k)$ be an instance of DagOddMultiwayNodeCut. We recall that $G^*$ denotes the undirected graph obtained by dropping the orientations of the edges in $G$. We show the following equivalence: a set $M \subseteq V \setminus V^\infty$ with the property as in the statement is a solution if and only if $G^* \setminus M$ is bipartite with a bipartition $(A, B)$ such that $T \subseteq A$.

Suppose $G^* \setminus M$ is bipartite with a bipartition $(A, B)$ such that $T \subseteq A$. In a bipartite graph, every two end-nodes of any odd path are necessarily in different parts. Hence, there is no odd $T$-path in $G^* \setminus M$. Thus, there is no odd $T$-path in $G \setminus M$. Hence, $M$ is a solution for the odd multiway cut instance $(G, V^\infty, T, k)$.

Suppose the solution $M$ has the property mentioned in the statement of the theorem. Let $U := V(G \setminus M) \setminus s_G(M)$. Define $A := \{x \in U : \text{there is an even } T \to x \text{ path in } G \setminus M\}$ and $B := \{x \in U : \text{there is an odd } T \to x \text{ path in } G \setminus M\}$. It follows from the definition of the shadow that every node in $U$ has a path $P_1$ from $T$ in $G \setminus M$. Therefore, every node of $U$ is in $A \cup B$. Also by definition, every node $v$ in $U$ has a path $P_2$ to $T$ in $G \setminus M$. The parity of every $T \to v$ path has to be the same as the parity of $P_2$, because the concatenation of a $T \to v$ path and a $v \to T$ path in $G \setminus M$ is a $T$-path in $G \setminus M$ and therefore must be even. We note that such a concatenation cannot be a cycle since $G$ is acyclic. Thus, no node of $U$ is in both $A$ and $B$. Hence, $(A, B)$ is a partition of $U$.

We observe that there cannot be an edge from a node $v$ in $A$ to a node $u$ in $A$, as otherwise the concatenation of the even $T \to v$ path $Q_1$ with the edge $v \to u$ is an odd $T \to u$ path in $G \setminus M$ which means $u \in B$. This contradicts our conclusion about $A$ and $B$ being disjoint. By a similar argument, there is no edge between any pair of nodes in $B$. Thus, the subgraph of $G$ induced by $A$ and $B$ are independent sets respectively. Hence $G^*[A \cup B]$ is a bipartite graph. Furthermore, $(A, B)$ is a bipartition of $G^*[A \cup B]$ with every node of $T$ in $A$. By assumption, the degree of every node $x \in s_G(M)$ is at most one. Therefore, $x$ has neighbors in at most one of $A$ and $B$. Thus, we can extend the bipartition $(A, B)$ of $G^*[A \cup B]$ to a bipartition $(A', B')$ of $G^* \setminus M$ as follows: denote $H := G^*[A \cup B]$; repeatedly pick a node

**Algorithm 1** SolveEasyInstance

Given: DAG $G$ with terminal set $T$, a set $V^\infty$ of protected nodes containing $T$, $k \in \mathbb{Z}_+$, where $(G, V^\infty, T, k)$ has the property specified in the statement of Theorem 6

1: $G_1 \leftarrow$ the underlying undirected graph of $G$.
2: Let $G_2$ be the graph obtained from $G_1$ by introducing a new node $x$ and connecting it to every node in $T$.
3: $N \leftarrow \text{MinOddCycleTransversal}(G_2, V^\infty \cup \{x\}, k)$
4: **return** $N$

$x \in s_G(M) \setminus V(H)$ with a neighbor in $H$, include $x$ in a part ($A$ or $B$) in which $x$ has no neighbor and update $A$, $B$ and $H$.

Hence, if the given instance has a solution $M$ of size at most $k$ such that every node $v \in s_G(M)$ has total degree at most one, then such a solution can be found by the fixed-parameter algorithm for MinOddCycleTransversal. To ensure that the terminal nodes will be in the same part, we introduce a new protected node into the graph and connect it to every terminal node. This approach is described in Algorithm 1. All steps in Algorithm 1 can be implemented to run in polynomial time except Step 3. The running time of Step 3 is $O(2.32^k \text{poly}(n))$ [14]. ◀

We will use the name SolveEasyInstance to refer to the algorithm of Theorem 6. Our aim now is to reduce the given arbitrary instance $(G, V^\infty, T, k)$ to another instance that has a solution with the property mentioned in Theorem 6 or determine that no solution of size at most $k$ exists. We need the notion of parity-preserving torso operation on DAGs.

## 2.2 Parity-Preserving Torso

The parity preserving torso operation was introduced by Lokshtanov and Ramanujan [15] for undirected graphs. We extend it in a natural fashion for DAGs.

▶ **Definition 7** (Parity-Preserving Torso)**.** Let $G$ be a DAG and $Z$ be a subset of $V(G)$. We define Parity-Torso$(G, V^\infty, Z)$ as $(G', V'^\infty)$, where $G'$ is the DAG obtained from $G \setminus Z$ by adding an edge from node $u$ to $v$, for every pair of nodes $u, v \in V(G) \setminus Z$ such that there is an odd-path from $u$ to $v$ in $G$ all of whose internal nodes are in $Z$, and including a new node $x_{uv}$ and edges $u \to x_{uv}$ and $x_{uv} \to v$ for every pair of nodes $u, v \in V(G) \setminus Z$ such that there is an even path from $u$ to $v$ in $G$ all of whose internal nodes are in $Z$. The set $V'^\infty$ is defined to be the union of $V^\infty \setminus Z$ and all the new nodes $x_{uv}$.

We emphasize that the acyclic nature of the input directed graph allows us to implement the parity-preserving torso operation in polynomial time. Moreover, applying parity-preserving torso on a DAG results in a DAG as well. In what follows, we state the properties of the Parity-Torso operation that are exploited by our algorithm. The parity-preserving torso operation, has the property that it maintains $u \to v$ paths along with their parities between any pair of nodes $u, v \in V(G) \setminus Z$. More precisely:

▶ **Lemma 8.** *Let $G$ be a DAG and $Z, V^\infty \subseteq V(G)$. Define $(G', V'^\infty) :=$ Parity-Torso$(G, V^\infty, Z)$. Let $u, v$ be nodes in $V(G) \setminus Z$. There is a $u \to v$ path $P$ in $G$ if and only if there is a $u \to v$ path $Q$ of the same parity in $G'$. Moreover, the path $Q$ can be chosen so that the nodes of $P$ in $G \setminus Z$ are the same as the nodes of $Q$ in $G \setminus Z$, i.e. $V(P) \cap (V(G) \setminus Z) = V(Q) \cap (V(G) \setminus Z)$.*

**Algorithm 2** Minimum odd multiway cut in DAGs

Given: DAG $G$ with terminal set $T$, a set $V^\infty$ of protected nodes containing $T$, and $k \in \mathbb{Z}_+$

1: **for** $Z \in$ ShadowContainer$(G, V^\infty, k)$ **do**
2: $\quad (G_1, V_1^\infty) \leftarrow$ Parity-Torso$(G, V^\infty, Z)$
3: $\quad N \leftarrow$ SolveEasyInstance$(G_1, V_1^\infty, T, k)$
4: $\quad$ **if** $N$ is a solution in $G$ **then**
5: $\quad\quad$ **return** $N$
6: **return** No solution

▶ **Corollary 9.** *Let $\mathcal{I} = (G, V^\infty, T, k)$ be an instance of* DagOddMultiwayNodeCut *and let $Z \subseteq V(G) \setminus T$. Let $(G', V'^\infty) :=$ Parity-Torso$(G, V^\infty, Z)$ and denote the instance $(G', V'^\infty, T, k)$ by $\mathcal{I}'$. The instance $\mathcal{I}$ admits a solution $S$ of size at most $k$ that is disjoint from $Z$ if and only if the instance $\mathcal{I}'$ admits a solution of size at most $k$.*

Therefore, we are interested in finding a set $Z$ of nodes that is disjoint from some solution of size at most $k$, and moreover, the instance (Parity-Torso$(G, V^\infty, Z), T, k)$ satisfies the property mentioned in Theorem 6. The following lemma summarizes our key observation: it shows that it is sufficient to find a set $Z$ that contains the shadow of a solution.

▶ **Lemma 10.** *Let $G$ be a DAG and $M, Z, V^\infty \subseteq V(G)$. Suppose $M$ intersects every odd $T$-path in $G$ and $s_G(M) \subseteq Z \subseteq V(G) \setminus M$. Define $(G', V'^\infty) :=$ Parity-Torso$(G, V^\infty, Z)$. Then every node in $s_{G'}(M)$ has total degree at most one in $G' \setminus M$.*

## 2.3 Difficult instances

Corollary 9 and Lemma 10 show that if we find a set $Z$ such that for some solution $M$, the set $Z$ is disjoint from $M$ and contains the shadow of $M$ in $G$, then considering Parity-Torso$(G, V^\infty, Z)$ will give a new instance that satisfies the conditions of Theorem 6. Our goal now is to obtain such a set $Z$. We will show the following lemma. We emphasize that the lemma holds for arbitrary digraphs.

▶ **Lemma 11.** *There is an algorithm* ShadowContainer *that given an instance $(G, V^\infty, T, k)$ of* DagOddMultiwayNodeCut*, where $G$ is a digraph, returns a family $\mathcal{Z}$ of subsets of $V(G)$ with $|\mathcal{Z}| = 2^{O(k^2)}$, with the property that if the problem has a solution of size at most $k$, then for some solution $M$ of size at most $k$, there exists a set $Z \in \mathcal{Z}$ that is disjoint from $M$ and contains $s_G(M)$. Moreover, the algorithm can be implemented to run in time $2^{O(k^2)}\,\mathrm{poly}(|V(G)|)$.*

We defer the proof of Lemma 11 to first see its implications.

▶ **Theorem 12.** *There exists an algorithm that given an instance $(G, V^\infty, T, k)$ of* DagOddMultiwayNodeCut*, runs in $2^{O(k^2)}\,\mathrm{poly}(|V(G)|)$ time and either finds a solution of size at most $k$ or determines that no such solution exists.*

**Proof.** We use Algorithm 2. Let $(G, V^\infty, T, k)$ be an instance of DagOddMultiwayNodeCut, where $G$ is a DAG. Suppose there exists a solution of size at most $k$. By Lemma 11, the procedure ShadowContainer$(G, V^\infty, k)$ in Line 1 returns a family $\mathcal{Z}$ of subsets of $V(G)$ with $|\mathcal{Z}| = 2^{O(k^2)}$ containing a set $Z$ such that there is a solution $M$ of size at most $k$ that is disjoint from $Z$ and $Z$ contains $s_G(M)$. Let $(G_1, V_1^\infty)$ be the result of applying Parity-Torso operation to the set $Z$ in $G$ (i.e., the result of Step 2 in Algorithm 2). By Lemma 10, every node in $s_{G_1}(M)$ has total degree at most one in $G_1 \setminus M$. Therefore, by Theorem 6, the set

$N$ returned in Line 3 is a solution to the instance $(G_1, V_1^\infty, T, k)$. By Corollary 9, the set $N$ is also a solution to the original instance of the problem.

If there is no solution of size at most $k$, the algorithm will not find any. Therefore, the algorithm is correct. The runtime of the algorithm is dominated by Line 2 which can be implemented to run in $2^{O(k^2)}\text{poly}(|V(G)|)$ time by Lemma 11. ◀

In order to prove Lemma 11, we will use the following result.

▶ **Theorem 13** (Chitnis *et al.* [2]). *There is an algorithm that given a digraph $G$, a subset of protected nodes $V^\infty \subseteq V(G)$, terminal nodes $T \subseteq V^\infty$ and an integer $k$, returns a family $\mathcal{Z}$ of subsets of $V(G) \setminus V^\infty$ with $|\mathcal{Z}| = 2^{O(k^2)}$ such that for every $S, Y \subseteq V(G)$ satisfying*
**(i)** *$S$ is a thin set with $|S| \leq k$, and*
**(ii)** *for every $v \in Y$, there exists an important $v \to T$ separator contained in $S$,*
*there exists $Z \in \mathcal{Z}$ with $Y \subseteq Z \subseteq V(G) \setminus S$. Moreover, the algorithm can be implemented to run in time $2^{O(k^2)}\,\text{poly}(|V(G)|)$.*

To invoke Theorem 13, we need to guarantee that there exists a solution $S$ of size at most $k$ such that $S$ is thin and its reverse shadow $Y$ in $G$ has the property that for every $v \in Y$ there is an important $v \to T$ separator contained in $S$. Towards obtaining such a solution, we prove the following.

▶ **Lemma 14.** *Let $(G, V^\infty, T, k)$ be an instance of* DagOddMultiwayNodeCut*, where $G$ is a DAG. Let $M$ be a solution for this instance. If there exists $v \in r_G(M)$ such that $M$ does not contain an important $v \to T$ separator, then there exists another solution $M'$ of size at most $|M|$, such that $r_G(M) \cup f_G(M) \cup M \subseteq r_G(M') \cup f_G(M') \cup M'$, and $r_G(M) \subsetneq r_G(M')$.*

**Proof.** Let $M_0$ be the set of nodes $u \in M$ for which there is a $v \to u$ path in $G$ that is internally disjoint from $M$. Since $v \in r_G(M)$, every $v \to T$ path intersects $M$. For a $v \to T$ path $P$, the first node $u \in P \cap M$ is in $M_0$. Hence, every $v \to T$ path intersects $M_0$. Therefore, the set $M_0$ is a $v \to T$ separator in $G$. Therefore, it contains a minimal separator $M_1$. Since we assumed that there is no important $v \to T$ separator contained in $M$, the set $M_1$ is not an important $v \to T$ separator. Suppose $M_1$ is dominated by another $v \to T$ separator and let $M_2$ be an important $v \to T$ separator that dominates $M_1$. Define $M'$ as $(M \setminus M_1) \cup M_2$. We recall that a separator is by definition, disjoint from the protected node set. Therefore, $M' \cap V^\infty = \emptyset$. We will show that $M'$ contradicts the choice of $M$. We need the following claims.

▶ **Claim 15.** *$M \setminus M' \subseteq r_G(M')$.*

**Proof.** We observe that $M \setminus M' = M_1 \setminus M_2$. Let $u$ be an arbitrary node in $M_1 \setminus M_2$. Since $u \in M_1$ and $M_1$ is a minimal $v \to T$ separator, there is a $v \to u$ path $P_1$ that is internally disjoint from $M_1$. Since $M_2$ dominates $M_1$, therefore, $R_{G \setminus M_1}(v) \subseteq R_{G \setminus M_2}(v)$. Thus, $V(P_1) \subseteq R_{G \setminus M_2}(v)$. Hence, $P_1$ is disjoint from $M_2$. Suppose $P_2$ is an arbitrary $u \to T$ path in $G$. Concatenation of $P_1$ and $P_2$ is a $v \to T$ path in $G$ and therefore, has to intersect $M_2$. Since $P_1$ is disjoint from $M_2$, the path $P_2$ has to intersect $M_2$. Hence, every $u \to T$ path in $G$ intersects $M_2$ and in particular, intersects $M'$. Equivalently, $u \in r_G(M')$. ◀

We next show that $M'$ is a feasible solution for the problem and is no larger than $M$.

▶ **Claim 16.** *The set $M'$ intersects every odd $T$-path in $G$ and $|M'| \leq |M|$.*

**Proof.** By assumption, every odd $T$-path $P$ intersects $M$. If $P$ intersects $M \cap M'$, then it also intersects $M'$. If $P$ intersects $M \setminus M'$, then by Claim 15 it also intersects $M'$. Thus, every odd $T$-path in $G$ intersects $M'$. Furthermore, by definition of $M'$, we have

$$|M'| = |M| + (|M_2 \setminus M| - |M_1 \setminus M_2|) \leq |M| + (|M_2| - |M_1|) \leq |M|\,. \quad \blacktriangleleft$$

▶ **Claim 17.** *$r_G(M) \subseteq r_G(M')$.*

**Proof.** Let $u$ be an arbitrary node in $r_G(M)$. The set $M$ is a $u \to T$ separator. Therefore, every $u \to T$ path intersects $M$. We need to show that every $u \to T$ path also intersects $M'$. Let $P$ be a $u \to T$ path. If $P$ intersects $M \cap M'$, then it also intersects $M'$. If $P$ does not intersect $M \cap M'$, then it has to intersect $M \setminus M'$. By Claim 15, every $M \setminus M' \to T$ path intersects $M'$. Therefore, $u \in r_G(M')$. ◀

▶ **Claim 18.** *$r_G(M) \cup f_G(M) \cup M \subseteq r_G(M') \cup f_G(M') \cup M'$.*

**Proof.** By Claim 15, we have $M \setminus M' \subseteq r_G(M')$ and by Claim 17, we have $r_G(M) \subseteq r_G(M')$. Thus, it remains to prove that $f_G(M) \subseteq r_G(M') \cup f_G(M') \cup M'$. Let $u$ be an arbitrary node in $f_G(M) \setminus (r_G(M') \cup f_G(M') \cup M')$. Since $u \notin f_G(M')$, there is a $T \to u$ path $P_1$ in $G$ that is disjoint from $M'$. But $u \in f_G(M)$. Thus $P_1$ has to intersect $M$, particularly it has to intersect $M \setminus M'$. Let $P_2$ be a subpath of $P_1$ from $M \setminus M'$ to $u$. Since $u \notin r_G(M')$, there is a $u \to T$ path $P_3$ in $G$ that is disjoint from $M'$. The concatenation of $P_2$ and $P_3$ is a path from $M \setminus M'$ to $T$ that is disjoint from $M'$. But by Claim 15, every $M \setminus M' \to T$ path in $G$ must intersect $M'$. This contradiction shows that $f_G(M) \subseteq (r_G(M') \cup f_G(M') \cup M')$. ◀

▶ **Claim 19.** *$r_G(M) \subsetneq r_G(M')$.*

**Proof.** By Claim 17, $r_G(M) \subseteq r_G(M')$. We need to prove $r_G(M) \neq r_G(M')$. We recall that $M \setminus M' = M_1 \setminus M_2$. Since $M_2$ is an important $v \to T$ separator, it follows that the $v \to T$ separator $M_1$ is not contained in $M_2$. Therefore $M \setminus M'$ is non-empty. Furthermore, by definition of reverse shadow, $M \setminus M'$ is not contained in $r_G(M)$, but by Claim 15, it is contained in $r_G(M')$. ◀

By Claim 16, $M'$ is a solution and is no larger than $M$. Therefore, the set $M'$ has the properties claimed in Lemma 14. ◀

We recall that a set $M \subseteq V(G)$ is *thin*, if every node $v \in M$ is not in $r_G(M \setminus \{v\})$. The next result follows from Lemma 14.

▶ **Corollary 20.** *Let $(G, V^\infty, T, k)$ be an instance of* DagOddMultiwayNodeCut*, where $G$ is a DAG. Let $M^*$ be an optimal solution that maximizes the size of $|r_G(S) \cup f_G(S) \cup S|$ among all optimal solutions $S$. If more than one optimal solution maximizes this quantity, choose the one with largest $|r_G(S)|$. The set $M^*$ is thin and for every node $v \in r_G(M^*)$ there is an important $v \to T$ separator in $M^*$.*

We will use Corollary 20 to prove Lemma 11.

**Proof of Lemma 11.** Let us use ReverseShadowContainer$(G, V^\infty, k)$ to denote the algorithm from Theorem 13. We will show that Algorithm 3 generates the desired set.

In Algorithm 3 we use procedure ReverseShadowContainer introduced in Theorem 13. By Theorem 13, the cardinality of $\mathcal{Z}$ returned by the algorithm is $2^{O(k^2)}$. The runtime of the algorithm follows from the runtime of the procedure ReverseShadowContainer in Theorem 13. To prove the correctness of this algorithm, we argue that at least one of the sets in the returned family $\mathcal{Z}$ has the desired properties.

**Algorithm 3** ShadowContainer

Given: DAG $G$ with terminal set $T$, a set $V^\infty$ of protected nodes containing $T$, and $k \in \mathbb{Z}_+$

1: Let $G^{\text{rev}}$ denote the graph obtained from $G$ by reversing the orientation of all edges
2: $\mathcal{Z}_1 \leftarrow \text{ReverseShadowContainer}(G, V^\infty, k)$
3: **for** $Z_1 \in \mathcal{Z}_1$ **do**
4: $\quad \mathcal{Z}_2 \leftarrow \text{ReverseShadowContainer}(G^{\text{rev}}, V^\infty \cup Z_1, k)$
5: $\quad$ **for** $Z_2 \in \mathcal{Z}_2$ **do**
6: $\quad\quad \mathcal{Z} \leftarrow \mathcal{Z} \cup \{Z_1 \cup Z_2\}$
7: **return** $\mathcal{Z}$

Suppose there exists a solution of size at most $k$ and let $M^*$ be an optimal solution that maximizes the size of $|r_G(S) \cup f_G(S) \cup S|$ among all optimal solutions $S$. If more than one solution maximizes this quantity, choose the one with largest $|r_G(S)|$. By Corollary 20, the solution $M^*$ is thin and has the property that every node $v$ in the reverse shadow of $M^*$ has an important $v \to T$ separator contained in $M^*$. By Theorem 13, the procedure $\text{ReverseShadowContainer}(G, V^\infty, k)$ in Line 2 will return a family $\mathcal{Z}_1$ of sets containing a set $Z_1$ that is disjoint from $M^*$ and contains its reverse shadow. Let us fix such a $Z_1$.

We note that a solution for the DagOddMultiwayNodeCut instance $(G^{\text{rev}}, V^\infty \cup Z_1, T, k)$ is also a solution for the instance $(G, V^\infty, T, k)$. Conversely, a solution for the instance $(G, V^\infty, T, k)$ that is disjoint from $Z_1$ is also a solution for the instance $(G^{\text{rev}}, V^\infty \cup Z_1, T, k)$. Therefore, the set $M^*$ is also an optimal solution to the instance $(G^{\text{rev}}, V^\infty \cup Z_1, T, k)$. We observe that $f_G(S) = r_{G^{\text{rev}}}(S)$ and $r_G(S) = f_{G^{\text{rev}}}(S)$ for all $S \subseteq V(G) \setminus V^\infty$. Therefore, $M^*$ maximizes the size of $r_{G^{\text{rev}}}(S) \cup f_{G^{\text{rev}}}(S) \cup S$ among all optimal solutions $S$ to $(G^{\text{rev}}, V^\infty \cup Z_1, T, k)$. We have the following claim.

▶ **Claim 21.** *If for an optimal solution $M'$ for the instance $(G^{rev}, V^\infty \cup Z_1, T, k)$ of* DagOddMultiwayNodeCut*, we have $r_{G^{rev}}(M^*) \cup f_{G^{rev}}(M^*) \cup M^* \subseteq r_{G^{rev}}(M') \cup f_{G^{rev}}(M') \cup M'$ and $r_{G^{rev}}(M^*) \subseteq r_{G^{rev}}(M')$, then $M' = M^*$.*

**Proof.** As $M^*$ maximizes $|r_{G^{\text{rev}}}(S) \cup f_{G^{\text{rev}}}(S) \cup S|$ among all the optimal solutions for the instance $(G, V^\infty, T, k)$ and as $r_{G^{\text{rev}}}(M^*) \cup f_{G^{\text{rev}}}(M^*) \cup M^* \subseteq r_{G^{\text{rev}}}(M') \cup f_{G^{\text{rev}}}(M') \cup M'$, hence, the two sets $r_{G^{\text{rev}}}(M^*) \cup f_{G^{\text{rev}}}(M^*) \cup M^*$ and $r_{G^{\text{rev}}}(M') \cup f_{G^{\text{rev}}}(M') \cup M'$ must be equal. Therefore, the set $M' \setminus M^*$ is contained inside $r_{G^{\text{rev}}}(M^*) \cup f_{G^{\text{rev}}}(M^*) \cup M^*$. Since nodes in $f_{G^{\text{rev}}}(M^*)$ are protected in $G^{\text{rev}}$ by construction, the solution $M'$ cannot contain any node from $f_{G^{\text{rev}}}(M^*)$. Since $r_{G^{\text{rev}}}(M^*) \subseteq r_{G^{\text{rev}}}(M')$ and by definition of reverse shadow, $M'$ is disjoint from $r_{G^{\text{rev}}}(M^*)$. Thus, the set $M' \setminus M^*$ is disjoint from $M^*$ and $r_{G^{\text{rev}}}(M^*)$ and $f_{G^{\text{rev}}}(M^*)$, while being contained in $r_{G^{\text{rev}}}(M^*) \cup f_{G^{\text{rev}}}(M^*) \cup M^*$. Hence, $M' \setminus M^* = \emptyset$ or equivalently $M' \subseteq M^*$. Therefore, $M' = M^*$, because $|M'| = |M^*|$. ◀

Suppose there is a node $v \in r_{G^{\text{rev}}}(M^*)$ such that no important $v \to T$ separator in $G^{\text{rev}}$ is contained in $M^*$. Then by Lemma 14, there is another optimal solution $M'$ such that $r_{G^{\text{rev}}}(M^*) \cup f_{G^{\text{rev}}}(M^*) \cup M^* \subseteq r_{G^{\text{rev}}}(M') \cup f_{G^{\text{rev}}}(M') \cup M'$ and $r_{G^{\text{rev}}}(M^*) \subsetneq r_{G^{\text{rev}}}(M')$. By Claim 21, the set $M' = M^*$, which contradicts $r_{G^{\text{rev}}}(M^*) \subsetneq r_{G^{\text{rev}}}(M')$. This contradiction shows that for every node $v \in r_{G^{\text{rev}}}(M^*)$, there is an important $v \to T$ separator in $G^{\text{rev}}$ that is contained in $M^*$. Thus, by Theorem 13, the procedure $\text{ReverseShadowContainer}(G^{\text{rev}}, V^\infty \cup Z_1, k)$ from Line 4 will return a family $\mathcal{Z}_2$ of sets containing a set $Z_2$ that is disjoint from $M^*$ and contains $r_{G^{\text{rev}}}(M^*) = f_G(M^*)$. Hence $Z_1 \cup Z_2$ is disjoint from $M^*$ and contains $s_G(M^*)$. ◀

## References


1. Karthekeyan Chandrasekaran and Sahand Mozaffari. Odd Multiway Cut in Directed Acyclic Graphs. https://arxiv.org/abs/????.????, 2017.
2. 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.
3. Rajesh Hemant Chitnis, MohammadTaghi 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.
4. Maria Chudnovsky, Jim Geelen, Bert Gerards, Luis A. Goddyn, Michael Lohman, and Paul D. Seymour. Packing non-zero a-paths in group-labelled graphs. *Combinatorica*, 26(5):521–532, 2006. doi:10.1007/s00493-006-0030-1.
5. Ross Churchley, Bojan Mohar, and Hehui Wu. Weak duality for packing edge-disjoint odd (u, v)-trails. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 2086–2094, 2016.
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. Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards*, 69(1-2):125–130, 1965.
8. Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3):449–467, 1965.
9. Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.
10. Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988. doi:10.1007/978-3-642-78240-4.
11. Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. doi:10.1016/j.jcss.2007.06.019.
12. Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Magnus Wahlström. Fixed-parameter tractability of multicut in directed acyclic graphs. *SIAM J. Discrete Math.*, 29(1):122–144, 2015. doi:10.1137/120904202.
13. Andrea S. LaPaugh and Christos H. Papadimitriou. The even-path problem for graphs and digraphs. *Networks*, 14(4):507–513, 1984. doi:10.1002/net.3230140403.
14. 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.
15. Daniel Lokshtanov and M. S. Ramanujan. Parameterized tractability of multiway cut with parity constraints. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 750–761. Springer, 2012. doi:10.1007/978-3-642-31594-7_63.
16. Sridharan Ramanujan. *Parameterized Graph Separation Problems: New Techniques and Algorithms*. PhD thesis, The Institute of Mathematical Sciences, Chennai, 2013.
17. Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics. Springer, 2003.
18. Alexander Schrijver and Paul D. Seymour. Packing odd paths. *J. Comb. Theory, Ser. B*, 62(2):280–288, 1994. doi:10.1006/jctb.1994.1070.

# A Fixed-Parameter Perspective on #BIS*†

**Radu Curticapean[1], Holger Dell[2], Fedor V. Fomin[3], Leslie Ann Goldberg[4], and John Lapinskas[5]**

1 **Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary**
radu.curticapean@gmail.com
2 **Saarland University and Cluster of Excellence (MMCI), Saarbrücken, Germany**
hdell@mmci.uni-saarland.de
3 **University of Bergen, Norway**
fomin@ii.uib.no
4 **University of Oxford, UK**
leslie.goldberg@cs.ox.ac.uk
5 **University of Oxford, UK**
john.lapinskas@cs.ox.ac.uk

## Abstract

The problem of (approximately) counting the independent sets of a bipartite graph (#BIS) is the canonical approximate counting problem that is complete in the intermediate complexity class $\#\mathrm{RH\Pi}_1$. It is believed that #BIS does not have an efficient approximation algorithm but also that it is not NP-hard. We study the robustness of the intermediate complexity of #BIS by considering variants of the problem parameterised by the size of the independent set. We map the complexity landscape for three problems, with respect to exact computation and approximation and with respect to conventional and parameterised complexity. The three problems are counting independent sets of a given size, counting independent sets with a given number of vertices in one vertex class and counting maximum independent sets amongst those with a given number of vertices in one vertex class. Among other things, we show that all of these problems are NP-hard to approximate within any polynomial ratio. (This is surprising because the corresponding problems without the size parameter are complete in $\#\mathrm{RH\Pi}_1$, and hence are not believed to be NP-hard.) We also show that the first problem is #W[1]-hard to solve exactly but admits an FPTRAS, whereas the other two are W[1]-hard to approximate even within any polynomial ratio. Finally, we show that, when restricted to graphs of bounded degree, all three problems have efficient exact fixed-parameter algorithms.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics, G.2.2 Graph Theory

**Keywords and phrases** approximate counting, parameterised complexity, independent sets

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.13

* Part of this work was done while the authors were visiting the Simons Institute for the Theory of Computing. RC is supported by ERC grant PARAMTIGHT (No. 280152). The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) ERC grant agreement no. 334828. The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.

† The full version containing detailed proofs is available at http://arxiv.org/abs/1702.05543. The theorem numbering here matches the full version.





## 1 Introduction

The problem of (approximately) counting the independent sets of a bipartite graph, called #BIS, is one of the most important problems in the field of approximate counting. This problem is known [6] to be complete in the intermediate complexity class $\#\mathsf{RH\Pi}_1$. Many approximate counting problems are equivalent in difficulty to #BIS, including those that arise in spin-system problems [10, 11] and in other domains. These problems are not believed to have efficient approximation algorithms, but they are also not believed to be NP-hard.

In this paper we study the robustness of the intermediate complexity of #BIS by considering relevant fixed parameters. It is already known that the complexity of #BIS is unchanged when the *degree* of the input graph is restricted (even if it is restricted to be at most 6) [2] but there is an efficient approximation algorithm when a stronger degree restriction (degree at most 5) is applied, even to the vertices in just one of the parts of the vertex partition of the bipartite graph [14].

We consider variants of the problem parameterised by the *size* of the independent set. We first show that all of the following problems are #P-hard to solve exactly and NP-hard to approximate *within any polynomial factor*.

- #Size-BIS: Given a bipartite graph $G$ and a non-negative integer $k$, count the size-$k$ independent sets of $G$.
- #Size-Left-BIS: Given a bipartite graph $G$ with vertex partition $(U, V)$ and a non-negative integer $k$, count the independent sets of $G$ that have $k$ vertices in $U$, and
- #Size-Left-Max-BIS: Given a bipartite graph $G$ with vertex partition $(U, V)$ and a non-negative integer $k$, count the maximum independent sets amongst all independent sets of $G$ with $k$ vertices in $U$.

The NP-hardness of these approximate counting problems is surprising given that the corresponding problems without the parameter $k$ (that is, the problem #BIS and also the problem #Max-BIS, which is the problem of counting the *maximum* independent sets of a bipartite graph) are both complete in $\#\mathsf{RH\Pi}_1$, and hence are not believed to be NP-hard. Therefore, it is the introduction of the parameter $k$ that causes the hardness.

To gain a more refined perspective on these problems, we also study them from the perspective of parameterised complexity, taking the number of vertices, $n$, as the size of the input and $k$ as the fixed parameter. Our results are summarised in Table 1, and stated in detail later in the paper. Rows 1 and 3 of the table correspond to the conventional (exact and approximate) setting that we have already discussed. Rows 2 and 4 correspond to the parameterised complexity setting, which we discuss next. As becomes apparent from the table, we have mapped the complexity landscape for the three problems in all four settings.

In parameterised complexity, the central goal is to determine whether computational problems have fixed-parameter tractable (FPT) algorithms, that is, algorithms that run in time $f(k) \cdot n^{O(1)}$ for some computable function $f$. Hardness results are presented using the $W$-hierarchy [8], and in particular using the complexity classes W[1] and W[2], which constitute the first two levels of the hierarchy. It is known (see [8]) that FPT $\subseteq$ W[1] $\subseteq$ W[2] and these classes are widely believed to be distinct from each other. It is also known [4, Chapter 14] that the Exponential Time Hypothesis (see [12]) implies FPT $\neq$ W[1]. Analogous classes #W[1] and #W[2] exist for counting problems [7].

As can be seen from the table, we prove that all of our problems are at least W[1]-hard to solve exactly, which indicates that they cannot be solved by FPT algorithms. Moreover, #Size-Left-BIS and #Size-Left-Max-BIS are W[1]-hard to solve even approximately. It is known [16] that each parameterised counting problem in the class #W[i] has a randomised

**Table 1** Our results. Each of the three problems that we consider (#Size-BIS, #Size-Left-BIS, #Size-Left-Max-BIS) has one column here, in which we list our results in all four settings (exact polynomial-time, exact FPT-time, approximate polynomial-time, approximate FPT-time).

| | #Size-BIS | #Size-Left-BIS | #Size-Left-Max-BIS |
|---|---|---|---|
| Exact poly | #P-complete even in graphs of max-degree 3. (Thm 1 full version) | #P-complete even in graphs of max-degree 3. (Thm 1 full version) | #P-hard even in graphs of max-degree 3. (Thm 2 full version) |
| Exact FPT | #W[1]-complete. (Thm 4 full version)<br>FPT for bounded-degree graphs. (Thm 14(i)) | #W[2]-hard. (Thm 5)<br>FPT for bounded-degree graphs. (Thm 14(ii)) | W[1]-hard. (Thm 6)<br>FPT for bounded-degree graphs. (Thm 14(iii)) |
| Approx poly | NP-hard to approximate within any polynomial factor. (Thm 9) | NP-hard to approximate within any polynomial factor. (Thm 7) | NP-hard to approximate within any polynomial factor. (Thm 6) |
| Approx FPT | Has FPTRAS. (Thm 11) | W[1]-hard to approximate within any polynomial factor. (Thm 7) | W[1]-hard to approximate within any polynomial factor. (Thm 6) |

FPT approximation algorithm using a W[i] oracle, so W[i]-hardness is the appropriate hardness notion for parameterised approximate counting problems. By contrast, we show that #Size-BIS *can* be solved approximately in FPT time. In fact, it has an FPT randomized approximation scheme (FPTRAS).

Motivated by the fact that #BIS is known to be #P-complete to solve exactly even on graphs of degree 3 [19], we also consider the case where the input graph has bounded degree. While the conventional problems remain intractable in this setting (Row one of the table), we prove that all three of our problems admit linear-time fixed-parameter algorithms for bounded-degree instances (Row two). Note that Theorem 14(i) is also implicit in independent work by Patel and Regts [17].

## 2 Preliminaries

For a positive integer $n$, we let $[n]$ denote the set $\{1, \ldots, n\}$. We consider graphs $G$ to be undirected. For a vertex set $X \subseteq V(G)$, denote by $G[X]$ the subgraph induced by $X$. For a vertex $v \in V(G)$, we write $\Gamma(v)$ for its open neighbourhood (that is, excluding $v$ itself).

Given a graph $G$, we denote the size of a maximum independent set of $G$ by $\mu(G)$. We denote the number of all independent sets of $G$ by $\mathrm{IS}(G)$, the number of size-$k$ independent sets of $G$ by $\mathrm{IS}_k(G)$, and the number of size-$\mu(G)$ independent sets of $G$ by $\mathrm{MIS}(G)$. A bipartite graph $G$ is presented as a triple $(U, V, E)$ in which $(U, V)$ is a partition of the vertices of $G$ and $E$ is a subset of $U \times V$. If $G = (U, V, E)$ is a bipartite graph then an independent set $S$ of $G$ is said to be an "$\ell$-left independent set of $G$" if $|S \cap U| = \ell$. The size of a *maximum* $\ell$-left independent set of $G$ is denoted by $\mu_{\ell\text{-left}}(G)$. An $\ell$-left independent set of $G$ is said to be "$\ell$-left-maximum" if and only if it has size $\mu_{\ell\text{-left}}(G)$. Finally, $\mathrm{IS}_{\ell\text{-left}}(G)$ denotes the number of $\ell$-left independent sets of $G$ and $\mathrm{IS}_{\ell\text{-left-max}}(G)$ denotes the number of $\ell$-left-maximum independent sets of $G$. Using these definitions, we now give formal definitions of #BIS and of the three problems that we study.

**Name:** #BIS.
**Input:** Bipartite graph $G$.
**Output:** $\mathrm{IS}(G)$.

**Name:** #Size-BIS
**Input:** Bipartite $G$ and $k \in \mathbb{N}$.
**Output:** $\mathrm{IS}_k(G)$.
**Parameter:** $k$.

**Name:** #Size-Left-BIS
**Input:** Bipartite $G$ and $\ell \in \mathbb{N}$.
**Output:** $\mathrm{IS}_{\ell\text{-left}}(G)$.
**Parameter:** $\ell$.

**Name:** #Size-Left-Max-BIS
**Input:** Bipartite $G$ and $\ell \in \mathbb{N}$.
**Output:** $\mathrm{IS}_{\ell\text{-left-max}}(G)$.
**Parameter:** $\ell$.

For each of our computational problems, we add "[$\Delta$]" to the end of the name of the problem to indicate that the input graph $G$ has degree at most $\Delta$. For example, the input of #BIS[$\Delta$] is a bipartite graph $G$ with degree at most $\Delta$, and the desired output is $\mathrm{IS}(G)$.

When stating quantitative bounds on running times of algorithms, we assume the standard word-RAM machine model with logarithmic-sized words.

## 3 Exact computation: fixed-parameter intractability

Our #P-hardness results (from Row 1 of Table 1) are in the full version. For the rest of the paper, we use standard definitions of reductions and complexity classes which are in Flum and Grohe [8] and in the full version. We defer the proof of Theorem 4, which shows that #Size-BIS is #W[1]-complete, to the full version. We give the following, stronger, hardness result for #Size-Left-BIS.

▶ **Theorem 5.** #Size-Left-BIS *is* #W[2]*-hard.*

**Proof.** Recall that if $G$ is a graph, a set $D \subseteq V(G)$ is called a *dominating set* of $G$ if every vertex $v \in V(G)$ is either contained in $D$ or adjacent to a vertex of $D$. We reduce from #Size-Dominating-Set, which is the problem of computing the number of size-$k$ dominating sets given a graph $G = (U, E)$ and a positive integer $k$. (The parameter of #Size-Dominating-Set is $k$.) Note that #Size-Dominating-Set is #W[2]-complete, as proved in Flum and Grohe [7, Theorem 19].

Write $U = \{u_1, \ldots, u_n\}$. The reduction computes the bipartite split graph of $G$; formally, let $V = \{v_1, \ldots, v_n\}$, let $E' = \{(u_a, v_b) \mid a = b \text{ or } \{u_a, u_b\} \in E\}$, and let $G' = (U, V, E')$.

For non-negative integers $\ell$ and $r$, we define an $(\ell, r)$*-set of* $G'$ to be a size-$\ell$ subset $X$ of $U$ that has exactly $r$ neighbours in $V$. Let $Z_{\ell,r}$ be the number of $(\ell, r)$-sets of $G'$. Note that a size-$k$ subset $X$ of $U$ is a dominating set of $G$ if and only if it is a $(k, n)$-set of $G'$, so there are precisely $Z_{k,n}$ size-$k$ dominating sets of $G$.

The algorithm applies polynomial interpolation to determine $Z_{k,r}$ for all $r \in \{0, \ldots, n\}$. For every positive integer $i$, let $V_i = V \times [i]$, let $E'_i = \{(u, (v, b)) \in U \times V_i \mid (u, v) \in E'\}$, and let $G'_i = (U, V_i, E'_i)$. For each $(k, r)$-set $X$ of $G'$, there are exactly $2^{i(n-r)}$ $k$-left independent sets $S$ of $G'_i$ with $S \cap U = X$. Thus for all $i \in [n+1]$,

$$\mathrm{IS}_{k\text{-left}}(G'_i) = \sum_{r=0}^{n} 2^{i(n-r)} Z_{k,r}. \tag{1}$$

Let $M$ be the $(n+1) \times (n+1)$ matrix whose rows are indexed by $[n+1]$ and columns are indexed by $\{0, \ldots, n\}$ such that $M_{i,r} = 2^{i(n-r)}$ holds. Then (1) can be viewed as a linear equation system $\boldsymbol{w} = M\boldsymbol{z}$, where $\boldsymbol{w} = (\mathrm{IS}_{k\text{-left}}(G'_1), \ldots, \mathrm{IS}_{k\text{-left}}(G'_{n+1}))^T$ and $\boldsymbol{z} = (Z_{k,0}, \ldots, Z_{k,n})^T$. The oracle for #Size-Left-BIS can be used to compute $\boldsymbol{w}$, and $M$ is invertible since it is a (transposed) Vandermonde matrix. Thus the reduction can compute $\boldsymbol{z}$, and in particular $Z_{k,n}$, as required. ◀

We defer the proof of the remaining hardness result in Row 2 of Table 1 (W[1]-hardness of #Size-Left-Max-BIS) to the next section, as it is implied by the corresponding approximation hardness result.

## 4 Approximate computation: Hardness results

In this section, we prove the hardness results in Rows 3 and 4 of Table 1. Note that the reductions from the first row of the table cannot be used here, since they are ultimately from #BIS, which is not known to be NP-hard to approximate. In order to state our hardness results formally, we introduce approximation versions of the problems that we consider.

**Name:** Deg-$c$-#ApxSizeLeftMaxBIS. **Parameter:** $\ell$.
**Input:** A bipartite graph $G$ on $n$ vertices and a non-negative integer $\ell$.
**Output:** A number $z$ such that $n^{-c} \cdot \mathrm{IS}_{\ell\text{-left-max}}(G) \leq z \leq n^c \cdot \mathrm{IS}_{\ell\text{-left-max}}(G)$.

**Name:** Deg-$c$-#ApxSizeLeftBIS. **Parameter:** $\ell$.
**Input:** A bipartite graph $G$ on $n$ vertices and a non-negative integer $\ell$.
**Output:** A number $z$ such that $n^{-c} \cdot \mathrm{IS}_{\ell\text{-left}}(G) \leq z \leq n^c \cdot \mathrm{IS}_{\ell\text{-left}}(G)$.

**Name:** Deg-$c$-#ApxSizeBIS. **Parameter:** $k$.
**Input:** A bipartite graph $G$ on $n$ vertices and a non-negative integer $k$.
**Output:** A number $z$ such that $n^{-c} \cdot \mathrm{IS}_k(G) \leq z \leq n^c \cdot \mathrm{IS}_k(G)$.

We also require the following problem for reductions.

**Name:** Size-Clique. **Parameter:** $k$.
**Input:** A graph $G$ and a positive integer $k$.
**Output:** True if $G$ contains a $k$-clique, false otherwise.

We first prove our #Size-Left-Max-BIS results, then establish the others by reduction.

▶ **Theorem 6.** *For all $c \geq 0$,* Deg-$c$-#ApxSizeLeftMaxBIS *is both* NP*-hard and* W[1]*-hard.*

**Proof.** Let $c$ be any non-negative integer. We will give a reduction from Size-Clique to Deg-$c$-#ApxSizeLeftMaxBIS which is both an FPT Turing reduction and a polynomial-time Turing reduction. The claim then follows from the fact that Size-Clique is both NP-hard [18, Theorem 7.32]) and W[1]-hard [5, Theorem 21.3.4].

Let $(G, k)$ be an instance of Size-Clique with $G = (V, E)$ and $n = |V|$. We use a standard powering construction to produce an intermediate instance $(G', k)$ of Size-Clique with $G' = (V', E')$. More precisely, let $t = n^{2c}$, let $C$ be a set of $k$ new vertices, and let $V' = (V \times [t]) \cup C$. We define $E'$ such that

$$E' = \big\{\{(u,i),(v,j)\} \mid \{u,v\} \in E, i,j \in [t]\big\} \cup \big\{\{u,v\} \mid u,v \in C, u \neq v\big\}.$$

From $(G', k)$, we construct an instance $(G'', \ell)$ of Deg-$c$-#ApxSizeLeftMaxBIS with $G'' = (U, V', E'')$ and $\ell = \binom{k}{2}$. For this, let $U = \{u_e \mid e \in E'\}$ be a set of vertices and let $E'' = \{(u_e, v) \mid e \in E', v \in e\}$. The reduction queries the oracle for $(G'', \ell)$, which yields an approximate value $z$ for the number $\mathrm{IS}_{\ell\text{-left-max}}(G'')$. If $z \leq n^c$, the reduction returns 'no', there is no $k$-clique in $G$, and otherwise it returns 'yes'. It is obvious that the reduction runs in polynomial time.

It remains to prove the correctness of the reduction. Let $\mathrm{CL}_k(G)$ be the number of $k$-cliques in $G$. The $\ell$-left-maximum independent sets $X$ of $G''$ correspond bijectively to the size-$\ell$ edge sets $\{e \mid u_e \in X \cap U\}$ of $G'$ which span a minimum number of vertices.

Note that any set of $\ell = \binom{k}{2}$ edges must span at least $k$ vertices, with equality only in the case of a $k$-clique. Since $G'$ contains at least one $k$-clique (induced by $C$), we have $\mathrm{IS}_{\ell\text{-left-max}}(G'') = \mathrm{CL}_k(G')$. Moreover, each $k$-clique $X$ in $G$ corresponds to a size-$t^k$ family of $k$-cliques in $G'$. Each $k$-clique in the family consists of exactly one vertex from each set $\{x\} \times [t]$ such that $x \in V(X)$. This accounts for all $k$-cliques in $G'$ except $G'[C]$. Thus $\mathrm{IS}_{\ell\text{-left-max}}(G'') = \mathrm{CL}_k(G') = t^k \mathrm{CL}_k(G) + 1$.

Let $z$ be the result of applying our oracle to $(G'', \ell)$. If $G$ contains no $k$-cliques, then we have $z \leq n^c \cdot \mathrm{IS}_{\ell\text{-left-max}}(G'') = n^c$ and the reduction returns 'no'. Otherwise, we have $z \geq n^{-c} \cdot \mathrm{IS}_{\ell\text{-left-max}}(G'') \geq n^{-c}(t^k + 1) > n^c$ and the reduction returns 'yes'. Thus the reduction is correct and the claim follows. ◀

▶ **Theorem 7.** *For all $c \geq 0$,* Deg-$c$-#ApxSizeLeftBIS *is both* NP*-hard and* W[1]*-hard.*

**Proof.** Let $c \geq 0$ be an integer. We will give a reduction from the problem Deg-$(c+1)$-#ApxSizeLeftMaxBIS to the problem Deg-$c$-#ApxSizeLeftBIS which is both an FPT Turing reduction and a polynomial-time Turing reduction. The result then follows by Theorem 6.

Let $(G, \ell)$ be an instance of Deg-$c$-#ApxSizeLeftMaxBIS. Write $G = (U, V, E)$, let $n = |V(G)|$, and let $t = 6n$. Without loss of generality, suppose $n \geq 5$ and $n$ is sufficiently large that $n^c 2^{-n} \leq 1$. Let $V' = V \times [t]$, let $E' = \{(u, (v, i)) \mid (u, v) \in E,\ i \in [t]\}$, and let $G' = (U, V', E')$. Let $\mu = \mu_{\ell\text{-left}}(G)$, and let $z$ be the result of applying our oracle to $(G', \ell)$.

For any non-negative integers $i$ and $j$, we define $\mathrm{IS}_{i,\,j}(G)$ to be the number of independent sets $X \subseteq V(G)$ with $|X \cap U| = i$ and $|X \cap V| = j$. Each $\ell$-left independent set $X$ of $G$ corresponds to the family of $\ell$-left independent sets of $G'$ consisting of $X \cap U$ together with at least one vertex from each set $\{x\} \times [t]$ such that $x \in X \cap V$. Thus by the definition of $\mu$,

$$\mathrm{IS}_{\ell\text{-left}}(G') = \sum_{r=0}^{\mu-\ell} \mathrm{IS}_{\ell,\,r}(G)(2^t - 1)^r. \tag{2}$$

Since $G$ contains at most $2^n$ independent sets and $\mathrm{IS}_{\ell,\,\mu-\ell}(G) \geq 1$, we have $(2^t-1)^{\mu-\ell} \leq \mathrm{IS}_{\ell\text{-left}}(G') \leq 2^n(2^t-1)^{\mu-\ell}$. Since $n^c \leq 2^n \leq (2^t-1)^{1/5}$, it follows that $(2^t-1)^{\mu-\ell-1/5} \leq z \leq (2^t-1)^{\mu-\ell+2/5}$, and hence the algorithm can obtain $\mu$ by rounding $\ell + \lg(z)/\lg(2^t-1)$ to the nearest integer. Moreover, by (2) we have

$$\mathrm{IS}_{\ell\text{-left}}(G') \leq \mathrm{IS}_{\ell,\,\mu-\ell}(G)(2^t-1)^{\mu-\ell} + 2^n(2^t-1)^{\mu-\ell-1} \leq 2\mathrm{IS}_{\ell,\,\mu-\ell}(G)(2^t-1)^{\mu-\ell}.$$

It follows that $\mathrm{IS}_{\ell,\,\mu-\ell}(G) \leq \mathrm{IS}_{\ell\text{-left}}(G')/(2^t-1)^{\mu-\ell} \leq 2\mathrm{IS}_{\ell,\,\mu-\ell}(G)$. Hence $n^{-c-1}\mathrm{IS}_{\ell,\,\mu-\ell}(G) \leq z/(2^t-1)^{\mu-\ell} \leq n^{c+1}\mathrm{IS}_{\ell,\,\mu-\ell}(G)$. The algorithm therefore outputs $z/(2^t-1)^{\mu-\ell}$. ◀

▶ **Theorem 9.** *For all $c \geq 0$,* Deg-$c$-#ApxSizeBIS *is* NP*-hard.*

**Proof.** For all $c \geq 0$, we give a polynomial-time Turing reduction from the problem Deg-$(c+1)$-#ApxSizeLeftBIS to the problem Deg-$c$-#ApxSizeBIS. The former is NP-hard by Theorem 7.

Fix $c \geq 0$ and let $(G, \ell)$ be an instance of Deg-$(c+1)$-#ApxSizeLeftBIS. Suppose that $G = (U, V, E)$ where $U = \{u_1, \ldots, u_p\}$. Note from the problem definition that $n = |U \cup V|$ and suppose without loss of generality that $\ell \in [p]$ and that $n \geq 40$ (otherwise, $(G, \ell)$ is an easy instance of Deg-$(c+1)$-#ApxSizeLeftBIS, so the answer can be computed, even without using the oracle).

Let $s = 2n^6$ and $t = \lfloor s \log_2 3 \rfloor - s$. For each $i \in [p]$, let $U_i$, $V_i$ and $U_i'$ be disjoint sets of vertices with $|U_i'| = |V_i| = s$ and $|U_i| = t$. Write $U_i' = \{u_{i,1}, \ldots, u_{i,s}\}$ and $V_i = \{v_{i,1}, \ldots, v_{i,s}\}$.

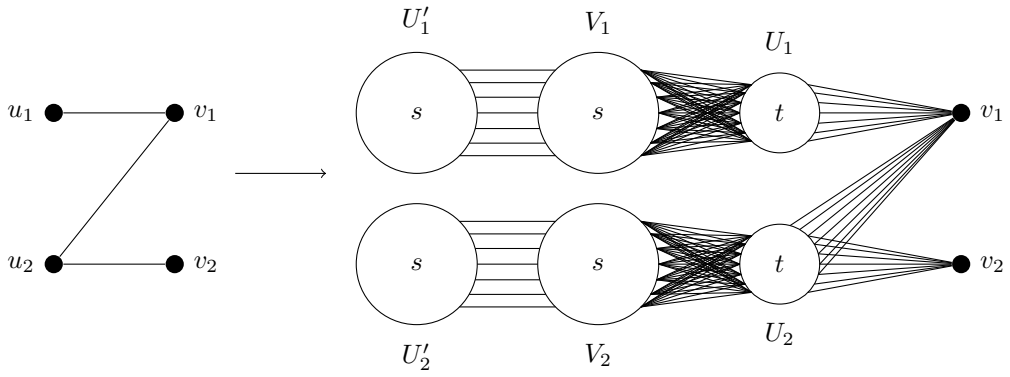



**Figure 1** An example of the reduction from Deg-$(c+1)$-#ApxSizeLeftBIS to Deg-$c$-#ApxSizeLeftBIS used in the proof of Theorem 9 when $G = P_3$. Each vertex $u_i \in U$ is replaced by three vertex sets $U_i'$, $V_i$ and $U_i$ in the resulting graph $G'$. Note that $G'$ does not depend on the input parameter $\ell$.

Then let $U' = \bigcup_{i\in[p]}(U_i \cup U_i')$, $V' = \bigcup_{i\in[p]} V_i \cup V$, and

$$E' = \bigcup_{i\in[p]} \Big((U_i \times V_i) \cup \{(u_{i,j}, v_{i,j}) \mid j \in [s]\}\Big) \cup \bigcup_{(u_j,v)\in E(G)} (U_j \times \{v\}).$$

Let $G' = (U', V', E')$, as depicted in Figure 1.

Intuitively, the proof will proceed as follows. We will map independent sets $X'$ of $G'$ to independent sets $X$ of $G$ by taking $X \cap V = X' \cap V$ and adding each $u_i \in U$ to $X$ if and only if $U_i \cap X' \neq \emptyset$. We will show that roughly half the independent sets of each gadget $U_i' \cup V_i \cup U_i$ have this form. We will also show that within each gadget, almost all independent sets with vertices in $U_i$ have size roughly $(s+t)/2$, and almost all others have size roughly $2s/3$. Thus the independent sets in $G$ with $\ell$ vertices in $U$ roughly correspond to the independent sets in $G'$ of size roughly $\ell \cdot (s+t)/2 + (p-\ell)\cdot 2s/3$, which we count using a #Size-BIS oracle.

We start by defining disjoint sets of independent sets of $G'$. For $x \in \{0,\dots,p\}$, let $E(x) = \frac{2s}{3}(p-x) + \frac{s+t}{2}x$ and let

$$\mathcal{A}_x = \left\{ X' \subseteq V(G') \;\middle|\; X' \text{ is an independent set of } G' \text{ and } \big||X'| - E(x)\big| \leq \tfrac{s}{20} + n \right\}.$$

Note that since $n \geq 3$, we have $t > 17s/30$ and $120n \leq s$. Thus, if $x' > x$,

$$E(x') - E(x) = (t/2 - s/6)\,(x' - x) > (17/60 - 1/6)\,s = s/10 + s/60 \geq s/10 + 2n.$$

We conclude that the sets $\mathcal{A}_0, \dots, \mathcal{A}_p$ are disjoint.

Next, we connect the independent sets of $G'$ with those of $G$. Each independent set $X'$ of $G'$ projects onto the independent set $(X' \cap V) \cup \{u_i \mid X' \cap U_i \neq \emptyset\}$ of $G$. Given an independent set $X$ of $G$, let $\varphi(X)$ be the set of independent sets $X'$ of $G'$ which project onto $X$. If $u_i \in X$, then there are $2^t - 1$ possibilities for $X' \cap U_i$ and $2^s$ possibilities for $X' \cap U_i'$, but $X' \cap V_i$ is empty. If $u_i \notin X$, then $X' \cap U_i$ is empty and there are $3^s$ possibilities for $X' \cap (U_i' \cup V_i)$. For $x \in \{0,\dots,p\}$, let $F(x) = (2^{s+t} - 2^s)^x \cdot 3^{(p-x)s}$. It follows that, for any $x$-left independent set $X$ of $G$, $|\varphi(X)| = F(x)$, which establishes the first of the following claims.

**Claim 1.** For any $\ell$-left independent set $X$ of $G$, $|\varphi(X) \cap \mathcal{A}_\ell| \leq F(\ell)$.

**Claim 2.** For any $\ell$-left independent set $X$ of $G$, $|\varphi(X) \cap \mathcal{A}_\ell| \geq F(\ell)/2$.

**Claim 3.** For any $x \in \{0, \ldots, p\} \setminus \{\ell\}$ and any $x$-left independent set $X$ of $G$, $|\varphi(X) \cap \mathcal{A}_\ell| \leq F(\ell)/2^n$.

The proofs of Claims 2 and 3 are mere calculation, so before proving them we use the claims to complete the proof of the lemma. Recall that $(G, \ell)$ is an instance of Deg-$(c+1)$-#ApxSizeLeftBIS with $\ell \in [p]$ and $n \geq 2$. Together, the claims imply

$$(F(\ell)/2) \cdot \mathrm{IS}_{\ell\text{-left}}(G) \leq |\mathcal{A}_\ell| \leq F(\ell)\mathrm{IS}_{\ell\text{-left}}(G) + F(\ell), \tag{3}$$

where the final $F(\ell)$ comes from the contribution to $|\mathcal{A}_\ell|$ corresponding to the (at most $2^n$) independent sets of $G$ that are not $\ell$-left independent sets. Since $\ell \in [p]$, the quantity $\mathrm{IS}_{\ell\text{-left}}(G)$ is at least 1, which means that the right-hand side of (3) is at most $2F(\ell)\mathrm{IS}_{\ell\text{-left}}(G)$. Also, $F(\ell) > 0$. Thus, (3) implies $\mathrm{IS}_{\ell\text{-left}}(G)/2 \leq |\mathcal{A}_\ell|/F(\ell) \leq 2\mathrm{IS}_{\ell\text{-left}}(G)$.

The oracle for Deg-$c$-#ApxSizeBIS can be used to compute a number $z$ such that $n^{-c}|\mathcal{A}_\ell| \leq z \leq n^c|\mathcal{A}_\ell|$. (To do this, just call the oracle repeatedly with input $G'$ and with every non-negative integer $k$ such that $|k - E(\ell)| \leq \frac{s}{20} + n$, adding the results.) Thus,

$$n^{-c}\mathrm{IS}_{\ell\text{-left}}(G)/2 \leq n^{-c}|\mathcal{A}_\ell|/F(\ell) \leq z/F(\ell) \leq n^c|\mathcal{A}_\ell|/F(\ell) \leq 2n^c\mathrm{IS}_{\ell\text{-left}}(G),$$

so the desired approximation of $\mathrm{IS}_{\ell\text{-left}}(G)$ can be achieved by dividing $z$ by $F(\ell)$. We now complete the proof by proving Claims 2 and 3.

**Claim 2:** Consider any $x \in \{0, \ldots, p\}$ and let $X$ be an $x$-left independent set of $G$. We will show $|\varphi(X) \cap \mathcal{A}_x| \geq F(x)/2$, which implies the claim by taking $\ell = x$. In fact, we will establish the much stronger inequality

$$|\varphi(X) \cap \mathcal{A}_x| \geq (1 - 3ne^{-n^2})F(x), \tag{4}$$

which will also be useful in the proof of Claim 3. To establish Equation (4) we will show that the probability that a random element $Y$ of $\varphi(X)$ satisfies $\big||Y| - E(x)\big| \leq \frac{s}{20} + n$ is at least $1 - 3ne^{-n^2}$.

So let $Y$ be a uniformly random element of $\varphi(X)$. We will show that, with probability at least $1 - 3ne^{-n^2}$, the following bullet points hold.

- For all $i \in [p]$ with $u_i \notin X$, we have $\big||Y \cap (U_i \cup V_i \cup U_i')| - \frac{2s}{3}\big| \leq \frac{s}{n^2}$, and
- for all $i \in [p]$ with $u_i \in X$, we have $\big||Y \cap (U_i \cup V_i \cup U_i')| - \frac{s+t}{2}\big| \leq \frac{s+t}{n^2}$,

Since $n \geq 40$, we have $(p-x)s/n^2 + x(s+t)/n^2 \leq 2ps/n^2 \leq s/20$ and $|Y \cap V| \leq n$, so the claim follows. To obtain the desired failure probability, we will show that, for any $i \in [p]$, the probability that the relevant bullet point fails to hold is at most $3e^{-n^2}$ (so the total failure probability is at most $3ne^{-n^2}$, by a union bound).

First, consider any $i \in [p]$ with $u_i \notin X$. In this case, $Y \cap (U_i \cup V_i \cup U_i')$ is generated by including (independently for each $j \in [s]$) one of three possibilities: (i) $u_{i,j}$ but not $v_{i,j}$, (ii) $v_{i,j}$ but not $u_{i,j}$, or (iii) neither $u_{i,j}$ nor $v_{i,j}$. Each of the three choices is equally likely. Thus $|Y \cap (U_i \cup V_i \cup U_i')|$ is distributed binomially with mean $2s/3$, so by a Chernoff bound (see Janson, Łuczak and Rucinski [13, Corollary 2.3]), the probability that the first bullet point fails for $i$ is at most $2e^{-s/2n^4} < 3e^{-n^2}$, as desired.

Second, consider any $i \in [p]$ with $u_i \in X$. In this case, $Y \cap (U_i \cup V_i \cup U_i')$ is chosen uniformly from all subsets of $U_i \cup U_i'$ that contain at least one element of $U_i$. The total variation distance between the uniform distribution on these subsets and the uniform distribution on all subsets of $U_i \cup U_i'$ is at most $2^{-t}$. Also, again by [13, Corollary 2.3]), the probability that a uniformly-random subset of $U_i \cup U_i'$ has a size that differs from its mean, $(s+t)/2$, by at

least $(s+t)/n^2$ is at most $2e^{-2(s+t)/(3n^4)}$. Thus, the probability that the second bullet point fails for $i$ is at most $2^{-t} + 2e^{-2(s+t)/(3n^4)} \leq 3e^{-n^2}$, as desired.

**Claim 3:** Suppose that $x \in \{0, \ldots, p\} \setminus \{\ell\}$ and that $X$ is an $x$-left independent set of $G$. We know from Equation (4) that $|\varphi(X) \cap \mathcal{A}_\ell| \leq 3ne^{-n^2}F(x)$. We wish to show that this is at most $F(\ell)/2^n$. Note that $t \geq 1$ and $3^{s-1} \leq 2^{s+t} \leq 3^s$, so for all $y \in \{0, \ldots, p\}$,

$$F(y) = (2^{s+t} - 2^s)^y \cdot 3^{ps-ys} \leq 2^{y(s+t)} \cdot 3^{ps-ys} \leq 3^{ps}, \text{and}$$
$$F(y) \geq 2^{y(s+t)-y} \cdot 3^{ps-ys} \geq 3^{ps-2y} \geq 3^{ps-2n}.$$

The claim follows from $F(x) \leq 3^{ps} \leq 3^{2n}F(\ell)$ and from the fact that $n \geq 40$. ◀

## 5 Algorithms

In this final section, we give our algorithmic results: An FPT randomized approximation scheme (FPTRAS) for #Size-BIS, and an exact FPT-algorithm for all three problems in bounded-degree graphs. The definition below follows Arvind and Raman [1].

▶ **Definition 10.** *An* FPTRAS *for* #Size-BIS *is a randomised algorithm that takes as input a bipartite graph $G$, a non-negative integer $k$, and a real number $\varepsilon \in (0,1)$ and outputs a real number $z$. With probability at least $2/3$, the output $z$ must satisfy $(1-\varepsilon)\mathrm{IS}_k(G) \leq z \leq (1+\varepsilon)\mathrm{IS}_k(G)$. Furthermore, there is a function $f : \mathbb{R} \to \mathbb{R}$ and a polynomial $p$ such that the running time of the algorithm is at most $f(k)\,p(|V(G)|, 1/\varepsilon)$.*

▶ **Theorem 11.** *There is an FPTRAS for* #Size-BIS *with time complexity $O\left(2^k \cdot k^2/\varepsilon^2\right)$ for input graphs with $n$ vertices.*

Note that the running time in Theorem 11 does not depend on $n$, as various logarithmic factors are absorbed by the word-RAM model. We defer the proof to the full version. We now turn to our algorithms for bounded-degree graphs. We require the following definitions. For any positive integer $s$, an *$s$-coloured graph* is a tuple $(G, c)$ where $G$ is a graph and $c : V(G) \to [s]$ is a map. Suppose $\mathcal{G} = (G, c)$ and $\mathcal{G}' = (G', c')$ are coloured graphs with $G = (V, E)$ and $G' = (V', E')$.

We say a map $\phi : V \to V'$ is a *homomorphism* from $\mathcal{G}$ to $\mathcal{G}'$ if $\phi$ is a homomorphism from $G$ to $G'$ and, for all $v \in V$, $c(v) = c'(\phi(v))$. If $\phi$ is also bijective, we say $\phi$ is an *isomorphism* from $\mathcal{G}$ to $\mathcal{G}'$, that $\mathcal{G}$ and $\mathcal{G}'$ are *isomorphic*, and write $\mathcal{G} \simeq \mathcal{G}'$. For all $X \subseteq V$, we define $\mathcal{G}[X] = (G[X], c|_X)$, and say $\mathcal{G}[X]$ is an *induced subgraph* of $\mathcal{G}$. Given coloured graphs $\mathcal{H}$ and $\mathcal{G}$, we denote the number of sets $X \subseteq V(\mathcal{G})$ with $\mathcal{G}[X] \simeq \mathcal{H}$ by $\#\mathrm{Ind}(\mathcal{H} \to \mathcal{G})$. Finally, we define $V(\mathcal{G}) = V$ and $E(\mathcal{G}) = E$ and we define $\Delta(\mathcal{G})$ to be the maximum degree of $G$.

For each positive integer $\Delta$, we consider a counting version of the induced subgraph isomorphism problem for coloured graphs of degree at most $\Delta$.

**Name:** #Induced-Coloured-Subgraph[$\Delta$]. **Parameter:** $|V(\mathcal{H})|$.
**Input:** Two coloured graphs $\mathcal{H}$ and $\mathcal{G}$, each with maximum degree bounded by $\Delta$.
**Output:** $\#\mathrm{Ind}(\mathcal{H} \to \mathcal{G})$.

We will later reduce our bipartite independent set counting problems to #Induced-Coloured-Subgraph[$\Delta$]. Note that this problem can be expressed as a first-order model-counting problem in bounded-degree structures. A well-known result of Frick [9, Theorem 6] yields an algorithm for #Induced-Coloured-Subgraph[$\Delta$] with running time $g(k) \cdot n$, where $k = |V(\mathcal{H})|$ and $n = |V(\mathcal{G})|$. However, the function $g$ of Frick's algorithm may grow faster

than any constant-height tower of exponentials. In the following, we provide an algorithm for #Induced-Coloured-Subgraph[$\Delta$] that is substantially faster: It runs in time $O(nk^{(2\Delta+3)k})$.

The algorithm follows the strategy of [3] to count small subgraphs: Instead of counting (coloured) induced subgraphs, we can count (coloured) homomorphisms and recover the number of induced subgraphs via a simple basis transformation. Given coloured graphs $\mathcal{H}$ and $\mathcal{G}$, we denote the number of homomorphisms from $\mathcal{H}$ to $\mathcal{G}$ by $\#\mathrm{Hom}(\mathcal{H} \to \mathcal{G})$.

▶ **Lemma 12.** *There is an algorithm to compute $\#\mathrm{Hom}(\mathcal{H} \to \mathcal{G})$ in time $O(nk^k(\Delta+1)^k)$, where $\mathcal{G}$ is a coloured graph with $n$ vertices, $\mathcal{H}$ is a coloured graph with $k$ vertices, and both graphs have maximum degree at most $\Delta$.*

**Proof.** The algorithm works as follows: If $\mathcal{H}$ is not connected, let $\mathcal{H}_1, \ldots, \mathcal{H}_\ell$ be its connected components. Then it is straightforward to verify that $\#\mathrm{Hom}(\mathcal{H} \to G) = \prod_{i=1}^{\ell} \#\mathrm{Hom}(\mathcal{H}_i \to \mathcal{G})$. Thus it remains to describe the algorithm for connected pattern graphs $\mathcal{H}$.

Let $\mathcal{H}$ be connected. A sequence of vertices $v_1, \ldots, v_k$ in a graph $F$ is a *traversal* if, for all $i \in \{1, \ldots, k-1\}$, the vertex $v_{i+1}$ is contained in $\{v_1, \ldots, v_i\} \cup \Gamma(\{v_1, \ldots, v_i\})$. Let $u_1, \ldots, u_k$ be an arbitrary traversal of $\mathcal{H}$ with $\{u_1, \ldots, u_k\} = V(\mathcal{H})$; the latter property can be satisfied since $\mathcal{H}$ is a connected graph with $k$ vertices. Note that if $f : V(\mathcal{H}) \to V(\mathcal{G})$ is a homomorphism from $\mathcal{H}$ to $\mathcal{G}$, then $f(u_1), \ldots, f(u_k)$ is a traversal in $\mathcal{G}$, and this correspondence is injective. Thus the algorithm computes the number of traversals $v_1, \ldots, v_k$ in $\mathcal{G}$ for which the mapping $f$ with $f(u_i) = v_i$ for all $i$ is a homomorphism from $\mathcal{H}$ to $\mathcal{G}$. This number is equal to $\#\mathrm{Hom}(\mathcal{H} \to \mathcal{G})$, which the algorithm seeks to compute.

Since the maximum degree of $G$ is $\Delta$, any set $S \subseteq V(\mathcal{G})$ satisfies $|\Gamma(S)| \leq \Delta|S|$. Thus there are at most $n \cdot (\Delta k + k)^{k-1}$ traversal sequences in $\mathcal{G}$, which can be generated in linear time in the number of such sequences. For each traversal sequence, verifying whether the sequence corresponds to a homomorphism takes time $O(k\Delta)$ (in the word-RAM model with incidence lists for $\mathcal{H}$ already prepared). Overall, we obtain a running time of $O(n \cdot k^k \cdot (\Delta+1)^k)$. ◀

▶ **Theorem 13.** *For all positive integers $\Delta$, there is a fixed-parameter tractable algorithm for #Induced-Coloured-Subgraph[$\Delta$] with time complexity $O(n \cdot k^{(2\Delta+3)\cdot k})$ for $n$-vertex coloured graphs $\mathcal{G}$ and $k$-vertex coloured graphs $\mathcal{H}$.*

**Proof.** Let $(\mathcal{H}, \mathcal{G})$ be an instance of #Induced-Coloured-Subgraph[$\Delta$], write $\mathcal{G} = (G, c)$ and $\mathcal{H} = (H, c')$, and let $k = |V(\mathcal{H})|$. Without loss of generality, suppose that the ranges of $c$ and $c'$ are $[q]$ for some positive integer $q \leq k$. Namely, if any vertices of $G$ receive colours not in the range of $c'$, then our algorithm may remove them without affecting $\#\mathrm{Ind}(\mathcal{H} \to \mathcal{G})$; if any vertices of $H$ receive colours not in the range of $c$, then $\#\mathrm{Ind}(\mathcal{H} \to \mathcal{G}) = 0$.

For coloured graphs $\mathcal{K}$ and $\mathcal{B}$, let $\#\mathrm{Surj}(\mathcal{K} \to \mathcal{B})$ be the number of vertex-surjective homomorphisms from $\mathcal{K}$ to $\mathcal{B}$, i.e., the number of those homomorphisms from $\mathcal{K}$ to $\mathcal{B}$ that contain all vertices of $\mathcal{B}$ in their image.

Let $S$ be the set of all $q$-coloured graphs $\mathcal{K}$ such that $\Delta(\mathcal{K}) \leq \Delta$ and, for some $t \in [k]$, $V(\mathcal{K}) = [t]$. Let $S'$ be a set of representatives of (coloured) isomorphism classes of $S$.

Let $\boldsymbol{x}$ be the vector indexed by $S'$ such that $\boldsymbol{x}_\mathcal{K} = \#\mathrm{Ind}(\mathcal{K} \to \mathcal{G})$ for all $\mathcal{K} \in S'$. This vector contains the number of induced subgraph copies of $\mathcal{H}$ in $\mathcal{G}$, but it also contains the number of subgraph copies of all other graphs in $S'$ in $\mathcal{G}$. Let $\boldsymbol{b}$ be the vector indexed by $S'$ such that $\boldsymbol{b}_\mathcal{K} = \#\mathrm{Hom}(\mathcal{K} \to \mathcal{G})$ for all $\mathcal{K} \in S'$; each entry of this vector can be computed via the algorithm of Lemma 12. Then we will show that $\boldsymbol{x}$ and $\boldsymbol{b}$ can be related to each other via an invertible matrix $A$ such that $A\boldsymbol{x} = \boldsymbol{b}$. By calculating $A$ and $\boldsymbol{b}$, we can then output $\#\mathrm{Ind}(\mathcal{H} \to \mathcal{G}) = (A^{-1}\boldsymbol{b})_\mathcal{H}$.

To elaborate on this linear relationship between induced subgraph and homomorphism numbers, let us first consider some arbitrary graph $\mathcal{K} \in S'$. By partitioning the homomorphisms from $\mathcal{K}$ to $\mathcal{G}$ according to their image, we have

$$\#\mathrm{Hom}(\mathcal{K} \to \mathcal{G}) = \sum_{\substack{X \subseteq V(\mathcal{G}) \\ |X| \leq k}} \#\mathrm{Surj}(\mathcal{K} \to \mathcal{G}[X]).$$

In the right-hand side sum, we can collect terms with isomorphic induced subgraphs $\mathcal{G}[X]$, since we clearly have $\#\mathrm{Surj}(\mathcal{K} \to \mathcal{B}) = \#\mathrm{Surj}(\mathcal{K} \to \mathcal{B}')$ if $\mathcal{B} \simeq \mathcal{B}'$. Hence, we obtain

$$\#\mathrm{Hom}(\mathcal{K} \to \mathcal{G}) = \sum_{\mathcal{K}' \in S'} \#\mathrm{Surj}(\mathcal{K} \to \mathcal{K}') \cdot \#\mathrm{Ind}(\mathcal{K}' \to \mathcal{G}). \tag{5}$$

Thus let $A$ be the matrix indexed by $S'$ with $A_{\mathcal{K},\mathcal{K}'} = \#\mathrm{Surj}(\mathcal{K} \to \mathcal{K}')$ for all $\mathcal{K}, \mathcal{K}' \in S'$. Then (5) implies that $A\boldsymbol{x} = \boldsymbol{b}$. (An uncoloured version of this linear system is folklore, and originally due to Lovász [15].)

We next prove that $A$ is invertible. Indeed, given $\mathcal{K}, \mathcal{K}' \in S'$, write $\mathcal{K} \lesssim \mathcal{K}'$ if $\mathcal{K}$ admits a vertex-surjective homomorphism to $\mathcal{K}'$. Since $\lesssim$ is a partial order, as is readily verified, it admits a topological ordering $\pi$. Permuting the rows and columns of $A$ to agree with $\pi$ does not affect the rank of $A$, and it yields an upper triangular matrix with non-zero diagonal entries, so it follows that $A$ is invertible.

The algorithm is now immediate. It first determines $S$ by listing all $q$-coloured graphs on at most $k$ vertices with at most $\lfloor \Delta k/2 \rfloor$ edges, then checking each one to see whether it satisfies the degree condition. It then determines $S'$ from $S$ by testing every pair of coloured graphs in $S$ for isomorphism (by brute force). It then determines each entry $A_{\mathcal{K},\mathcal{K}'}$ of $A$ (by brute force) by listing the vertex-surjective maps $\mathcal{K} \to \mathcal{K}'$. It then determines $\boldsymbol{b}$ by invoking Lemma 12 to compute each entry $\boldsymbol{b}_{\mathcal{K}} = \#\mathrm{Hom}(\mathcal{K} \to \mathcal{G})$ for $\mathcal{K} \in S'$. Finally, it outputs $\#\mathrm{Ind}(\mathcal{H} \to \mathcal{G}) = (A^{-1}\boldsymbol{b})_{\mathcal{H}}$. We defer the running time analysis to the full version. ◀

We note that the above algorithm can be generalised to any host graph class for which counting homomorphisms from (vertex-coloured) patterns with $k$ vertices has an $f(k) \cdot n^{O(1)}$ time algorithm. To this end, simply use this algorithm as a sub-routine instead of Lemma 12 in the algorithm constructed in the proof of Theorem 13. Examples for such classes of host graphs are planar graphs or, more generally, any graph class of bounded local treewidth [9].

Recent independent work by Patel and Regts [17] implicitly contains an algorithm for counting independent sets of size $k$ in graphs of maximum degree $\Delta$ in time $O(c^k n)$, where $c$ is a constant depending on $\Delta$. This implies Theorem 14(i).

▶ **Theorem 14.** *For all positive integers $\Delta$:*

**(i)** $\#$Size-BIS[$\Delta$] *has an algorithm with time complexity $O(|V(G)| \cdot k^{(2\Delta+3)k})$;*

**(ii)** $\#$Size-Left-BIS[$\Delta$] *has an algorithm with time complexity $O(|V(G)| \cdot \ell^{(2\Delta^2+8\Delta+4)\ell})$;*

**(iii)** $\#$Size-Left-Max-BIS[$\Delta$] *has an algorithm with time complexity $O(|V(G)| \cdot \ell^{(2\Delta^2+8\Delta+4)\ell})$.*

**Proof.** Part (i) of the result is immediate from Theorem 13, since $\#$Size-BIS[$\Delta$] is a special case of $\#$Induced-Coloured-Subgraph[$\Delta$] (taking $\mathcal{G}$ to be monochromatic and $\mathcal{H}$ to be a monochromatic independent set of size $k$).

For any bipartite graph $G = (U, V, E)$ with degree at most $\Delta$ and any non-negative integers $\ell$ and $r$, let $N_{\ell,r}(G)$ be the number of sets $X \subseteq U$ with $|X| = \ell$ and $|\Gamma(X)| = r$. Let $N'_{\ell,r}(G)$ be the number of pairs of sets $X \subseteq U$, $Y \subseteq V$ such that $|X| = \ell$, $|Y| = r$ and $Y \subseteq \Gamma(X)$. Then we have

$$N_{\ell,r}(G) = N'_{\ell,r}(G) - \sum_{i=r+1}^{\Delta\ell} \binom{i}{r} N_{\ell,i}(G). \tag{6}$$

For any bipartite graph $J = (U_J, V_J, E_J)$, we define the corresponding 2-colouring by $c_J(v) = 1$ for all $v \in U_J$ and $c_J(v) = 2$ for all $v \in V_J$. We define the corresponding coloured graph by $\phi(J) = ((U_J \cup V_J, \{\{u, v\} \mid (u, v) \in E_J\}), c_J)$. Let $S_{\ell,r}$ be the set of all bipartite graphs $J = (U_J, V_J, E_J)$ with $U_J = [\ell]$, $V_J = \{\ell + 1, \ldots, \ell + r\}$, degree at most $\Delta$ and no isolated vertices in $V_J$. Let $\mathcal{S}_{\ell,r}$ be the corresponding set of coloured graphs, and let $\mathcal{S}'_{\ell,r}$ be a set of representatives of (coloured) isomorphism classes in $\mathcal{S}_{\ell,r}$. Then $N'_{\ell,r}(G) = \sum_{\mathcal{K} \in \mathcal{S}'_{\ell,r}} \#\mathrm{Ind}(\mathcal{K} \to \phi(G))$, and hence by (6) we have

$$N_{\ell,r}(G) = \sum_{\mathcal{K} \in \mathcal{S}'_{\ell,r}} \#\mathrm{Ind}(\mathcal{K} \to \phi(G)) - \sum_{i=r+1}^{\Delta\ell} \binom{i}{r} N_{\ell,i}(G). \tag{7}$$

Now suppose that $(G, \ell)$ is an instance of #Size-Left-BIS[$\Delta$]. Then we have

$$\mathrm{IS}_{\ell\text{-left}}(G) = \sum_{\substack{X \subseteq U \\ |X| = \ell}} 2^{|V| - |\Gamma(X)|} = \sum_{0 \le r \le \Delta\ell} N_{\ell,r}(G) 2^{|V| - r}. \tag{8}$$

To compute $N_{\ell,\Delta\ell}(G), \ldots, N_{\ell,0}(G)$, our algorithm applies (7). For each $r \in \{\Delta\ell, \ldots, 0\}$, it determines the $\#\mathrm{Ind}(\mathcal{K} \to \phi(G))$ terms of (7) using the #Induced-Coloured-Subgraph[$\Delta$] algorithm of Theorem 13, and the remaining terms of (7) recursively with dynamic programming. Finally, it computes $\mathrm{IS}_{\ell\text{-left}}(G)$ using (8). Thus part (ii) of the result follows, except for the running time analysis which we defer to the full version.

Finally, suppose that $(G, \ell)$ is an instance of #Size-Left-Max-BIS[$\Delta$]. Let $\mu = \min\{r \mid N_{\ell,r}(G) \neq 0\}$, and note that $\mathrm{IS}_{\ell\text{-left-max}}(G) = N_{\ell,\mu}(G)$. As above, our algorithm determines $N_{\ell,\Delta\ell}(G), \ldots, N_{\ell,0}(G)$ using (7), and thereby determines and outputs $N_{\ell,\mu}(G)$. The overall running time is again $O(|V(G)| \cdot \ell^{(2\Delta^2+8\Delta+4)\ell})$, so part (iii) of the result follows. ◀

## References


1. V. Arvind and V. Raman. *Approximation Algorithms for Some Parameterized Counting Problems*, pages 453–464. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.
2. Jin-Yi Cai, Andreas Galanis, Leslie Ann Goldberg, Heng Guo, Mark Jerrum, Daniel Stefankovic, and Eric Vigoda. #bis-hardness for 2-spin systems on bipartite bounded degree graphs in the tree non-uniqueness region. *J. Comput. Syst. Sci.*, 82(5):690–711, 2016. doi:10.1016/j.jcss.2015.11.009.
3. R. Curticapean, H. Dell, and D. Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proc. STOC 2017*, pages 210–213, 2017.
4. M. Cygan, F. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, Berlin Heidelberg, 2015.
5. R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer Publishing Company, Incorporated, 2013.
6. Martin E. Dyer, Leslie Ann Goldberg, Catherine S. Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004. doi:10.1007/s00453-003-1073-y.
7. J. Flum and M. Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004.
8. J. Flum and M. Grohe. *Parameterized complexity theory*. Springer-Verlag, Berlin Heidelberg, 2006.
9. M. Frick. Generalized model-checking over locally tree-decomposable classes. *Theory of Computing Systems*, 37(1):157–191, 2004.

**10** Andreas Galanis, Daniel Stefankovic, Eric Vigoda, and Linji Yang. Ferromagnetic potts model: Refined #bis-hardness and related results. *SIAM J. Comput.*, 45(6):2004–2065, 2016. doi:10.1137/140997580.

**11** L. A. Goldberg and M. Jerrum. A complexity classification of spin systems with an external field. *Proceedings of the National Academy of Sciences*, 112(43):13161–13166, 2015.

**12** R. Impagliazzo and R. Paturi. On the complexity of $k$-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

**13** S. Janson, T. Łuczak, and A. Rucinski. *Random Graphs*. John Wiley & Sons, Inc., 2000.

**14** Jingcheng Liu and Pinyan Lu. FPTAS for #bis with degree bounds on one side. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 549–556. ACM, 2015. doi:10.1145/2746539.2746598.

**15** László Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. American Mathematical Society, 2012. URL: http://www.ams.org/bookstore-getitem/item=COLL-60.

**16** M. Müller. Randomized approximations of parameterized counting problems. In *Proc. IWPEC 2006*, pages 50–59, Berlin, Heidelberg, 2006. Springer-Verlag.

**17** Viresh Patel and Guus Regts. Deterministic polynomial-time approximation algorithms for partition functions and graph polynomials. *CoRR*, abs/1607.01167, 2016. arXiv:1607.01167.

**18** M. Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996.

**19** M. Xia, P. Zhang, and W. Zhao. Computational complexity of counting problems on 3-regular planar graphs. *Theoretical Computer Science*, 384(1):111–125, 2007.

# The Dominating Set Problem in Geometric Intersection Graphs*†

**Mark de Berg[1], Sándor Kisfaludi-Bak[2], and Gerhard Woeginger[3]**

1 **Department of Mathematics and Computer Science, TU Eindhoven, Eindhoven, the Netherlands**
2 **Department of Mathematics and Computer Science, TU Eindhoven, Eindhoven, the Netherlands**
3 **Department of Computer Science, RWTH Aachen University, Aachen, Germany**

## Abstract

We study the parameterized complexity of dominating sets in geometric intersection graphs.

- In one dimension, we investigate intersection graphs induced by translates of a fixed pattern Q that consists of a finite number of intervals and a finite number of isolated points. We prove that Dominating Set on such intersection graphs is polynomially solvable whenever Q contains at least one interval, and whenever Q contains no intervals and for any two point pairs in Q the distance ratio is rational. The remaining case where Q contains no intervals but does contain an irrational distance ratio is shown to be NP- complete and contained in FPT (when parameterized by the solution size).
- In two and higher dimensions, we prove that Dominating Set is contained in W[1] for intersection graphs of semi-algebraic sets with constant description complexity. This generalizes known results from the literature. Finally, we establish W[1]-hardness for a large class of intersection graphs.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, F.1.3 Complexity Measures and Classes

**Keywords and phrases** dominating set, intersection graph, W-hierarchy

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.14

## 1 Introduction

A *dominating set* in a graph $G = (V, E)$ is a subset $D \subseteq V$ of vertices such that every node in $V$ is either contained in $D$ or has some neighbor in $D$. The decision version of the dominating set problem asks for a given graph $G$ and a given integer $k$, whether $G$ admits a dominating set of size at most $k$. Dominating set is a popular and classic problem in algorithmic graph theory. It has been studied extensively for various graph classes; we only mention that it is polynomially solvable on interval graphs, strongly chordal graphs, permutation graphs and co-comparability graphs and that it is NP-complete on bipartite graphs, comparability graphs, and split graphs. We refer the reader to the book [9] by Hales, Hedetniemi and Slater for lots of comprehensive information on dominating sets.

* This research was supported by the Netherlands Organization for Scientific Research (NWO) under project no. 024.002.003.
† See full version at `https://arxiv.org/abs/1709.05182`


Dominating set is also a model problem in parameterized complexity, as it is one of the few natural problems known to be W[2]-complete (with the solution size $k$ as natural parameterization); see [5]. In the parameterized setting, dominating set on a concrete graph class typically is either in P, FPT, W[1]-complete, or W[2]-complete. (Note that the problem cannot be on higher levels of the W-hierarchy, as it is W[2]-complete on general graphs.)

In this paper we study the dominating set problem on geometric intersection graphs: Every vertex in $V$ corresponds to a geometric object in $\mathbb{R}^d$, and there is an edge between two vertices if and only if the corresponding objects intersect. Well-known graph classes that fit into this model are interval graphs and unit disk graphs. In $\mathbb{R}^1$, Chang [3] has given a polynomial time algorithm for dominating set in interval graphs and Fellows, Hermelin, Rosamond and Vialette [6] have proven W[1]-completeness for 2-interval graphs (where the geometric objects are pairs of intervals). In $\mathbb{R}^2$, Marx [10] has shown that dominating set is W[1]-hard for unit disk graphs as well as for unit square graphs. For unit square graphs the problem is furthermore known to be contained in W[1] [10], whereas for unit disk graphs this was previously not known.

## Our contribution

We investigate the dominating set problem on intersection graphs of 1- and 2-dimensional objects, thereby shedding more light on the borderlines between P and FPT and W[1] and W[2].

For 1-dimensional intersection graphs, we consider the following setting. There is a fixed *pattern* $Q$, which consists of a finite number of points and a finite number of closed intervals (specified by their endpoints). The objects corresponding to the vertices in the intersection graph simply are a finite number of translates of this fixed pattern $Q$. More formally, for a real number $x$ we define $Q(x) := x + Q$ to be the pattern $Q$ translated by $x$, and for the input $\{x_1, \ldots, x_n\}$, we consider the intersection graph defined by the objects $\{x_1 + Q, \ldots, x_n + Q\}$. The class of unit interval graphs arises by choosing $Q = [0, 1]$. Our model of computation is the word RAM model, where real numbers are restricted to a field $K$ which is a finite extension of the rationals.

▶ Remark (Machine representation of numbers). As finite extensions of $\mathbb{Q}$ are finite dimensional vector spaces over $\mathbb{Q}$, there exists a basis $b_1, \ldots, b_k$ with $k = [K : \mathbb{Q}]$, so that any real $x \in K$ is representable in the form $x = q_1 b_1 + q_2 b_2 + \cdots + q_k b_k$ for some $q_1, \ldots, q_k \in \mathbb{Q}$. As $k$ is fixed, any arithmetic operation that takes $O(1)$ steps on the rationals will also take $O(1)$ steps on elements of $K$.

We define the *distance ratio* of two point pairs $(x_1, x_2), (x_3, x_4) \in \mathbb{R} \times \mathbb{R}$ as $\frac{|x_1 - x_2|}{|x_3 - x_4|}$. We derive the following complexity classification for $Q$-Intersection Dominating Set.

▶ **Theorem 1.** *$Q$-Intersection Dominating Set has the following complexity:*

**(i)** *It is in* P *if the pattern $Q$ contains at least one interval.*

**(ii)** *It is in* P *if the pattern $Q$ does not contain any intervals, and if for any two point pairs in $Q$ the distance ratio is rational.*

**(iii)** *It is* NP*-complete and in* FPT *if pattern $Q$ is a finite point set which has at least one irrational distance ratio.*

In the final version we show that any graph can be obtained as a 1-dimensional pattern intersection graph for a suitable choice of pattern $Q$. Consequently $Q$-Intersection Dominating Set is W[2]-complete if the pattern $Q$ is part of the input.

For 2-dimensional intersection graphs, our results are inspired by a question that was not resolved in [10]: *"Is dominating set on unit disk graphs contained in* W[1]*?"* We answer this question affirmatively (and thereby fully settle the complexity status of this problem). Our result is in fact far more general: We show that dominating set is contained in W[1] whenever the geometric objects in the intersection graph come from a family of semi-algebraic sets that can be described by a constant number of parameters. We also show that this restriction to shapes of constant-complexity is crucial, as dominating set is W[2]-hard on intersection graphs of convex polygons with a polynomial number of vertices. On the negative side, we generalize the W[1]-hardness result of Marx [10] by showing that for any non-trivial simple polygonal pattern $Q$, the corresponding version of dominating set is W[1]-hard.

The full version of this paper is available as a preprint [4].

## 2 1-dimensional patterns

In this section, we study the $Q$-Intersection Dominating Set problem in $\mathbb{R}^1$. If $Q$ contains an unbounded interval, then all translates are intersecting; the intersection graph is a clique and the minimum dominating set is a single vertex. In what follows, we assume that all intervals in $Q$ are bounded. We define the *span* of $Q$ to be the distance between its leftmost and rightmost point. We prove Theorem 1 by studying each claim separately.

▶ **Lemma 2.** *$Q$-Intersection Dominating Set can be solved in $O(n^{6w+4})$ time if $Q$ contains at least one interval, where $w$ is the ratio of the span of $Q$ and the length of the longest interval in $Q$.*

Note that since $Q$ is a fixed pattern, the value of $w$ does not depend on the input size and so Lemma 2 implies Theorem 1(i). We translate $Q$ so that its leftmost endpoint lies at the origin, and we rescale $Q$ so that its longest interval has length 1. Consider an intersection graph $\mathcal{G}$ of a set of translates of $Q$. The vertices of $\mathcal{G}$ are $Q(x_i)$ for the given values $x_i$. We call $x_i$ the *left endpoint* of $Q_i$. Let $+$ also denote the Minkowski sum of sets: $A + B = \{a + b \mid a \in A, b \in B\}$. If $A$ or $B$ is a singleton, then we omit the braces, i.e., we let $a + B$ denote $\{a\} + B$. In order to prove Lemma 2, we need the following lemma first.

▶ **Lemma 3.** *Let $D \subseteq V(\mathcal{G})$ be a minimum dominating set and let $X(D)$ be the set of left endpoints corresponding to the patterns in $D$. Then for all $y \in \mathbb{R}$ it holds that $|X(D) \cap [y, y + w]| \leqslant 3w$.*

**Proof.** We prove this lemma first for unit interval graphs (where $Q$ consists of a single interval). The following observation is easy to prove.

▶ **Observation 4.** *In any unit interval graph there is a minimum dominating set whose intervals do not overlap.*

Notice that the lemma immediately follows from this claim in case of unit interval graphs since then $|X(D) \cap [y, y + 1]| \leqslant 1 < 3 = 3w$. Let $Q$ be any other pattern, and suppose that $|X(D) \cap [y, y + w]| \geqslant 3w + 1$. The patterns starting in $[y, y + w]$ can only dominate patterns with a left endpoint in $[y - w, y + 2w]$, a window of width $3w$. Let $H$ be the set of patterns starting in $[y - w, y + 2w]$ (see Figure 1. Let $I$ be a unit interval of $Q$, and let $U$ the set of unit intervals that are the translates of $I$ in the patterns of $H$. Notice that $X(U)$ is a point set that is also in a window of length $3w$. By the claim above, we know that the interval graph $\mathcal{G}(U)$ defined by $U$ has a dominating set that contains non-overlapping intervals, in particular, a dominating set $D_U$ of size at most $3w$. Since $\mathcal{G}(U)$ corresponds to a spanning

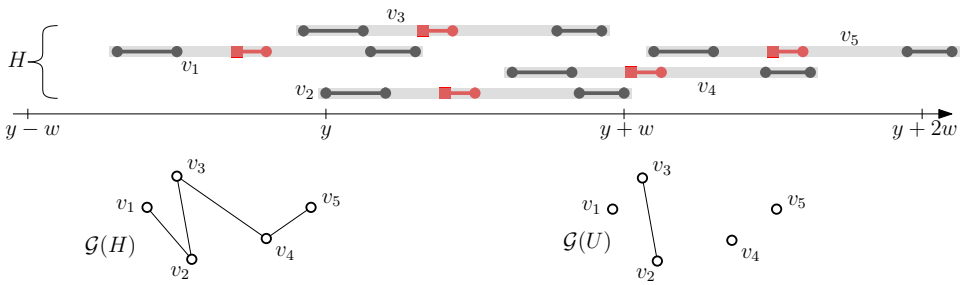



**Figure 1** Patterns in a window $[y - w, y + 2w]$. Intervals of $U$ are red.

subgraph of $\mathcal{G}(H)$, the patterns $D_U^H$ corresponding to $D_U$ in $H$ form a dominating set of $\mathcal{G}(H)$. Thus, $(D \setminus H) \cup D_U^H$ is a dominating set of our original graph that is smaller than $D$, which contradicts the minimality of $D$. ◀

We can now move on to the proof of Lemma 2.

**Proof.** We give a dynamic programming algorithm. We translate our input so that the left endpoint of the leftmost pattern is 0. Moreover, we can assume that the graph induced by our pattern is connected, since we can apply the algorithm to each connected component separately. The connectivity implies that the left endpoint of the rightmost pattern is at most $(n-1)w$. Let $0 < k \leqslant n$ be an integer and let $\mathcal{G}(k)$ be the intersection graph induced by the patterns with left endpoints in $[0, kw]$. Let $\mathcal{I}(k)$ be the set of input patterns with left endpoints in $[(k-1)w, kw)$ and let $S \subseteq \mathcal{I}(k)$. Let $A(k, S)$ be the size of a minimum dominating set $D$ of $\mathcal{G}(k)$ for which $D \cap \mathcal{I}(k) = S$. By Lemma 3 it follows that $|S| \leqslant 3w$.

The following recursion holds for $A(k, S)$ if we define $A(0, S) := 0$:

$$A(k, S) = \min \left\{ A(k-1, S') + |S| \;\middle|\; S' \subset \mathcal{I}(k-1),\ |S'| \leqslant 3w,\ S \cup S' \text{ dominates } \mathcal{I}(k) \right\}.$$

The inequality "$\leqslant$" is easy to see, we are only minimizing over the sizes of feasible dominating sets of $\mathcal{G}(k)$. For the other direction ("$\geqslant$"), Lemma 3 implies that there is a minimum dominating set containing at most $3w$ left endpoints from both $\mathcal{I}(k-1)$ and $\mathcal{I}(k)$, therefore its size is $A(k-1, S') + |S|$ for some $S' \subset \mathcal{I}(k-1)$, $|S'| \leqslant 3w$ that together with $S$ dominates $\mathcal{I}(k)$. The number of subproblems for a fixed value of $k$ is $\sum_{j=0}^{3w} \binom{n}{j} = O(n^{3w})$; thus the number of subproblems is $O(n^{3w+1})$. Computing the value of a subproblem requires looking at $O(n^{3w+1})$ potential subsets $S'$, and $O(n^2)$ time is sufficient to check whether $S \cup S'$ dominates $\mathcal{I}(k)$. Overall, the running time of our algorithm is $O(n^{6w+4})$. ◀

▶ **Lemma 5.** *If $Q$ is a point pattern so that the distance ratios of any two point pairs of $Q$ are rational, then $Q$-*Intersection Dominating Set *can be solved in polynomial time.*

**Proof.** By shifting and rescaling, we may assume without loss of generality that the leftmost point in $Q$ is in the origin and that all points in $Q$ have integer coordinates. (Note that this could not be done if the pattern contained an irrational distance ratio.) We define a new pattern $Q'$ that results from $Q$ by replacing point 0 by the interval $[0, 1/3]$.

Now consider an intersection graph whose vertices are associated with $x_i + Q$ where $x_1 \leq x_2 \leq \cdots \leq x_n$. We assume without loss of generality that the graph is connected and that all $x_i$ are integers. It can be seen that the intersection graph does not change, if every object $x_i + Q$ is replaced by the object $x_i + Q'$. Since pattern $Q'$ contains the interval $[0, 1/3]$, we may simply apply Lemma 2 to compute the optimal dominating set in polynomial time. ◀

▶ **Lemma 6.** *If $Q$ is a point pattern that contains two point pairs with an irrational distance ratio, then $Q$-*Intersection Dominating Set *is* NP*-complete.*

**Proof.** The containment in NP is trivial; we show the hardness by reducing from dominating set on induced triangular grid graphs. (These are finite induced subgraphs of the triangular grid, which is the graph with vertex set $V = \mathbb{Z}^2$ and edge set $E = \big\{\big((a,b),(a+\alpha,b+\beta)\big) : |\alpha| \leqslant 1, |\beta| \leqslant 1, \alpha \neq \beta\big\}$.) The NP-hardness of dominating set in induced triangular grid graphs is proven in the final version. Note that the dominating set problem is known to be NP-hard on induced grid graphs, but this does not imply the hardness on triangular grids, because triangular grid graphs are not a superclass of grid graphs.

We show that the infinite triangular grid can be realized as a $Q$-intersection graph, where the $Q$-translates are in a bijection with the vertices of the triangular grid. Therefore, any induced triangular grid graph is realized as the intersection graph of the $Q$-translates corresponding to its vertices.

Rescale $Q$ so that it has span 1. It cannot happen that all the points are rational, because it would make all distance ratios rational as well. Let $x^* \in Q$ be the smallest irrational point. Let $a \in \mathbb{Z}$, and consider the intersection of the translate $ax^* + Q$ with the set $\mathbb{Z} + Q$. We claim that this intersection is non-empty only for a finite number of values $a \in \mathbb{Z}$. Suppose the opposite. Since $Q$ is a finite pattern, there must be a pair $z, z' \in Q$ such that $ax^* + z = b + z'$ has infinitely many solutions $(a,b) \in \mathbb{Z}^2$. In particular, there are two solutions $(a_1,b_1)$ and $(a_2,b_2)$ such that $a_1 \neq a_2$ and $b_1 \neq b_2$. Subtracting the two equations we get $(a_1 - a_2)x^* = b_1 - b_2$, which implies $x^* = \frac{b_1-b_2}{a_1-a_2}$. This is a contradiction since $x$ is irrational.

Let $y^* = a'x^*$, where $a'$ is the largest value $a$ for which $ax^* + Q$ intersects $\mathbb{Z} + Q$. It follows that $\big\{j \in \mathbb{Z} \,\big|\, (jy^* + Q) \cap (\mathbb{Z} + Q) \neq \emptyset\big\} = \big\{-1, 0, 1\big\}$.

Consider the intersection graph induced by the sets $\big\{jy^* + k + Q \,\big|\, (j,k) \in \mathbb{Z}^2\big\}$. The above shows that a fixed translate $jy^* + k + Q$ is not intersected by the translates $(j+\alpha)y^* + (k+\beta) + Q$ if $|\alpha| \geq 2$. It is easy to see that $|\beta| \geq 2$ does not lead to an intersection either. Also note that $\alpha = \beta = \pm 1$ does not give an intersection; however all the remaining cases are intersecting, i.e., if

$$(\alpha, \beta) \in \big\{(-1,0),(-1,1),(0,-1),(0,0),(0,1),(1,-1),(1,0)\big\}$$

then $(j+\alpha)y^* + (k+\beta) + Q$ intersects $jy^* + k + Q$. Thus, the intersection graph induced by $\big\{jy^* + k + Q \,\big|\, (j,k) \in \mathbb{Z}^2\big\}$ is a triangular grid. ◀

▶ **Lemma 7.** *If $Q$ is a point pattern that has point pairs with an irrational distance ratio, then $Q$-*Intersection Dominating Set *has an* FPT *algorithm parameterized by solution size.*

**Proof.** In polynomial time, we can remove all duplicate translates, since a minimum dominating set contains at most one of these objects, and any minimum dominating set of the resulting graph is a dominating set of the original graph. Suppose our pattern consists of $t$ points. In the duplicate-free graph, point $i$ of the pattern translate may intersect point $j$ of another translate, for some $i \neq j$, so the maximum degree is $t^2 - t$. Therefore we are looking for a dominating set in a graph of bounded degree. Hence, a straightforward branching approach gives an FPT algorithm: choose any undominated vertex $v$; either $v$ or one of its at most $t^2 - t$ neighbors is in the dominating set, so we can branch $t^2 - t + 1$ ways. If all vertices are dominated after choosing $k$ vertices, then we have found a solution. This branching algorithm has depth $k$, with linear time required at each branching, so the total running time is $O\big(t^{2k}(|V| + |E|)\big)$. ◀

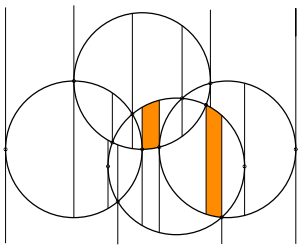


**Figure 2** Two faces of a vertical decomposition.

▶ Remark. In our handling of the problem, the pattern was part of the problem definition. Making the pattern part of the input leads to an NP-complete problem: Lemma 6 can be adapted to this scenario. If we also allow the size of the pattern to depend on the input, then the problem is W[2] complete (when parameterized by solution size): see the final version, where we show that for any graph $G$ there is a finite pattern whose translates can produce $G$ as an intersection graph.

We propose the following problem for further study, where the pattern depends on the input, but has fixed size.

**Open question.** Let $Q$ be the pattern defined by two unit intervals on a line at distance $\ell$. Is there an FPT algorithm (either with parameter $k$ or $k + \ell$) on intersection graphs defined by translates of $Q$, that can decide if such a graph has a dominating set of size $k$? It can be shown that this problem is NP-complete, and Theorem 10 below shows that it is contained in W[1].

## 3 Higher dimensional shapes: W[1] vs. W[2]

In this section we show that dominating set on intersection graphs of 2-dimensional objects is contained in W[1] if the shapes have a constant size description. First, we demonstrate the method on unit disk graphs, and later we state a much more general version where the shapes are semi-algebraic sets. In order to show containment, it is sufficient to give a non-deterministic algorithm that has an FPT time deterministic preprocessing, then a nondeterministic phase where the number of steps is only dependent on the parameter. More precisely, we use the following theorem.

▶ **Theorem 8** ([7]). *A parameterized problem is in* W[1] *if and only if it can be computed by a nondeterministic RAM program accepting the input that*

1. *performs at most $f(k)p(n)$ deterministic steps;*
2. *uses at most $f(k)p(n)$ registers;*
3. *contains numbers smaller than $f(k)p(n)$ in any register at any time;*
4. *for any run on any input, the nondeterministic steps are among the last $g(k)$ steps.*

*Here $n$ is the size of the input, $k$ is the parameter, $p$ is a polynomial and $f, g$ are computable functions. The non-deterministic instruction is defined as guessing a natural number between 0 and the value stored in the first register, and storing it in the first register. Acceptance of an input is defined as having a computation path that accepts.*

▶ **Theorem 9.** *The dominating set problem on unit disk graphs is contained in* W[1].

**Proof.** Let $P$ be the set of centers of the unit disks that form the input instance. For a subset $D \subseteq P$, let $\mathcal{C}_2(D)$ and $\mathcal{D}_2(D)$ be the set of circles and disks of radius 2, respectively, centered at the points of $D$. (Note that $D$ is a dominating set if and only if $\bigcup \mathcal{D}_2(D)$, the union of the disks in $\mathcal{D}_2(D)$, covers all points in $P$.) Shoot a vertical ray up and down from each of the $O(k^2)$ intersection points between the circles of $\mathcal{C}_2(D)$, and also from the leftmost and rightmost point of each circle. Each ray is continued until it hits a circle (or to infinity). The arrangement we get is a *vertical decomposition* [2] (see Fig. 2). Each face of this decomposition is defined by at most four circles. This is not only true for the 2-dimensional faces, but also for the 1-dimensional faces (the edges of the arrangement) and 0-dimensional faces (the vertices). We consider the faces to be relatively open, so that they are pairwise disjoint.

In our preprocessing phase, we compute all potential faces of a vertical decomposition of any subset $D \subseteq P$ by looking at all 4-tuples of circles from $\mathcal{C}_2(P)$. We create a lookup table that contains the number of input points covered by each potential face in $O(n^4)$ time.

Next, using nondeterminism we guess $k$ integers, representing the points of our solution; let $D$ be this point set. The rest of the algorithm deterministically checks if $D$ is dominating. We need to compute the vertical decomposition of $\mathcal{C}_2(D)$; this can be done in $O(k^2)$ time [2]. Finally, for each of the $O(k^2)$ resulting faces of $\bigcup \mathcal{D}_2(D)$, we can get the number of input points covered from the lookup table in constant time. We accept if these numbers sum to $n$. By Theorem 8 we can thus conclude that dominating set on unit disk graphs is in W[1]. ◀

In order to state the general version of this theorem, we introduce semi-algebraic sets. A *semi-algebraic set* is a subset of $\mathbb{R}^d$ obtained from a finite number of sets of the form $\{x \in \mathbb{R}^d \mid g(x) \geqslant 0\}$, where $g$ is a $d$-variate polynomial with integer coefficients, by Boolean operations (unions, intersections, and complementations). Let $\Gamma_{d,\Delta,s}$ denote the family of all semi-algebraic sets in $\mathbb{R}^d$ defined by at most $s$ polynomial inequalities of degree at most $\Delta$ each. If $d, \Delta, s$ are all constants, we refer to the sets in $\Gamma_{d,\Delta,s}$ as constant-complexity semi-algebraic sets.

Let $\mathcal{F}$ be a family of constant complexity semi-algebraic sets in $\mathbb{R}^d$ that can be specified using $t$ parameters $a_1, \ldots, a_t$. If the expressions defining $\mathcal{F}$ are also polynomials in terms of the parameters, then we call $\mathcal{F}$ a *$t$-parameterized family of semi- algebraic sets*. For example, the family of all balls in the $\mathbb{R}^3$ is a 4-parameterized family of semi- algebraic sets, since any ball can be specified using an inequality of the form $(x_1-a_1)^2+(x_2-a_2)^2+(x_3-a_3)^2-a_4^2 \leqslant 0$. As another example, the family of all triangles in the plane is a 6-parameterized algebraic set, since any triangle is the intersection of three half-planes, and any half- plane can be specified using two parameters.

We only give a sketch of the proof, the complete proof can be found in the final version.

▶ **Theorem 10.** *Let $\mathcal{F}$ be a $t$-parameterized family of semi-algebraic sets, for some constant $t$. Then dominating set is in* W[1] *for intersection graphs defined by $\mathcal{F}$.*

**Proof sketch of Theorem 10.** By definition, any set $S \in \mathcal{F}$ can be specified using $t$ parameters $a_1, \ldots, a_t$. Thus we can represent $S$ by the point $\mathbf{p}(S) := (a_1, \ldots, a_t)$ in $\mathbb{R}^t$. Conversely, for a point $(a_1, \ldots, a_t) \in \mathbb{R}^t$, let $S(a_1, \ldots, a_t)$ be the corresponding semi-algebraic set. Now we define, for any set $S \in \mathcal{F}$, a region $\mathbf{R}(S)$ as follows:

$$\mathbf{R}(S) := \{(a_1, \ldots, a_t) \in \mathbb{R}^t : S(a_1, \ldots, a_t) \cap S \neq \emptyset\}.$$

Thus for any two sets $S_1, S_2 \in \mathcal{F}$ we have that $S_1 \cap S_2 \neq \emptyset$ if and only if $\mathbf{p}(S_1) \in \mathbf{R}(S_2)$.

Now consider a set $\mathcal{S} \subset \mathcal{F}$ of $n$ sets from the family $\mathcal{F}$. We proceed in a similar way as in the proof of Theorem 9, where the sets $\mathbf{R}(S)$ for $S \in \mathcal{S}$ play the same role as the radius-2

disks in that proof. Consider any subset $\mathcal{D} \subseteq \mathcal{S}$, and note that $\mathcal{D}$ is a dominating set if and only if $\bigcup_{S\in\mathcal{D}} \mathbf{R}(S)$ contains the point set $\{\mathbf{p}(S)|S \in \mathcal{S}\}$.

Now we can decompose the arrangement defined by $\{\mathbf{R}(S) : S \in \mathcal{D}\}$ into polynomially many cells using a so-called *cylindrical decomposition* [1]; note that such a decomposition is made possible by the fact that the regions $\mathbf{R}(S)$ are semi-algebraic. (This decomposition plays the role of the vertical decomposition in the proof for unit disks.) Each cell of the cylindrical decomposition is defined by at most $t'$ regions $\mathbf{R}(S)$, for some $t' = O(1)$. Thus, for each subset of at most $t'$ regions $\mathbf{R}(S)$, we compute all cells that arise in the cylindrical decomposition of the subset. The number of possible cells is polynomial in $n$.

In the preprocessing phase, we compute for each possible cell the number of points $\mathbf{p}(S)$ contained in it, and store the results in a lookup table. The next phase of the algorithm is the same as for unit disks: we guess a solution, compute the cells in the cylindrical decomposition of the corresponding arrangement, and check using the lookup table if the guessed solution is a dominating set. ◀

### W[1]-hardness for simple polygon translates

We generalize a proof by Marx [10] for the W[1]-hardness of dominating set in unit square/unit disk graphs. Our result is based on the observation that many 2-dimensional shapes share the crucial properties of unit squares when it comes to the type of intersections needed for this specific construction. We prove the following theorem.

▶ **Theorem 11.** *The dominating set problem is* W[1]*-hard for intersection graphs of the translates of a simple polygon in* $\mathbb{R}^2$*.*

Our proof uses the same global strategy as Marx's proof [10] for the W[1]-hardness of dominating set for intersection graphs of squares. (We give an overview of the proof in the final version.) To apply this proof strategy, all we need to prove is that the family of shapes for which we want to prove W[1]-hardness has a certain property, as defined next.

We say that a shape $S \subseteq \mathbb{R}^2$ is *square-like* if there are two base vectors $\mathbf{b}_1$ and $\mathbf{b}_2$ and for any $n$ there are two small offset vectors $\mathbf{u}_1 = \mathbf{u}_1(n)$ and $\mathbf{u}_2 = \mathbf{u}_2(n)$ with the following properties. Define $S(i, j) := S + i\mathbf{u}_1 + j\mathbf{u}_2$ for all $-n^2 \leqslant i, j \leqslant n^2$, and consider the set $\mathcal{K} := \{S(i, j) : -n^2 \leqslant i, j \leqslant n^2\}$. Note that $\mathcal{K}$ consists of $(2n^2 + 1)^2$ translated copies of $S$ whose reference points from a $(2n^2 + 1) \times (2n^2 + 1)$ grid. Also note that $S = S(0, 0)$. For the shape $S$ to be square-like, we require the following properties:

- $\mathcal{K}$ is a clique in the intersection graph, i.e.,
  for all $-n^2 \leqslant i,j \leqslant n^2$ we have: $S \cap S(i, j) \neq \emptyset$.
- "Horizontal" neighbors intersect only when close:
  for all $-n^2 \leqslant j \leqslant n^2$ we have: $S \cap (\mathbf{b}_1 + S(i, j)) \neq \emptyset \iff i \leqslant 0$.
- "Vertical" neighbors intersect only when close:
  for all $-n^2 \leqslant i \leqslant n^2$ we have: $S \cap (\mathbf{b}_2 + S(i, j)) \neq \emptyset \iff j \leqslant 0$.
- Distant copies of $\mathcal{K}$ are disjoint:
  for all $-n^2 \leqslant i,j,i',j' \leqslant n^2$ we have: $|k|+|\ell| \geqslant 2 \Rightarrow S(i, j) \cap (k\mathbf{b}_1 + \ell\mathbf{b}_2 + S(i', j')) = \emptyset$.

Moreover, we require that each of the vectors can be represented on $O(\log n)$ bits. It is helpful to visualize a square grid, with unit side lengths $\mathbf{b}_1$ and $\mathbf{b}_2$, where we place the centers of unit squares with small offsets compared to the grid points. We are requiring a very similar intersection structure here. See Figure 5 for an example of a good choice of vectors.

Since the above properties are sufficient for the construction given by Marx [10], we only need to prove the following theorem.

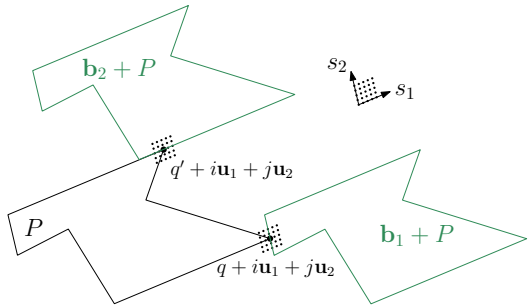



**Figure 3** A good choice of $\mathbf{b}_1, \mathbf{b}_2$ and offsets.

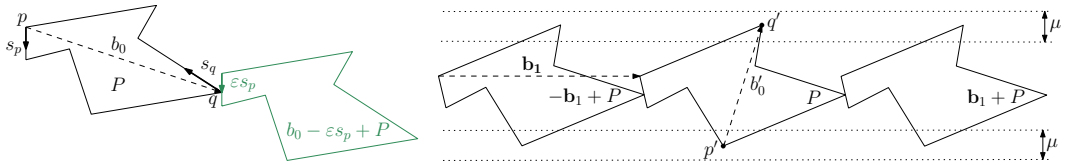



**Figure 4** Left: Defining $\mathbf{b}_1$. Right: Defining $\mathbf{b}_2$.

▶ **Theorem 12.** *Every simple polygon is square-like.*

Before giving a formal proof, we give a short overview. First, we would like to define a "horizontal" direction, i.e., a good vector $\mathbf{b}_1$. A natural choice would be to select a diameter of the polygon (see $b_0$ on the left of Figure 4), however that would result in $S$ and $\mathbf{b}_1 + S$ intersecting each other at vertices. That would pose a severe restriction on the offset vectors; therefore, we use a perturbed version of a diameter, making sure that the intersection of $S$ and $\mathbf{b}_1 + S$ is realized by a polygon side from at least one party. The direction of this polygon side also defines a suitable direction of the offset vector $\mathbf{u}_2$: because of the second property, choosing $\mathbf{u}_2$ to be parallel to this direction ensures the independence with respect to the choice of $j$.

Next, we define the other base vector $\mathbf{b}_2$. This definition is based on laying out an infinite sequence of translates horizontally next to each other (right side of Figure 4). We want a translate of this sequence to touch the original sequence in a "non-intrusive" way: small perturbations of $\mathbf{b}_2 + S$ should only intersect $S$, but stay disjoint from $\mathbf{b}_1 + S$ or $-\mathbf{b}_1 + S$. This is fairly easy to achieve; again with a small perturbation of our first candidate vector we can also ensure that the intersection between $\mathbf{b}_2 + S$ and $S$ is not a vertex-vertex intersection. Finally, a suitable direction for the offset vector $\mathbf{u}_1$ is given by the polygon side taking part in the intersection between $\mathbf{b}_2 + S$ and $S$.

**Proof of Theorem 12.** Let $P$ be a simple polygon, and let $p$ and $q$ be two endpoints of a diameter of $P$. Let $b_0 = q - p$. Since $P$ is a polygon, both $p$ and $q$ are vertices. Let $s_p$ and $s_q$ be unit vectors in the direction of the side of $P$ that follows vertex $p$ and $q$ in the counter-clockwise order. Let $\varepsilon > 0$ be a small number to be specified later. Consider the intersection of $P$ and the translate $b_0 + \varepsilon s_q + P$. If $\varepsilon$ is small enough, then depending on the angle of $s_p$ and $s_q$, this intersection is either the point $b_0 + \varepsilon s_q$, or part of the side with direction $s_q$, or it is an intersection of positive area. The left of Figure 4 illustrates the third case. In the first case, let $\mathbf{b}_1 = b_0 + \varepsilon s_p$; in the second and third case, let $\mathbf{b}_1 = b_0 - \varepsilon s_q$. Furthermore, let $s_1 = \mathbf{b}_1 - b_0$. We will later use $s_1$ to define the offset vector $\mathbf{u}_2$.

Imagine that $\mathbf{b}_1$ is the horizontal direction, and consider the set $P_\infty = \{k\mathbf{b}_1 + P \mid k \in \mathbb{Z}\}$ (right side of Figure 4). Its top and bottom boundary are infinite periodic polylines, with

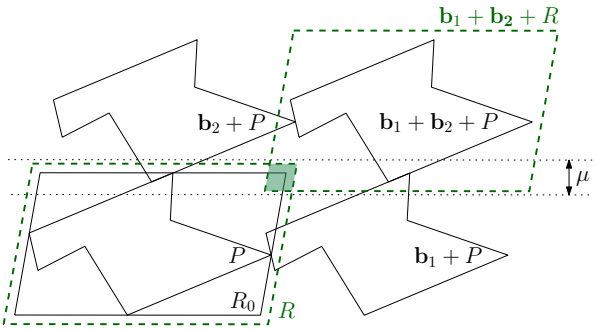



**Figure 5** Part of the grid $k\mathbf{b}_1+\ell\mathbf{b}_2+P$.

period length $|\mathbf{b}_1|$. Take a pair of horizontal lines that touch the top and bottom boundary. By manipulating $\varepsilon$ in the definition of $\mathbf{b}_1$, we can achieve a general position in the sense that both of these lines touch the respective boundaries exactly once in each period, moreover, there is a value $\mu$, such that there are no vertices other than the touching points in the $\frac{\mu}{2}$-neighborhood of the touching lines. Let $p'$ and $q'$ be vertices touched by the bottom and top lines inside $P$, and let $b'_0 = q' - p'$. Similarly as before, the direction of the sides following $p'$ and $q'$ counter-clockwise are denoted by $s_{p'}$ and $s_{q'}$. If the intersection of $P$ and the translate $b'_0 + \varepsilon s_{q'} + P$ has zero area, then let $\mathbf{b}_2 = b'_0 - \varepsilon s_{q'}$; otherwise, (if the area of the intersection is positive), let $\mathbf{b}_2 = b'_0 + \varepsilon s_{p'}$. We denote by $s_2$ the difference $\mathbf{b}_2 - b'_0$. If $s_2$ and $s_1$ are parallel, then we can define $s_2$ similarly, by replacing the sides $s_{p'}$ and $s_{q'}$ with the sides that follow $p'$ and $q'$ in clockwise direction. The new direction of $s_2$ will not be parallel to the old one, therefore it will not be parallel to $s_1$.

We need to choose the values of $\mathbf{u}_1$ and $\mathbf{u}_2$. Let $\mathbf{u}_1 = \frac{\varepsilon}{2n^2} s_2$ and let $\mathbf{u}_2 = \frac{\varepsilon}{2n^2} s_1$. We claim that if $\varepsilon$ is small enough, then $P$ is square-like for the vectors $\mathbf{b}_1, \mathbf{b}_2, \mathbf{u}_1, \mathbf{u}_2$. It is easy to check that for a small enough value of $\varepsilon$, the first condition is satisfied, namely that $P \cap P(i,j) \neq \emptyset$ for all $-n^2 \leqslant i, j \leqslant n^2$.

Next, we show that for $i \leqslant 0$, the intersection of $P$ and $\mathbf{b}_1 + P(i,j)$ is non-empty. Consider the small grid of points $q - i\mathbf{u}_1 - j\mathbf{u}_2$, $-n^2 \leqslant i, j \leqslant n^2$ (see Figure 3). This grid fits into a parallelogram whose sides are parallel to $s_2$ and $s_1$. Notice that if $\varepsilon$ is small enough, then $q - i\mathbf{u}_1 - j\mathbf{u}_2$ for all $-n^2 \leqslant i < 0$ and $-n^2 \leqslant j \leqslant n^2$ is contained in $\mathbf{b}_1 + P$, thus the intersection $P \cap (\mathbf{b}_1 + i\mathbf{u}_1 + j\mathbf{u}_2 + P)$ is non-empty if $i \leqslant 0$. Moreover, (if $\varepsilon$ is small enough), then no other type of intersection can happen by moving $\mathbf{b}_1 + P$ slightly: the only sides that can intersect $\mathbf{b}_1 + P(i,j)$ from $P$ are adjacent to $q$. Therefore, if $q$ is outside $\mathbf{b}_1 + P(i,j)$, then the intersection is empty – which is true for $i > 0$. A similar argument works for the intersection of $P$ and $\mathbf{b}_2 + P(i,j)$.

Let $R_0$ be a minimum area parallelogram containing $P$ whose sides are parallel to $\mathbf{b}_1$ and $\mathbf{b}_2$ (see Figure 5). Notice that the side lengths of this parallelogram are at most $|\mathbf{b}_1| + \varepsilon$ and $|\mathbf{b}_2| + \varepsilon$ respectively. Let $\bar{P} = \bigcup \mathcal{K} = \bigcup_{-n^2 \leqslant i,j \leqslant n^2} P(i,j)$. Notice that $\bar{P}$ is contained in the slightly larger rectangle $R$ that we get by extending all sides of $R_0$ by $2\varepsilon$.

Now consider the rectangle translates $k\mathbf{b}_1 + \ell\mathbf{b}_2 + R$. Since $\varepsilon$ is small enough, if either $k$ or $\ell$ is at least two then $R \cap (k\mathbf{b}_1 + \ell\mathbf{b}_2 + R) = \emptyset$, so specifically, $\bar{P}$ is disjoint from $k\mathbf{b}_1 + \ell\mathbf{b}_2 + \bar{P}$. It remains to show that $\bar{P}$ is disjoint from $k\mathbf{b}_1 + \ell\mathbf{b}_2 + \bar{P}$ if $|k| = |\ell| = 1$. Consider $\bar{P}$ and $\mathbf{b}_1 + \mathbf{b}_2 + \bar{P}$ for example. They could only intersect inside $R \cap (\mathbf{b}_1 + \mathbf{b}_2 + R)$; however, if $\varepsilon < \frac{\mu}{4}$, then this is contained in the $\mu$ wide horizontal strip defined earlier. By the definition of this strip, it also means that there is an intersection point $q$ that is within distance $O(\varepsilon)$ from both $q'$ and $\mathbf{b}_1 + \mathbf{b}_2 + p'$. This would mean that $|\mathbf{b}_1| = O(\varepsilon)$, and thus it can be avoided by choosing a small enough $\varepsilon$.

Finally, we note that all restrictions on the value of $\varepsilon$ are dependent on the polygon $P$ itself, thus the length of the short vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ is $\Omega(n^{-2})$, and a precision of $O(n^{-2})$ is sufficient for all the vectors, thus the vectors can be represented on $O(\log n)$ bits. ◀

We remark that it is fairly easy to further generalize the above theorem to other families of objects, we can allow objects with certain curved boundaries for example. A simple example of an object that is not square-like is a pair of perpendicular disjoint unit segments: for any choice of offset vectors, the set $\mathcal{K}$ does not form a clique (as required by the first property of square-like objects).

### W[2]-hardness for convex polygons

We conclude with the following hardness result; the reduction uses a basic geometric idea that has been used for hardness proofs before [8, 11]. Note the crucial difference between the setting in this theorem, where the polygons defining the intersection graph can be different and have description complexity dependent on $n$, versus the previous settings (where we had constant description complexity and some uniformity among the object descriptions).

▶ **Theorem 13.** *The dominating set problem is* W[2]*-hard for intersection graphs of convex polygons.*

**Proof.** A *split graph* is a graph that has a vertex set which can be partitioned into a clique $C$ and an independent set $I$. It was shown by Raman and Saurabh [12] that dominating set is W[2]-hard on split graphs. Thus it is sufficient to show that any split graph can be represented as the intersection graph of convex polygons.

Let $G = (C \cup I, E)$ be an arbitrary split graph. Let $Q'$ be a regular $2|I|$-gon and let $Q$ be the regular $I$-gon defined by every second vertex of $Q'$. Notice that $Q' \setminus Q$ consists of small triangles, any subset of which together with $Q$ forms a convex polygon.

The polygons corresponding to $I$ are small equilateral triangles, placed in the interior of each small triangle of $Q' \setminus Q$. The polygon corresponding to a vertex $v \in C$ whose neighborhood in $I$ is $N_I(v)$ is the union of $Q$ and the small triangles corresponding to the vertices of $N_I(v)$.

In this construction, the polygons corresponding to $C$ all intersect (they all contain $Q$), and the polygons corresponding to $I$ are all disjoint. Finally, for any pair of vertices $u \in C$ and $v \in I$ the polygon of $u$ contains the polygon of $v$ if and only if $uv \in E$. ◀

## 4 Conclusion

We have classified the parameterized complexity of dominating set in intersection graphs defined by sets of various types in $\mathbb{R}^1$ and $\mathbb{R}^2$. More precisely, in $\mathbb{R}^1$, we gave a classification for the case when the intersection graph is defined by the translates of a fixed pattern that consists of points and intervals that is independent of the input. In $\mathbb{R}^2$, we have identified a fairly large class of W[1]-complete instances, namely, if our intersection graph is defined by a subset of a constant description complexity family of semi-algebraic sets. Even though our results hold for a large class of geometric intersection graphs, there are still some open problems. In particular, the complexity of dominating set on the following types of intersections graphs is unknown.

- translates of a 1-dimensional pattern that contains two unit intervals at some distance $\ell$ (given by the input) (FPT vs. W[1]?)

- translates of a 2-dimensional pattern that contains two disjoint perpendicular unit intervals (FPT vs. W[1]?)
- $n$ translates of a regular $n$-gon (W[1] vs. W[2]?)

## References


1. Dennis S. Arnon, George E. Collins, and Scott McCallum. Cylindrical algebraic decomposition I: the basic algorithm. *SIAM J. Comput.*, 13(4):865–877, 1984. doi:10.1137/0213054.
2. Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.
3. 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.
4. Mark de Berg, Sándor Kisfaludi-Bak, and Gerhard Woeginger. The dominating set problem in geometric intersection graphs. *CoRR*, abs/1709.05182, 2017. URL: http://arxiv.org/abs/1709.05182.
5. 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.
6. Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009. doi:10.1016/j.tcs.2008.09.065.
7. Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.
8. Sariel Har-Peled. Being fat and friendly is not enough. *arXiv preprint arXiv:0908.2369*, 2009.
9. Teresa W. Haynes, Stephen T. Hedetniemi, and Peter J. Slater. *Domination in Graphs: Advanced Topics*. Pure and Applied Mathematics. Marcel Dekker, Inc., 1998.
10. Dániel Marx. Parameterized complexity of independence and domination on geometric graphs. 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 154–165. Springer, 2006. doi:10.1007/11847250_14.
11. Dániel Marx and Michał Pilipczuk. Optimal parameterized algorithms for planar facility location problems using Voronoi diagrams. *arXiv preprint arXiv:1504.05476*, 2015.
12. Venkatesh Raman and Saket Saurabh. Short cycles make $W$ -hard problems hard: FPT algorithms for $W$ -hard problems in graphs with no short cycles. *Algorithmica*, 52(2):203–225, 2008. doi:10.1007/s00453-007-9148-9.

# Tight Conditional Lower Bounds for Longest Common Increasing Subsequence*

**Lech Duraj**[†1], **Marvin Künnemann**[2], **and Adam Polak**[‡3]

1 **Theoretical Computer Science, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland**
duraj@tcs.uj.edu.pl

2 **Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany**
marvin@mpi-inf.mpg.de

3 **Theoretical Computer Science, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland**
polak@tcs.uj.edu.pl

## Abstract

We consider the canonical generalization of the well-studied Longest Increasing Subsequence problem to multiple sequences, called $k$-LCIS: Given $k$ integer sequences $X_1, \ldots, X_k$ of length at most $n$, the task is to determine the length of the longest common subsequence of $X_1, \ldots, X_k$ that is also strictly increasing. Especially for the case of $k = 2$ (called LCIS for short), several algorithms have been proposed that require quadratic time in the worst case.

Assuming the Strong Exponential Time Hypothesis (SETH), we prove a tight lower bound, specifically, that no algorithm solves LCIS in (strongly) subquadratic time. Interestingly, the proof makes no use of normalization tricks common to hardness proofs for similar problems such as LCS. We further strengthen this lower bound to rule out $\mathcal{O}\left((nL)^{1-\varepsilon}\right)$ time algorithms for LCIS, where $L$ denotes the solution size, and to rule out $\mathcal{O}\left(n^{k-\varepsilon}\right)$ time algorithms for $k$-LCIS. We obtain the same conditional lower bounds for the related Longest Common Weakly Increasing Subsequence problem.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** fine-grained complexity, combinatorial pattern matching, sequence alignments, parameterized complexity, SETH

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.15

## 1 Introduction

The longest common subsequence problem (LCS) and its variants are computational primitives with a variety of applications, which includes, e.g., uses as similarity measures for spelling correction [36, 42] or DNA sequence comparison [38, 5], as well as determining the differences of text files as in the UNIX DIFF utility [27]. LCS shares characteristics of both an easy and a hard problem: (*Easy*) A simple and elegant dynamic-programming algorithm computes an LCS of two length-$n$ sequences in time $\mathcal{O}\left(n^2\right)$ [42], and in many practical settings, certain properties of typical input sequences can be exploited to obtain faster, "tailored" solutions

* The full version of this paper is available at: http://arxiv.org/abs/1709.10075.
† Partially supported by Polish National Science Center grant 2016/21/B/ST6/02165.
‡ Partially supported by Polish Ministry of Science and Higher Education program *Diamentowy Grant*.


(e.g., [26, 28, 7, 37]; see also [13] for a survey). (*Hard*) At the same time, no polynomial improvements over the classical solution are known, thus exact computation may become infeasible for very long general input sequences. The research community has sought for a resolution of the question *"Do subquadratic algorithms for LCS exist?"* already shortly after the formalization of the problem [20, 4].

Recently, an answer conditional on the Strong Exponential Time Hypothesis (SETH; see Section 2 for a definition) could be obtained: Based on a line of research relating the satisfiability problem to quadratic-time problems [43, 40, 14, 3] and following a breakthrough result for Edit Distance [9], it has been shown that unless SETH fails, there is no (strongly) subquadratic-time algorithm for LCS [1, 15]. Subsequent work [2] strengthens these lower bounds to hold already under weaker assumptions and even provides surprising consequences of sufficiently strong polylogarithmic improvements.

Due to its popularity and wide range of applications, several variants of LCS have been proposed. This includes the heaviest common subsequence (HCS) [31], which introduces weights to the problem, as well as notions that constrain the structure of the solution, such as the longest common increasing subsequence (LCIS) [45], LCSk [12], constrained LCS [41, 19, 8], restricted LCS [25], and many other variants (see, e.g., [18, 6, 32]). Most of these variants are (at least loosely) motivated by biological sequence comparison tasks. To the best of our knowledge, in the above list, LCIS is the only LCS variant for which (1) the best known algorithms run in quadratic time in the worst case and (2) its definition does not include LCS as a special case (for such generalizations of LCS, the quadratic-time SETH hardness of LCS [1, 15] would transfer immediately). As such, it is open to determine whether there are (strongly) subquadratic algorithms for LCIS or whether such algorithms can be ruled out under SETH. The starting point of our work is to settle this question.

## 1.1 Longest Common Increasing Subsequence (LCIS)

The Longest Common Increasing Subsequence problem on $k$ sequences ($k$-LCIS) is defined as follows: Given integer sequences $X_1, \ldots, X_k$ of length at most $n$, determine the length of the longest sequence $Z$ such that $Z$ is a strictly increasing sequence of integers and $Z$ is a subsequence of each $X_i, i \in \{1, \ldots, k\}$. For $k = 1$, we obtain the well-studied longest increasing subsequence problem (LIS; we refer to [21] for an overview), which has an $\mathcal{O}(n \log n)$ time solution and a matching lower bound in the decision tree model [24]. The extension to $k = 2$, denoted simply as LCIS, has been proposed by Yang, Huang, and Chao [45], partially motivated as a generalization of LIS and by potential applications in bioinformatics. They obtained an $\mathcal{O}(n^2)$ time algorithm, leaving open the natural question whether there exists a way to extend the near-linear time solution for LIS to a near-linear time solution for multiple sequences.

Interestingly, already a classic connection between LCS and LIS combined with a recent conditional lower bound of Abboud, Backurs and Vassilevska Williams [1] yields a partial negative answer assuming SETH.

▶ **Observation 1** (Folklore reduction, implicit in [28], explicit in [31]). *After $\mathcal{O}(kn^2)$ time preprocessing, we can solve $k$-LCS by a single call to $(k-1)$-LCIS on sequences of length at most $n^2$.*

Note that by the above reduction, an $\mathcal{O}\left(n^{\frac{3}{2}-\varepsilon}\right)$ time LCIS algorithm would give an $\mathcal{O}\left(n^{3-2\varepsilon}\right)$ time algorithm for 3-LCS, which would refute SETH by a result of Abboud et al. [1].

▶ **Corollary 2.** *Unless SETH fails, there is no $\mathcal{O}\left(n^{\frac{3}{2}-\varepsilon}\right)$ time algorithm for LCIS for any constant $\varepsilon > 0$.*

While this rules out near-linear time algorithms, still an unsatisfying large polynomial gap between best upper and conditional lower bounds persists.

## 1.2 Our Results

Our first result is a tight SETH-based lower bound for LCIS.

▶ **Theorem 3.** *Unless SETH fails, there is no* $\mathcal{O}\left(n^{2-\varepsilon}\right)$ *time algorithm for LCIS for any constant* $\varepsilon > 0$.

We extend our main result in several directions.

### 1.2.1 Parameterized Complexity I: Solution Size

Subsequent work [17, 34] improved over Yang et al.'s algorithm when certain input parameters are small. Here, we focus particularly on the solution size, i.e., the length $L$ of the LCIS. Kutz et al. [34] provided an algorithm running in time $\mathcal{O}\left(nL \log\log n + n \log n\right)$. If $L$ is small compared to its worst-case upper bound of $n$, say $L = n^{\frac{1}{2}\pm o(1)}$, this algorithm runs in strongly subquadratic time. Interestingly, exactly for this case, the reduction from 3-LCS to LCIS of Observation 1 already yields a matching SETH-based lower bound of $(Ln)^{1-o(1)} = n^{\frac{3}{2}-o(1)}$. However, for smaller $L$, this reduction yields no lower bound at all and only a non-matching lower bound for larger $L$. We remedy this situation by the following result.[1]

▶ **Theorem 4.** *Unless SETH fails, there is no* $\mathcal{O}\left((nL)^{1-\varepsilon}\right)$ *time algorithm for LCIS for any constant* $\varepsilon > 0$. *This even holds restricted to instances with* $L = n^{\gamma\pm o(1)}$, *for* arbitrarily chosen $0 < \gamma \leqslant 1$.

### 1.2.2 Parameterized Complexity II: $k$-LCIS

For constant $k \geqslant 3$, we can solve $k$-LCIS in $\mathcal{O}\left(n^k \text{polylog}(n)\right)$ time [17, 34], or even $\mathcal{O}\left(n^k\right)$ time (see the appendix in the full version). While it is known that $k$-LCS cannot be computed in time $\mathcal{O}\left(n^{k-\varepsilon}\right)$ for any constant $\varepsilon > 0, k \geqslant 2$ unless SETH fails [1], this does not directly transfer to $k$-LCIS, since the reduction in Observation 1 is not tight. However, by extending our main construction, we can prove the analogous result.

▶ **Theorem 5.** *Unless SETH fails, there is no* $\mathcal{O}\left(n^{k-\varepsilon}\right)$ *time algorithm for* $k$*-LCIS for any constant* $k \geqslant 3$ *and* $\varepsilon > 0$.

### 1.2.3 Longest Common Weakly Increasing Subsequence (LCWIS)

We consider a closely related variant of LCIS called the Longest Common Weakly Increasing Subsequence ($k$-LCWIS): Here, given integer sequences $X_1, \ldots, X_k$ of length at most $n$, the task is to determine the longest *weakly increasing* (i.e. non-decreasing) integer sequence $Z$ that is a common subsequence of $X_1, \ldots, X_k$. Again, we write LCWIS as a shorthand for 2-LCWIS. Note that the seemingly small change in the notion of increasing sequence has a major impact on algorithmic and hardness results: Any instance of LCIS in which the input sequences are defined over a small-sized alphabet $\Sigma \subseteq \mathbb{Z}$, say $|\Sigma| = \mathcal{O}\left(n^{1/2}\right)$, can be solved in strongly subquadratic time $\mathcal{O}\left(nL \log n\right) = \mathcal{O}\left(n^{3/2} \log n\right)$ [34], by using the fact that $L \leqslant |\Sigma|$. In contrast, LCWIS is quadratic-time SETH hard already over slightly

[1] We mention in passing that a systematic study of the complexity of LCS in terms of such input parameters has been performed recently in [16].

superlogarithmic-sized alphabets [39]. We give a substantially different proof for this fact and generalize it to $k$-LCWIS.

▶ **Theorem 6.** *Unless SETH fails, there is no $\mathcal{O}\left(n^{k-\varepsilon}\right)$ time algorithm for $k$-LCWIS for any constant $k \geqslant 3$ and $\varepsilon > 0$. This even holds restricted to instances defined over an alphabet of size $|\Sigma| \leqslant f(n) \log n$ for any function $f(n) = \omega(1)$ growing arbitrarily slowly.*

## 1.3 Discussion, Outline and Technical Contributions

Apart from an interest in LCIS and its close connection to LCS, our work is also motivated by an interest in the *optimality of dynamic programming (DP) algorithms*[2]. Notably, many conditional lower bounds in P target problems with natural DP algorithms that are proven to be near-optimal under some plausible assumption (see, e.g., [14, 3, 9, 10, 1, 15, 11, 22, 33] and [44] for an introduction to the field). Even if we restrict our attention to problems that find optimal sequence alignments under some restrictions, such as LCS, Edit Distance and LCIS, the currently known hardness proofs differ significantly, despite seemingly small differences between the problem definitions. Ideally, we would like to classify the properties of a DP formulation which allow for matching conditional lower bounds.

One step in this direction is given by the *alignment gadget framework* [15]. Exploiting normalization tricks, this framework gives an abstract property of sequence similarity measures to allow for SETH-based quadratic lower bounds. Unfortunately, as it turns out, we cannot directly transfer the alignment gadget hardness proof for LCS to LCIS – some indication for this difficulty is already given by the fact that LCIS can be solved in strongly subquadratic time over sublinear-sized alphabets [34], while the LCS hardness proof already applies to binary alphabets. By collecting gadgetry needed to overcome such difficulties (that we elaborate on below), we hope to provide further tools to generalize more and more quadratic-time lower bounds based on SETH.

### 1.3.1 Technical Challenges

The known conditional lower bounds for global alignment problems such as LCS and Edit Distance work as follows. The reductions start from the quadratic-time SETH-hard Orthogonal Vectors problem (OV), that asks to determine, given two sets of $(0,1)$-vectors $\mathcal{U} = \{u_0, \ldots, u_{n-1}\}, \mathcal{V} = \{v_0, \ldots, v_{n-1}\} \subseteq \{0,1\}^d$ over $d = n^{o(1)}$ dimensions, whether there is a pair $i, j$ such that $u_i$ and $v_j$ are orthogonal, i.e., whose inner product $(u_i \cdot v_j) := \sum_{k=0}^{d-1} u_i[k] \cdot v_j[k]$ is 0 (over the integers). Each vector $u_i$ and $v_j$ is represented by a (normalized) vector gadget $\mathrm{VG}_{\mathrm{X}}(u_i)$ and $\mathrm{VG}_{\mathrm{Y}}(v_j)$, respectively. Roughly speaking, these gadgets are combined to sequences $X$ and $Y$ such that each candidate for an optimal alignment of $X$ and $Y$ involves locally optimal alignments between $n$ pairs $\mathrm{VG}_{\mathrm{X}}(u_i), \mathrm{VG}_{\mathrm{Y}}(v_j)$ – the optimal alignment exceeds a certain threshold if and only if there is an orthogonal pair $u_i, v_j$.

An analogous approach does not work for LCIS: Let $\mathrm{VG}_{\mathrm{X}}(u_i)$ be defined over an alphabet $\Sigma$ and $\mathrm{VG}_{\mathrm{X}}(u_{i'})$ over an alphabet $\Sigma'$. If $\Sigma$ and $\Sigma'$ overlap, then $\mathrm{VG}_{\mathrm{X}}(u_i)$ and $\mathrm{VG}_{\mathrm{X}}(u_{i'})$ cannot both be aligned in an optimal alignment without interference with each other. On the other hand, if $\Sigma$ and $\Sigma'$ are disjoint, then each vector $v_j$ should have its corresponding vector gadget $VG_{\mathrm{Y}}(v_j)$ defined over both $\Sigma$ and $\Sigma'$ to enable to align $\mathrm{VG}_{\mathrm{X}}(u_i)$ with $\mathrm{VG}_{\mathrm{Y}}(v_j)$ as well as $\mathrm{VG}_{\mathrm{X}}(u_{i'})$ with $\mathrm{VG}_{\mathrm{Y}}(v_j)$. The latter option drastically increases the size of vector gadgets.

[2] We refer to [46] for a simple quadratic-time DP formulation for LCIS.

Thus, we must define all vector gadgets over a common alphabet $\Sigma$ and make sure that *only a single pair* $\mathrm{VG}_X(u_i), \mathrm{VG}_Y(v_j)$ is aligned in an optimal alignment (in contrast with $n$ pairs aligned in the previous reductions for LCS and Edit Distance).

### 1.3.2 Technical Contributions and Proof Outline

Fortunately, a surprisingly simple approach works: As a key tool, we provide *separator sequences* $\alpha_0 \dots \alpha_{n-1}$ and $\beta_0 \dots \beta_{n-1}$ with the following properties: (1) for every $i, j \in \{0, \dots, n-1\}$ the LCIS of $\alpha_0 \dots \alpha_i$ and $\beta_0 \dots \beta_j$ has a length of $f(i+j)$, where $f$ is a linear function, and (2) $\sum_i |\alpha_i|$ and $\sum_j |\beta_j|$ are bounded by $n^{1+o(1)}$. Note that existence of such a gadget is somewhat unintuitive: condition (1) for $i = 0$ and $j = n-1$ requires $|\alpha_0| = \Omega(n)$, yet still the total length $\sum_i |\alpha_i|$ must not exceed the length of $|\alpha_0|$ significantly. Indeed, we achieve this by a careful inductive construction that generates such sequences with heavily varying block sizes $|\alpha_i|$ and $|\beta_j|$.

We apply these separator sequences as follows. We first define simple vector gadgets $\mathrm{VG}_X(u_i), \mathrm{VG}_Y(v_j)$ over an alphabet $\Sigma$ such that the length of an LCIS of $\mathrm{VG}_X(u_i)$ and $\mathrm{VG}_Y(v_j)$ is $d - (u_i \cdot v_j)$. Then we construct the separator sequences as above over an alphabet $\Sigma_<$ whose elements are strictly smaller than all elements in $\Sigma$. Furthermore, we create analogous separator sequences $\alpha'_0 \dots \alpha'_{n-1}$ and $\beta'_0 \dots \beta'_{n-1}$ which satisfy a property like (1) for all suffixes instead of prefixes, using an alphabet $\Sigma_>$ whose elements are strictly larger than all elements in $\Sigma$. Now, we define

$$X = \alpha_0 \mathrm{VG}_X(u_0) \alpha'_0 \dots \alpha_{n-1} \mathrm{VG}_X(u_{n-1}) \alpha'_{n-1},$$
$$Y = \beta_0 \mathrm{VG}_Y(v_0) \beta'_0 \dots \beta_{n-1} \mathrm{VG}_Y(v_{n-1}) \beta'_{n-1}.$$

As we will show in Section 3, the length of an LCIS of $X$ and $Y$ is $C - \min_{i,j}(u_i \cdot v_j)$ for some constant $C$ depending only on $n$ and $d$.

In contrast to previous such OV-based lower bounds, we use heavily varying separators (paddings) between vector gadgets.

## 1.4 Paper organization

After setting up conventions and introducing our hardness assumptions in Section 2, we give the main construction, i.e., the hardness of LCIS in Section 3. The proofs of Theorems 4, 5 and 6 can be found in the full version. We conclude with some open problems in Section 4.

# 2 Preliminaries

As a convention, we use capital or Greek letters to denote sequences over integers. Let $X, Y$ be integer sequences. We write $|X|$ for the length of $X$, $X[k]$ for the $k$-th element in the sequence $X$ ($k \in \{0, \dots, |X|-1\}$), and $X \circ Y = XY$ for the concatenation of $X$ and $Y$. We say that $Y$ is a subsequence of $X$ if there exist indices $0 \leqslant i_1 < i_2 < \dots < i_{|Y|} \leqslant |X| - 1$ such that $X[i_k] = Y[k]$ for all $k \in \{0, \dots, |Y|-1\}$. Given any number of sequences $X_1, \dots, X_k$, we say that $Y$ is a common subsequence of $X_1, \dots, X_k$ if $Y$ is a subsequence of each $X_i, i \in \{1, \dots, k\}$. $X$ is called strictly increasing (or weakly increasing) if $X[0] < X[1] < \dots < X[|X|-1]$ (or $X[0] \leqslant X[1] \leqslant \dots \leqslant X[|X|-1]$). For any $k$ sequences $X_1, \dots, X_k$, we denote by $\mathrm{lcis}(X_1, \dots, X_k)$ the length of their longest common subsequence that is strictly increasing.

All of our lower bounds hold assuming the Strong Exponential Time Hypothesis (SETH), introduced by Impagliazzo and Paturi [29, 30]. It essentially states that no exponential speed-up over exhaustive search is possible for the CNF satisfiability problem.

▶ **Hypothesis 7** (Strong Exponential Time Hypothesis (SETH)). *There is no $\varepsilon > 0$ such that for all $d \geqslant 3$ there is an $\mathcal{O}\left(2^{(1-\varepsilon)n}\right)$ time algorithm for $d$-SAT.*

This hypothesis implies tight hardness of the $k$-Orthogonal Vectors problem ($k$-OV), which will be the starting point of our reductions: Given $k$ sets $\mathcal{U}_1, \ldots, \mathcal{U}_k \subseteq \{0,1\}^d$, each with $|\mathcal{U}_i| = n$ vectors over $d = n^{o(1)}$ dimensions, determine whether there is a $k$-tuple $(u_1, \ldots, u_k) \in \mathcal{U}_1 \times \cdots \times \mathcal{U}_k$ such that $\sum_{\ell=0}^{d-1} \prod_{i=1}^{k} u_i[\ell] = 0$. By exhaustive enumeration, it can be solved in time $\mathcal{O}\left(n^k d\right) = n^{k+o(1)}$. The following conjecture is implied by SETH by the well-known split-and-list technique of Williams [43] (and the sparsification lemma [30]).

▶ **Hypothesis 8** ($k$-OV conjecture). *Let $k \geqslant 2$. There is no $\mathcal{O}\left(n^{k-\varepsilon}\right)$ time algorithm for $k$-OV, with $d = \omega(\log n)$, for any constant $\varepsilon > 0$.*

For the special case of $k = 2$, which we simply denote by OV, we obtain the following weaker conjecture.

▶ **Hypothesis 9** (OV conjecture). *There is no $\mathcal{O}\left(n^{2-\varepsilon}\right)$ time algorithm for OV, with $d = \omega(\log n)$, for any constant $\varepsilon > 0$. Equivalently, even restricted to instances with $|\mathcal{U}_1| = n$ and $|\mathcal{U}_2| = n^\gamma$, $0 < \gamma \leqslant 1$, there is no $\mathcal{O}\left(n^{1+\gamma-\varepsilon}\right)$ time algorithm for OV, with $d = \omega(\log n)$, for any constant $\varepsilon > 0$.*

A proof of the folklore equivalence of the statements for equal and unequal set sizes can be found, e.g., in [15].

## 3 Main Construction: Hardness of LCIS

In this section, we prove quadratic-time SETH hardness of LCIS, i.e., prove Theorem 3. We first introduce an *inflation* operation, which we then use to construct our separator sequences. After defining simple vector gadgets, we show how to embed an Orthogonal Vectors instance using our vector gadgets and separator sequences.

### 3.1 Inflation

We begin by introducing the inflation operation, which simulates weighing the sequences.

▶ **Definition 10.** For a sequence $A = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ of integers we define:

$$\operatorname{inflate}(A) = \langle 2a_0 - 1, 2a_0, 2a_1 - 1, 2a_1, \ldots, 2a_{n-1} - 1, 2a_{n-1} \rangle .$$

▶ **Lemma 11.** *For any two sequences $A$ and $B$, $\operatorname{lcis}(\operatorname{inflate}(A), \operatorname{inflate}(B)) = 2 \cdot \operatorname{lcis}(A, B)$.*

**Proof.** Let $C$ be the longest common increasing subsequence of $A$ and $B$. Observe that $\operatorname{inflate}(C)$ is a common increasing subsequence of $\operatorname{inflate}(A)$ and $\operatorname{inflate}(B)$ of length $2 \cdot |C|$, thus $\operatorname{lcis}(\operatorname{inflate}(A), \operatorname{inflate}(B)) \geqslant 2 \cdot \operatorname{lcis}(A, B)$.

Conversely, let $\bar{A}$ denote $\operatorname{inflate}(A)$ and $\bar{B}$ denote $\operatorname{inflate}(B)$. Let $\bar{C}$ be the longest common increasing subsequence of $\bar{A}$ and $\bar{B}$. If we divide all elements of $\bar{C}$ by 2 and round up to the closest integer, we end up with a weakly increasing sequence. Now, if we remove duplicate elements to make this sequence strictly increasing, we obtain $C$, a common increasing subsequence of $A$ and $B$. At most 2 distinct elements may become equal after division by 2 and rounding, therefore $C$ contains at least $\lceil \operatorname{lcis}(\bar{A}, \bar{B})/2 \rceil$ elements, so $2 \cdot \operatorname{lcis}(A, B) \geqslant \operatorname{lcis}(\bar{A}, \bar{B})$. This completes the proof. ◀

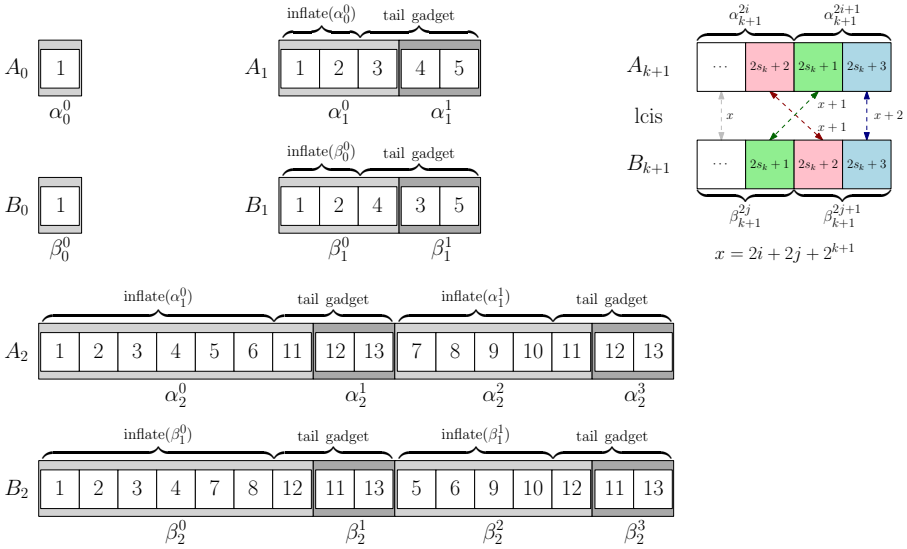



**Figure 1** Initial steps of inductive construction of separator sequences (left), and intuition behind tail gadgets (right).

## 3.2 Separator sequences

Our goal is to construct two sequences $A$ and $B$ which can be split into $n$ blocks, i.e. $A = \alpha_0\alpha_1 \dots \alpha_{n-1}$ and $B = \beta_0\beta_1 \dots \beta_{n-1}$, such that the length of the longest common increasing subsequence of the first $i$ blocks of $A$ and the first $j$ blocks of $B$ equals $i + j$, up to an additive constant. We call $A$ and $B$ *separator sequences*, and use them later to separate vector gadgets in order to make sure that only one pair of gadgets may interact with each other at the same time.

We construct the separator sequences inductively. For every $k \in \mathbb{N}$, the sequences $A_k$ and $B_k$ are concatenations of $2^k$ blocks (of varying sizes), $A_k = \alpha_k^0\alpha_k^1 \dots \alpha_k^{2^k-1}$ and $B_k = \beta_k^0\beta_k^1 \dots \beta_k^{2^k-1}$. Let $s_k$ denote the largest element of both sequences. As we will soon observe, $s_k = 2^{k+2} - 3$.

The construction works as follows: for $k = 0$, we can simply set $A_0$ and $B_0$ as one-element sequences $\langle 1 \rangle$. We then construct $A_{k+1}$ and $B_{k+1}$ inductively from $A_k$ and $B_k$ in two steps. First, we inflate both $A_k$ and $B_k$, then after each (now inflated) block we insert 3-element sequences, called *tail gadgets*, $\langle 2s_k + 2, 2s_k + 1, 2s_k + 3 \rangle$ for $A_{k+1}$ and $\langle 2s_k + 1, 2s_k + 2, 2s_k + 3 \rangle$ for $B_{k+1}$. Formally, we describe the construction by defining blocks of the new sequences. For $i \in \{0, 1, \dots, 2^k - 1\}$,

$$\alpha_{k+1}^{2i} = \text{inflate}(\alpha_k^i) \circ \langle 2s_k + 2 \rangle, \qquad \alpha_{k+1}^{2i+1} = \langle 2s_k + 1, 2s_k + 3 \rangle,$$
$$\beta_{k+1}^{2i} = \text{inflate}(\beta_k^i) \circ \langle 2s_k + 1 \rangle, \qquad \beta_{k+1}^{2i+1} = \langle 2s_k + 2, 2s_k + 3 \rangle.$$

Note that the symbols appearing in tail gadgets do not appear in the inflated sequences. The largest element of both new sequences $s_{k+1}$ equals $2s_k + 3$, and solving the recurrence gives indeed $s_k = 2^{k+2} - 3$.

Now, let us prove two useful properties of the separator sequences.

▶ **Lemma 12.** $|A_k| = |B_k| = \left(\frac{3}{2}k + 1\right) \cdot 2^k = \mathcal{O}\left(k2^k\right)$.

**Proof.** Observe that $|A_{k+1}| = 2|A_k| + 3 \cdot 2^k$. Indeed, to obtain $A_{k+1}$ first we double the size of $A_k$ and then add 3 new elements for each of the $2^k$ blocks of $A_k$. Solving the recurrence completes the proof. The same reasoning applies to $B_k$. ◀

▶ **Lemma 13.** *For every* $i, j \in \{0, 1, \ldots, 2^k - 1\}$, $\operatorname{lcis}(\alpha_k^0 \ldots \alpha_k^i, \beta_k^0 \ldots \beta_k^j) = i + j + 2^k$.

**Proof.** The proof is by induction on $k$. Assume the statement is true for $k$ and let us prove it for $k+1$.

**The "$\geqslant$" direction.** First, consider the case when both $i$ and $j$ are even. Observe that $\operatorname{inflate}(\alpha_k^0 \ldots \alpha_k^{i/2})$ and $\operatorname{inflate}(\beta_k^0 \ldots \beta_k^{j/2})$ are subsequences of $\alpha_{k+1}^0 \ldots \alpha_{k+1}^i$ and $\beta_{k+1}^0 \ldots \beta_{k+1}^j$, respectively. Thus, using the induction hypothesis and inflation properties,

$$\begin{aligned}\operatorname{lcis}(\alpha_{k+1}^0 \ldots \alpha_{k+1}^i, \beta_{k+1}^0 \ldots \beta_{k+1}^j) &\geqslant \operatorname{lcis}(\operatorname{inflate}(\alpha_k^0 \ldots \alpha_k^{i/2}), \operatorname{inflate}(\beta_k^0 \ldots \beta_k^{j/2})) = \\ &= 2 \cdot \operatorname{lcis}(\alpha_k^0 \ldots \alpha_k^{i/2}, \beta_k^0 \ldots \beta_k^{j/2}) = 2 \cdot (i/2 + j/2 + 2^k) = i + j + 2^{k+1}.\end{aligned}$$

If $i$ is odd and $j$ is even, refer to the previous case to get a common increasing subsequence of $\alpha_{k+1}^0 \ldots \alpha_{k+1}^{i-1}$ and $\beta_{k+1}^0 \ldots \beta_{k+1}^j$ of length $i - 1 + j + 2^{k+1}$ consisting only of elements less than or equal to $2s_k$, and append the element $2s_k + 1$ to the end of it. Analogously, for $i$ even and $j$ odd, take such an LCIS of $\alpha_{k+1}^0 \ldots \alpha_{k+1}^i$ and $\beta_{k+1}^0 \ldots \beta_{k+1}^{j-1}$, and append $2s_k + 2$. Finally, for both $i$ and $j$ odd, take an LCIS of $\alpha_{k+1}^0 \ldots \alpha_{k+1}^{i-1}$ and $\beta_{k+1}^0 \ldots \beta_{k+1}^{j-1}$, and append $2s_k + 1$ and $2s_k + 3$.

**The "$\leqslant$" direction.** We proceed by induction on $i + j$. Fix $i$ and $j$, and let $L$ be a longest common increasing subsequence of $\alpha_{k+1}^0 \ldots \alpha_{k+1}^i$ and $\beta_{k+1}^0 \ldots \beta_{k+1}^j$.

If the last element of $L$ is less than or equal to $2s_k$, $L$ is in fact a common increasing subsequence of $\operatorname{inflate}(\alpha_k^0 \ldots \alpha_k^{\lfloor i/2 \rfloor})$ and $\operatorname{inflate}(\beta_k^0 \ldots \beta_k^{\lfloor j/2 \rfloor})$, thus, by the induction hypothesis and inflation properties, $|L| \leqslant 2 \cdot (\lfloor i/2 \rfloor + \lfloor j/2 \rfloor + 2^k) \leqslant i + j + 2^{k+1}$.

The remaining case is when the last element of $L$ is greater than $2s_k$. In this case, consider the second-to-last element of $L$. It must belong to some blocks $\alpha_{k+1}^{i'}$ and $\beta_{k+1}^{j'}$ for $i' \leqslant i$ and $j' \leqslant j$, and we claim that $i = i'$ and $j = j'$ cannot hold simultaneously: by construction of separator sequences, if blocks $\alpha_{k+1}^i$ and $\beta_{k+1}^j$ have a common element larger than $2s_k$, then it is the only common element of these two blocks. Therefore, it cannot be the case that both $i = i'$ and $j = j'$, because the last two elements of $L$ would then be located in $\alpha_{k+1}^i$ and $\beta_{k+1}^j$. As a consequence, $i' + j' < i + j$, which lets us apply the induction hypothesis to reason that the prefix of $L$ omitting its last element is of length at most $i' + j' + 2^{k+1}$. Therefore, $|L| \leqslant 1 + i' + j' + 2^{k+1} \leqslant i + j + 2^{k+1}$, which completes the proof. ◀

Observe that if we reverse the sequences $A_k$ and $B_k$ along with changing all elements to their negations, i.e. $x$ to $-x$, we obtain sequences $\hat{A}_k$ and $\hat{B}_k$ such that $\hat{A}_k$ splits into $2^k$ blocks $\hat{\alpha}_k^0 \ldots \hat{\alpha}_k^{2^k-1}$, $\hat{B}_k$ splits into $2^k$ blocks $\hat{\beta}_k^0 \ldots \hat{\beta}_k^{2^k-1}$, and

$$\operatorname{lcis}(\hat{\alpha}_k^i \ldots \hat{\alpha}_k^{2^k-1}, \hat{\beta}_k^j \ldots \hat{\beta}_k^{2^k-1}) = 2 \cdot (2^k - 1) - i - j + 2^k. \tag{1}$$

Finally, observe that we can add any constant to all elements of the sequences $A_k$ and $B_k$ (as well as $\hat{A}_k$ and $\hat{B}_k$) without changing the property stated in Lemma 13 (and its analogue for $\hat{A}_k$ and $\hat{B}_k$, i.e. Equation (1)).

## 3.3 Vector gadgets

Let $\mathcal{U} = \{u_0, \ldots, u_{n-1}\}$ and $\mathcal{V} = \{v_0, \ldots, v_{n-1}\}$ be two sets of $d$-dimensional $(0,1)$-vectors.

For $i \in \{0, 1, \ldots, n-1\}$ let us construct the vector gadgets $U_i$ and $V_i$ as $2d$-element sequences, by defining, for every $p \in \{0, 1, \ldots, d-1\}$,

$$(U_i[2p-1], U_i[2p]) = \begin{cases} (2p-1, 2p) & \text{if } u_i[p] = 0, \\ (2p-1, 2p-1) & \text{if } u_i[p] = 1, \end{cases}$$

$$(V_i[2p-1], V_i[2p]) = \begin{cases} (2p, 2p-1) & \text{if } v_i[p] = 0, \\ (2p, 2p) & \text{if } v_i[p] = 1. \end{cases}$$

Observe that at most one of the elements $2p-1$ and $2p$ may appear in the LCIS of $U_i$ and $V_j$, and it happens if and only if $u_i[p]$ and $v_j[p]$ are not both equal to one. Therefore, $\operatorname{lcis}(U_i, V_j) = d - (u_i \cdot v_j)$, and, in particular, $\operatorname{lcis}(U_i, V_j) = d$ if and only if $u_i$ and $v_j$ are orthogonal.

## 3.4 Final construction

To put all the pieces together, we plug vector gadgets $U_i$ and $V_j$ into the separator sequences from Section 3.2, obtaining two sequences whose LCIS depends on the minimal inner product of vectors $u_i$ and $v_j$. We provide a general construction of such sequences, which will be useful for proving further results in the full version of the paper.

▶ **Lemma 14.** *Let $X_0, X_1, \ldots, X_{n-1}$, $Y_0, Y_1, \ldots, Y_{n-1}$ be integer sequences such that none of them has an increasing subsequence longer than $\delta$. Then there exist sequences $X$ and $Y$ of length $\mathcal{O}(\delta \cdot n \log n) + \sum |X_i| + \sum |Y_j|$, constructible in linear time, such that:*

$$\operatorname{lcis}(X, Y) = \max_{i,j} \operatorname{lcis}(X_i, Y_j) + C$$

*for a constant $C$ that only depends on $n$ and $\delta$ and is $\mathcal{O}(n\delta)$.*

**Proof.** We can assume that $n = 2^k$ for some positive integer $k$, adding some dummy sequences if necessary. Recall the sequences $A_k$, $B_k$, $\hat{A}_k$ and $\hat{B}_k$ constructed in Section 3.2. Let $A, B, \hat{A}, \hat{B}$ be the sequences obtained from $A_k, B_k, \hat{A}_k, \hat{B}_k$ by applying inflation $\lceil \log_2 \delta \rceil$ times (thus increasing their length by a factor of $\ell = 2^{\lceil \log_2 \delta \rceil} \geqslant \delta$). Each of these four sequences splits into (now inflated) blocks, e.g. $A = \alpha_0 \alpha_1 \ldots \alpha_{n-1}$, where $\alpha_i = \operatorname{inflate}^{\lceil \log_2 \delta \rceil}(\alpha_k^i)$.

We subtract from $A$ and $B$ a constant large enough for all their elements to be smaller than all elements of every $X_i$ and $Y_j$. Similarly, we add to $A'$ and $B'$ a constant large enough for all their elements to be larger than all elements of every $X_i$ and $Y_j$. Now, we can construct the sequences $X$ and $Y$ as follows:

$$X = \alpha_0 X_0 \hat{\alpha}_0 \alpha_1 X_1 \hat{\alpha}_1 \ldots \alpha_{n-1} X_{n-1} \hat{\alpha}_{n-1},$$
$$Y = \beta_0 Y_0 \hat{\beta}_0 \beta_1 Y_1 \hat{\beta}_1 \ldots \beta_{n-1} Y_{n-1} \hat{\beta}_{n-1}.$$

We claim that

$$\operatorname{lcis}(X, Y) = \ell \cdot (4n-2) + M, \text{ where } M = \max_{i,j} \operatorname{lcis}(X_i, Y_j).$$

Let $X_i$ and $Y_j$ be the pair of sequences achieving $\operatorname{lcis}(X_i, Y_j) = M$. Recall that $\operatorname{lcis}(\alpha_0 \ldots \alpha_i, \beta_0 \ldots \beta_j) = \ell \cdot (i + j + n)$, with all the elements of this common subsequence preceding the elements of $X_i$ and $Y_j$ in $X$ and $Y$, respectively, and being smaller than them. In the same way $\operatorname{lcis}(\hat{\alpha}_i \ldots \hat{\alpha}_{n-1}, \hat{\beta}_j \ldots \hat{\beta}_{n-1}) = \ell \cdot (2 \cdot (n-1) - (i+j) + n)$ with all the elements of LCIS being greater and appearing later than those of $X_i$ and $Y_j$. By

concatenating these three sequences we obtain a common increasing subsequence of $X$ and $Y$ of length $\ell \cdot (4n-2) + M$.

We defer the simple remainder of the proof, i.e., proving $\text{lcis}(X,Y) \leqslant \ell \cdot (4n-2) + M$ to the full version of the paper. ◀

**Proof of Theorem 3.** Let $\mathcal{U} = \{u_0, \ldots, u_{n-1}\}$, $\mathcal{V} = \{v_0, \ldots, v_{n-1}\}$ be two sets of binary vectors in $d$ dimensions. In Section 3.3 we constructed vector gadgets $U_i$ and $V_j$, for $i, j \in \{0, 1, \ldots, n-1\}$, such that $\text{lcis}(U_i, V_j) = d - (u_i \cdot v_j)$. To these sequences we apply Lemma 14, with $\delta = 2d$, obtaining sequences $X$ and $Y$ of length $\mathcal{O}(n \log n \text{poly}(d))$ such that $\text{lcis}(X,Y) = C + d - \min_{i,j}(u_i \cdot v_j)$ for a constant $C$. This reduction, combined with an $\mathcal{O}\left(n^{2-\varepsilon}\right)$ time algorithm for LCIS, would yield an $\mathcal{O}\left(n^{2-\varepsilon}\text{polylog}(n)\text{poly}(d)\right)$ algorithm for OV, refuting Hypothesis 9 and, in particular, SETH. ◀

## 4 Conclusion and Open Problems

We prove a tight quadratic lower bound for LCIS, ruling out strongly subquadratic-time algorithms under SETH. It remains open whether LCIS admits mildly subquadratic algorithms, such as the Masek-Paterson algorithm for LCS [35]. Furthermore, we give tight SETH-based lower bounds for $k$-LCIS.

For the related variant LCWIS that considers weakly increasing sequences, strongly subquadratic-time algorithms are ruled out under SETH for slightly superlogarithmic alphabet sizes ([39] and Theorem 6). On the other hand, for binary and ternary alphabets, even linear time algorithms exist [34, 23]. Can LCWIS be solved in time $\mathcal{O}\left(n^{2-f(|\Sigma|)}\right)$ for some decreasing function $f$ that yields strongly subquadratic-time algorithms for any constant alphabet size $|\Sigma|$?

Finally, we can compute a $(1+\varepsilon)$-approximation of LCIS in $\mathcal{O}\left(n^{3/2}\varepsilon^{-1/2}\text{polylog}(n)\right)$ time by an easy observation (see the appendix in the full version). Can we improve upon this running time or give a matching conditional lower bound? Note that a positive resolution seems difficult by the reduction in Observation 1: Any $n^\alpha$, $\alpha > 0$, improvement over this running time would yield a strongly subcubic $(1+\varepsilon)$-approximation for 3-LCS, which seems hard to achieve, given the difficulty to find strongly subquadratic $(1+\varepsilon)$-approximation algorithms for LCS.

## References


1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Quadratic-time hardness of LCS and other sequence similarity measures. In *Proc. 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*, pages 59–78, 2015.
2. Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends or: A polylog shaved is a lower bound made. In *Proc. 48th Annual ACM Symposium on Symposium on Theory of Computing (STOC'16)*, pages 375–388, 2016.
3. Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Proc. of 41st International Colloquium on Automata, Languages, and Programming (ICALP'14)*, pages 39–51, 2014.
4. Alfred V. Aho, Daniel S. Hirschberg, and Jeffrey D. Ullman. Bounds on the complexity of the longest common subsequence problem. *Journal of the ACM*, 23(1):1–12, 1976.
5. Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

**6** Hsing-Yen Ann, Chang-Biau Yang, and Chiou-Ting Tseng. Efficient polynomial-time algorithms for the constrained LCS problem with strings exclusion. *Journal of Combinatorial Optimization*, 28(4):800–813, 2014.

**7** Alberto Apostolico and Concettina Guerra. The longest common subsequence problem revisited. *Algorithmica*, 2(1):316–336, 1987.

**8** Abdullah N. Arslan and Ömer Egecioglu. Algorithms for the constrained longest common subsequence problems. *International Journal of Foundations of Computer Science*, 16(6):1099–1109, 2005.

**9** Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proc. 47th Annual ACM Symposium on Theory of Computing (STOC'15)*, pages 51–58, 2015.

**10** Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *Proc. 57th Annual Symposium on Foundations of Computer Science, (FOCS'16)*, pages 457–466, 2016.

**11** Arturs Backurs and Christos Tzamos. Improving viterbi is hard: Better runtimes imply faster clique algorithms. In *Proc. 34th International Conference on Machine Learning (ICML'17)*, 2017. To appear.

**12** Gary Benson, Avivit Levy, S. Maimoni, D. Noifeld, and B. Riva Shalom. Lcsk: A refined similarity measure. *Theoretical Computer Science*, 638:11–26, 2016.

**13** Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence algorithms. In *Proc. 7th International Symposium on String Processing and Information Retrieval (SPIRE'00)*, pages 39–48, 2000.

**14** Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *Proc. 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS'14)*, pages 661–670, 2014.

**15** Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proc. 56th Annual IEEE Symposium on Foundations of Compu ter Science (FOCS'15)*, pages 79–97, 2015.

**16** Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, 2018. To appear.

**17** Wun-Tat Chan, Yong Zhang, Stanley P. Y. Fung, Deshi Ye, and Hong Zhu. Efficient algorithms for finding a longest common increasing subsequence. *Journal of Combinatorial Optimization*, 13(3):277–288, 2007.

**18** Yi-Ching Chen and Kun-Mao Chao. On the generalized constrained longest common subsequence problems. *Journal of Combinatorial Optimization*, 21(3):383–392, 2011.

**19** Francis Y. L. Chin, Alfredo De Santis, Anna Lisa Ferrara, N. L. Ho, and S. K. Kim. A simple algorithm for the constrained sequence problems. *Inf. Process. Lett.*, 90(4):175–179, 2004. doi:10.1016/j.ipl.2004.02.008.

**20** Vaclav Chvatal, David A. Klarner, and Donald E. Knuth. Selected combinatorial research problems. Technical Report CS-TR-72-292, Stanford University, Department of Computer Science, 6 1972.

**21** Maxime Crochemore and Ely Porat. Fast computation of a longest increasing subsequence and application. *Information & Computation*, 208(9):1054–1059, 2010.

**22** Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. On problems equivalent to (min,+)-convolution. In *Proc. 44th International Colloquium on Automata, Languages, and Programming (ICALP'17)*, pages 22:1–22:15, 2017.

**23** Lech Duraj. A linear algorithm for 3-letter longest common weakly increasing subsequence. *Information Processing Letters*, 113(3):94–99, 2013.

**24** Michael L. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975.

**25** Zvi Gotthilf, Danny Hermelin, Gad M. Landau, and Moshe Lewenstein. Restricted LCS. In *Proc. 17th International Symposium on String Processing and Information Retrieval (SPIRE'10)*, pages 250–257, 2010.

**26** Daniel S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM*, 24(4):664–675, 1977.

**27** J. W. Hunt and M. D. McIlroy. An algorithm for differential file comparison. Computing Science Technical Report 41, Bell Laboratories, 1975.

**28** James W. Hunt and Thomas G. Szymanski. A fast algorithm for computing longest subsequences. *Communications of the ACM*, 20(5):350–353, 1977.

**29** Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

**30** Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

**31** Guy Jacobson and Kiem-Phong Vo. Heaviest increasing/common subsequence problems. In *Combinatorial Pattern Matching, Third Annual Symposium, CPM 92, Tucson, Arizona, USA, April 29 - May 1, 1992, Proceedings*, pages 52–66, 1992.

**32** Tao Jiang, Guohui Lin, Bin Ma, and Kaizhong Zhang. The longest common subsequence problem for arc-annotated sequences. *Journal of Discrete Algorithms*, 2(2):257–270, 2004.

**33** Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the Fine-grained Complexity of One-Dimensional Dynamic Programming. In *Proc. 44th International Colloquium on Automata, Languages, and Programming (ICALP'17)*, pages 21:1–21:15, 2017.

**34** Martin Kutz, Gerth Stølting Brodal, Kanela Kaligosi, and Irit Katriel. Faster algorithms for computing longest common increasing subsequences. *Journal of Discrete Algorithms*, 9(4):314–325, 2011.

**35** William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

**36** Howard L. Morgan. Spelling correction in systems programs. *Communications of the ACM*, 13(2):90–94, 1970.

**37** Eugene W. Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.

**38** Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

**39** Adam Polak. Why is it hard to beat $O(n^2)$ for longest common weakly increasing subsequence? *CoRR*, abs/1703.01143, 2017.

**40** Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proc. 45th Annual ACM Symposium on Symposium on Theory of Computing (STOC'13)*, pages 515–524, 2013.

**41** Yin-Te Tsai. The constrained longest common subsequence problem. *Information Processing Letters*, 88(4):173–176, 2003.

**42** Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

**43** Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.

**44** Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *Proc. 10th International Symposium on Parameterized and Exact Computation (IPEC'15)*, pages 17–29, 2015.

**45** I-Hsuan Yang, Chien-Pin Huang, and Kun-Mao Chao. A fast algorithm for computing a longest common increasing subsequence. *Information Processing Letters*, 93(5):249–253, 2005.

**46** Daxin Zhu, Lei Wang, Tinran Wang, and Xiaodong Wang. A simple linear space algorithm for computing a longest common increasing subsequence. *CoRR*, abs/1608.07002, 2016.

# $K$-Best Solutions of MSO Problems on Tree-Decomposable Graphs*

**David Eppstein**[†1] **and Denis Kurz**[‡2]

1 **Computer Science Department, UC Irvine, USA**
eppstein@uci.edu
2 **Department of Computer Science, TU Dortmund, Germany**
denis.kurz@tu-dortmund.de

**Abstract**

We show that, for any graph optimization problem in which the feasible solutions can be expressed by a formula in monadic second-order logic describing sets of vertices or edges and in which the goal is to minimize the sum of the weights in the selected sets, we can find the $k$ best solution values for $n$-vertex graphs of bounded treewidth in time $\mathcal{O}(n+k\log n)$. In particular, this applies to finding the $k$ shortest simple paths between given vertices in directed graphs of bounded treewidth, giving an exponential speedup in the per-path cost over previous algorithms.

**1998 ACM Subject Classification** G.2.2 Graph Theory

**Keywords and phrases** graph algorithm, $k$-best, monadic second-order logic, treewidth

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.16

## 1 Introduction

Finding multiple alternative routes between two vertices in a network, formalized as the *$k$ shortest paths problem*, has many applications including biological sequence alignment, metabolic pathway reconstruction, hypothesis generation in natural language processing, computer network routing, and vehicle routing [15]. It can be solved in constant time per path, after near-linear preprocessing time [14]. However, for graphs with cycles, its paths may have repeated vertices, which are often undesirable. A variant of the problem, the *$k$ shortest simple paths problem*, disallows these repetitions, but for the past 45 years there have been no asymptotic improvements to an algorithm of Yen, which takes quadratic time per path [23]. Indeed, Vassilevska Williams and Williams show that, at least for $k = 2$, no substantial improvement to this algorithm is likely [22]. Even for undirected graphs, known algorithms for $k$ shortest simple paths take at least linear time per path [21], far from the constant time per path for non-simple paths.

This situation suggests studying parameterized classes of graphs for which faster $k$-best optimization algorithms are possible. In this paper we provide a first result of this type, showing that the lengths of the $k$ shortest simple paths can be found in logarithmic time per path (exponentially faster than the polynomial per-path time of previous algorithms) for graphs of bounded treewidth. Our results are based on general algorithmic metatheorems

* A full version of the paper is available at https://arxiv.org/abs/1703.02784.

† The research of David Eppstein was supported in part by NSF grants CCF-1228639, CCF-1618301, and CCF-1616248.

‡ This research was performed in part during a visit of Denis Kurz to UC Irvine, supported by DFG GRK 1855 (DOTS).





and in particular *Courcelle's theorem* according to which decision and optimization problems expressible in the monadic second-order logic of graphs (MSO) can be solved in linear time on graphs of bounded treewidth. MSO is a form of logic in which the variables of a formula represent vertices, edges, sets of vertices, and sets of edges of a graph, one can test set membership and vertex–edge incidence, and variables can be existentially or universally quantified. For instance the property that an edge set $P$ represents a simple path from $s$ to $t$ can be expressed in MSO as a formula

$$\begin{aligned}
&(\forall v,e,f,g)\Big[\big(I(v,e)\wedge e\in P\wedge I(v,f)\wedge f\in P\wedge I(v,g)\wedge g\in P\big)\Rightarrow\big(e=f\vee e=g\vee f=g\big)\Big]\\
&\quad\wedge(\exists e)\Big[I(s,e)\wedge e\in P\wedge(\forall f)\Big[\big(I(s,f)\wedge f\in P\big)\Rightarrow e=f\big)\Big]\Big]\\
&\quad\wedge(\exists e)\Big[I(t,e)\wedge e\in P\wedge(\forall f)\Big[\big(I(t,f)\wedge f\in P\big)\Rightarrow e=f\big)\Big]\Big]\\
&\quad\wedge(\forall S)\Big[\neg(\exists v,e)\big[v\in S\wedge e\in P\wedge I(v,e)\big]\vee\neg(\exists v,e)\big[\neg(v\in S)\wedge e\in P\wedge I(v,e)\big]\vee\\
&\qquad\qquad(\exists v,w,e)\big[v\in S\wedge\neg(w\in S)\wedge I(v,e)\wedge I(w,e)\big]\Big]
\end{aligned}$$

(where $s$, $t$, $v$ and $w$ are vertex variables, $e$, $f$, and $g$ are edge variables, $S$ is a vertex-set variable, and $I$ is the vertex–edge incidence predicate). This formula expresses the constraints that each vertex is incident to at most two edges of $P$, $s$ and $t$ are each incident to exactly one edge of $P$, and every partition of the vertices that is not crossed by $P$ has $P$ only on one of its two sides.

Courcelle's theorem translates MSO formulas to tree automata, allowing graphs of bounded treewidth that satisfy the formula to be recognized in linear time by a bottom-up dynamic programming algorithm that executes the tree automaton on a tree-decomposition of the graph. Extensions of this method also solve optimization problems for MSO predicates (formulas with one unbound set-variable) that seek the minimum weight vertex or edge set obeying the predicate [3, 10]. For instance, it can find shortest simple paths on bounded-treewidth graphs with negative edges and negative cycles, an NP-hard problem on arbitrary graphs.

In this paper, we show that, for any MSO predicate, the weights of the $k$ minimum-weight sets satisfying the predicate can be found on graphs of bounded treewidth in logarithmic time per set. In particular, using the formula given above, we can find the $k$ shortest simple paths in logarithmic time per path. Other previously-studied graph optimization problems to which our method applies (with an exponential per-solution speedup) include finding the $k$ smallest spanning trees [16], the $k$ best matchings [8], and (with a doubly exponential speedup) the $k$ best solutions to the traveling salesperson problem [11]. Although the example formula above describes simple paths in undirected graphs, our method applies as well to directed graphs whose underlying undirected graph has bounded treewidth.

To prove this we use a special tree-decomposition with bounded width, logarithmic depth, and bounded degree. We translate the dynamic program for finding the minimum weight set satisfying an MSO predicate into a *fully persistent dynamic graph algorithm* for the same optimization problem, one that can report the minimum weight solution after modifying the given graph by changing the weights of some of its edges or vertices. We apply this method to find the *second-best* (rather than best) solution, and to detect a feature of the graph (a vertex or edge) at which the best and second-best solution differ. By branching on this feature we can recursively decompose the original problem into a hierarchy of subproblems whose second-best solutions (together with the global best solution) include all of the $k$ best

solutions to the input problem. To find the $k$ best solutions, we perform a best-first search of this hierarchy.

For space reasons, we defer some proofs to the full version, online at arXiv:1703.02784.

## 2 Preliminaries

In this section, we establish most of our notation and definitions, and review algorithmic components from previous research that we will use to establish our results.

We denote by $[i]$ the set $\{1, \ldots, i\}$ for $i \in \mathbb{N}$. The cardinality of a set $M$ is denoted by $|M|$, and its power set by $2^M$. For a function $f : M \to N$, we write $f(M') = \{f(a) \mid a \in M'\}$ for $M' \subseteq M$. For an $n$-element sequence $S$ and $i \in [n]$, $S_i$ denotes the $i$-th element of $S$.

Our algorithms assume a RAM model of computation in which addition and comparison operations on input weights or sums of weights can be performed in constant time per operation.

### 2.1 Binary heap of subproblems

We adopt the following technique for finding the $k$ best solutions to a combinatorial optimization problem [20, 15], which was first used by Gabow for the $k$ smallest spanning trees [19]. We assume that the problem's solutions can be represented as sets of edges or vertices in a weighted graph, and that the goal is to minimize total weight. We form *subproblems* by forcing certain edges or vertices to be included in the solution and preventing other edges or vertices from being included; these constraints can be simulated without changing the graph by changing some weights to large positive and negative numbers. We say that a subproblem is *feasible* if it has a solution consistent with its constraints. If $S$ is any subproblem, then we may consider two solutions to $S$, its *best solution* (the one with minimum weight, subject to the constraints) and its *second-best solution* (the one that differs from the best solution and otherwise has the minimum possible weight), with ties broken in any consistent way. We say that an edge or vertex is a *pivot feature* if it is present in the best solution but absent in the second-best solution, or vice versa. If a subproblem has only one solution, we say that it is *uniquely solvable*.

We then form a binary tree of feasible subproblems, as follows. The root of the tree is the subproblem with no constraints (the one whose solutions are all solutions to the given problem on the whole graph $G$). Then, for each subproblem $S$ in the tree that is not uniquely solvable, the two children of $S$ are determined by choosing (arbitrarily) a pivot feature of $S$, constraining that pivot feature to be included in the solutions for one child, and constraining the same pivot feature to be excluded from the solutions for the other child. These two children of $S$ have solution sets that partition the solutions of $S$ into two nonempty subsets, one containing the best solution and the other containing the second-best solution. A uniquely solvable subproblem in this tree of subproblems forms a leaf, with no children.

Each solution of the input problem appears as second-best in exactly one subproblem in this tree, except for the global best solution which is never second-best anywhere. The tree is ordered as a binary min-heap: each subproblem's second-best solution is better than the second-best solutions of its children. Therefore, the $k$-best solutions of the input problem can be listed by finding the best solution and then using a best-first search in the tree of subproblems to find the $k - 1$ subproblems whose second-best solutions have the smallest values. This search creates an active set consisting of the tree root, and then (for $k - 1$ iterations) uses a priority queue to find the smallest second-best solution value in

the active set, outputs that solution, and replaces that node in the active set by its two children. Alternatively, the added time from the priority queue can be avoided by using a heap-selection algorithm of Frederickson [18] which runs in $O(k)$ time, plus the time for $\mathcal{O}(k)$ evaluations of subproblems.

Using this method, the main remaining task is to find the second-best solution values and pivot features of each subproblem in this tree of subproblems, as efficiently as possible.

## 2.2 Path-copying persistence

We will develop a data structure that allows us to add a new constraint to a subproblem, forming one of its child subproblems, and efficiently compute the new second-best solution value of the new child subproblem. However, without additional techniques such a data structure would allow us to follow only a single branch of the tree of subproblems. We use ideas from *persistent data structures*, following Sarnak et al. [12], to extend these data structures to ones that let us explore multiple branches of the tree of subproblems concurrently.

An *ephemeral data structure* is one that has only a single version, which is changed by certain *update* operations and accessed but not changed by additional *query* operations. In the corresponding *fully persistent data structure*, each operation takes an additional argument, the *version* of the data structure, and operates on that version. Persistent queries return the result of the query on that version, and do not change it. Persistent updates create and return a new version of the data structure, in which the given change has been made to the version given as an argument. The previous version is left intact, allowing future updates and queries to it.

*Path copying* is a technique introduced by Sarnak et al. for making any tree-based ephemeral data structure (such as a binary heap or binary search tree) fully persistent. The underlying ephemeral data structure must form a tree of nodes, with each node pointing to its children but without pointers to parents or other non-child nodes. Every operation in the ephemeral data structure should be performed by starting at the tree root, following child pointers to find additional nodes reachable by paths from the root, and then (in case of a query) collecting information from those nodes or (for updates) changing or replacing some of the reached nodes.

To make such a data structure fully persistent, we represent each version of the data structure by the root of its tree. Persistent queries follow the same algorithm as ephemeral queries, starting from the root node representing the desired version. When an ephemeral update would change or replace some nodes, the persistent structure creates new nodes for all of these changed or replaced nodes and all of their ancestors, without changing to the existing nodes. In this way, the space and time of each persistent update are proportional to the time for an ephemeral update.

Path-copying persistence was used in the $k$-shortest paths algorithm of Eppstein [14], to construct certain persistent heap structures that represent sets of *detours*. Here, we apply the same technique in a different way, to make persistent an ephemeral data structure for second-best solutions. We will associate a version of the second-best solution data structure with each subproblem in the binary tree of subproblems described in the previous section. Then, when we expand a subproblem (finding its pivot feature and using that feature to define two new child subproblems) the data structure versions of the two child subproblems can be found by applying two different persistent updates to the version of their parent.

## 2.3 Shallow tree decompositions

The data structure to which we will apply the path-copying persistence technique will be based on a tree decomposition of the given graph. However, in order to make the path-copying efficient, we need to use a special kind of tree decomposition, one with low depth.

A *tree decomposition* of a graph $G = (V, E)$ is a pair $(T = (U, F), bag)$, where $T$ is a tree, $bag : U \to 2^V$ maps tree nodes to subsets of $V$ ("bags"), each vertex belongs to a nonempty collection of bags that induce a connected subtree of $T$, and for each edge at least one bag contains both edge endpoints. The *width* of a tree decomposition is one less than the size $\max\{|bag(u)| \mid u \in U\}$ of its largest bag. The *treewidth* of $G$ is the smallest $w$ such that $G$ has a tree decomposition of width $w$. For any fixed $w$, one can recognize the graphs of treewidth at most $w$ and compute a tree decomposition of optimal width for these graphs, in linear time [6].

We define the *depth* of a tree decomposition to be the longest distance from a leaf to a root node chosen to minimize this distance. A *shallow tree decomposition* of a graph $G$ of bounded treewidth $w$ is a tree decomposition with width $\mathcal{O}(w)$ and depth $\mathcal{O}(\log |G|)$ whose tree is binary. Shallow tree decompositions always exist [5], and can be constructed by a parallel algorithm whose sequential version takes linear time [7].

## 2.4 Hypergraph algebra

We adopt much of the following notation from Courcelle and Mosbah [10].

Let $A$ be a ranked alphabet consisting of *edge labels*, and let $\tau : A \to \mathbb{N}$ map labels to their *orders*. A *hypergraph* $G = (V, E, lab, vert, src)$ *of order* $r$ consists of a set of vertices $V$ and a set of hyperedges $E$, an edge labeling function $lab : E \to A$, a function $vert : E \to V^*$ that maps edges to node sequences, and a sequence $src$ of $r$ source nodes. The *order* of a hyperedge $e \in E$ is the length $|vert(e)|$ of its vertex sequence, and must match the order of its label: $\tau(lab(e)) = |vert(e)|$. A graph of order $r$ is a hypergraph of order $r$ with $\tau(lab(E)) = \{2\}$, so every hyperedge has order 2. Hyperedges of a graph may be called *edges*.

We define, for an edge label alphabet $A$, a hypergraph algebra with a possibly infinite set of hypergraph operators and the following finite set of constants. The constant $\mathbf{0}$ denotes the empty hypergraph of order 0. The constant $\mathbf{1}$ denotes the hypergraph of order 1 with a single source vertex and no hyperedges. For each $a \in A$, the constant $\mathbf{a}$ denotes the hypergraph of order $\tau(a)$ with node set $\{v_1, \ldots, v_{\tau(a)}\}$, a single hyperedge $e$ with $lab(e) = a$ and $vert(e) = (v_i)_{i \in [\tau(a)]}$, and $src = vert(e)$.

Let $G$ be a hypergraph of order $r$ and let $G'$ be a hypergraph of order $r'$. The hypergraph algebra has the following operators. The $(r + r')$-order hypergraph $G \oplus_{r,r'} G'$ consists of the disjoint union of the vertex and edge sets of $G$ and $G'$, and the concatenation of their source sequences. For each $i, j \in [r]$, $\theta_{i,j,r}(G)$ is the hypergraph of order $r$ obtained from $G$ by replacing every occurence of $src_j$ with $src_i$ in the source sequence of $G$ and in every vertex sequence of a hyperedge of $G$. This is equivalent to fusing $src_j$ into $src_i$. For a mapping $\alpha : [p] \to [r]$, $\sigma_\alpha(G)$ is the hypergraph of order $p$ obtained from $G$ by replacing its $r$-element source sequence $src$ with the $p$-element sequence $src'$, with $src'_i = src_{\alpha(i)}$. This infinite family of operators can generate any hypergraph. For each family of hypergraphs $L$ that only contains hypergraphs of bounded treewidth and bounded order, a finite subset of these operators generates a superset of $L$ [2, 4, 9]. We denote by $\mathcal{G}_w$ a finite hypergraph algebra as above that generates all $r$-order hypergraphs over a fixed label set of treewidth at most $w$ and $r < R$ for some fixed but arbitrary $R$.

Let $A$ be an alphabet of edge labels, and let $G = (V, E, lab, vert, src)$ be a hypergraph over $A$. A formula in *counting monadic second-order logic* (*CMS formula*) is a formula in monadic second-order logic, extended by predicates $\mathbf{Card}_{m,p}(X)$ for $m, p \in \mathbb{N}$, with $X \models \mathbf{Card}_{m,p}(X)$ iff $|X| \equiv m \mod p$, and by incidence predicates $edg_a(e, v_1, \ldots, v_{\tau(a)})$ for $a \in A$, with $(G, e, v_1, \ldots, v_{\tau(a)}) \models edg_a(e, v_1, \ldots, v_{\tau(a)})$ iff $lab(e) = a$ and $vert(e) = (v_1, \ldots, v_{\tau(a)})$. The finite set $\Phi^{h,q}_{A,R}(\mathcal{W})$ consists of exactly one representative of every class of equivalent CMS formulas for hypergraphs of order at most $R$ over edge labels $A$ on variables $\mathcal{W}$ whose depth of nested quantification is at most $h$ and $p < q$ for all subformulas of the form $\mathbf{Card}_{m,p}(X)$. Since $h$, $q$, $A$ and $\mathcal{W}$ are fixed in most contexts, we use the short form $\Phi_R = \Phi^{h,q}_{A,R}(\mathcal{W})$. We require variable alphabets $\mathcal{W}$ that include constants for all the source nodes of the hypergraphs, i.e., $\{\mathbf{s}_i \mid i \in [R]\} \subset \mathcal{W}$ and $(G, v) \models (v = \mathbf{s}_i)$ iff $v = src_i$. Every other variable $X$ in $\mathcal{W}$ is assumed to be either a vertex set variable, denoted as $type(X) = V$, or a hyperedge set variable, denoted as $type(X) = E$. This can be enforced by only considering formulas that include subformulas $\forall x : x \in X \Rightarrow x \in V$ if X is supposed to model a set of vertices, and $\forall x : x \in X \Rightarrow x \in E$ otherwise. Variables representing single elements can be emulated by a subformula $\exists x : \forall y : x \in X \wedge (y \in X \Rightarrow x = y)$.

Our algorithms work on parse trees of hypergraphs with respect to some fixed hypergraph algebra $\mathcal{G}_w$. A *parse tree* $T = (U, F)$ is a directed rooted tree, with edges being directed away from the root $r$. Leaves of $T$ are associated with a constant of $\mathcal{G}_w$; inner nodes are associated with an operator of $\mathcal{G}_w$. The hypergraph represented by $T$ is the hypergraph constant associated with $r$ if $r$ is a leaf. Otherwise, $T$ represents the hypergraph obtained by applying the operator associated with $r$ to the hypergraphs represented by the parse subtrees rooted in the children of $r$. For $u \in U$, the hypergraph represented by the parse subtree of $T$ rooted in $u$ is denoted by $G(u)$. Sometimes it is convenient to have exactly two child nodes for each inner node of the parse tree. We can think of $\sigma_\alpha$ operators as having a second operand that is always the empty hypergraph. Likewise, $\theta_{i,j,r}$ operators can be viewed as having a second operand that is always the one-vertex hypergraph. These modified operators can even be derived from the original ones. Instead of $\sigma_\alpha$, we can use $\sigma'_\alpha = \sigma_\alpha \circ \oplus_{0,r}(\mathbf{0})$. The composition $\theta'_{i,j,r} = \theta_{i,j,r} \circ \sigma_\alpha \circ \theta_{i,r+1,r+1} \circ \oplus_{1,r}(\mathbf{1})$ with $\sigma : [r] \to [r+1]$, $\sigma(i) = i$, can replace $\theta_{i,j,r}$. All operators of this derived algebra are binary. We call a respective parse tree a *full parse tree*. The *proper* child of a $\theta$ or $\sigma$ node is the one that does not represent the one-vertex or empty hypergraph, respectively.

Let $G$ be a hypergraph, and $T = (U, F)$ a parse tree of $G$. We consider combinatorial problems on $G$ that can be characterized by a CMS formula $\varphi \in \Phi_R$. Let $n$ be the number of free variables of $\varphi$, and let $X = (X_i)_{i \in [n]}$ be the free variables themselves. For example, we need $n = 1$ edge set to describe a (simple) path, or $n = c - 1$ node sets to describe a $c$-coloring of a graph. An *assignment* maps every $X_i$ to a subset of $type(X_i)$. A *satisfying assignment* is an assignment $f$ such that $(f(X), G) \models \varphi$. A *solution* is the sequence $(f(X_i))_{i \in [n]}$ for some assignment $f$. Two solutions $S$, $S'$, are considered distinct if there is a $j \in [n]$ with $S_j \neq S'_j$. For $S_1 \neq S_2$, $(S_1, S_2)$ is also distinct from $(S_2, S_1)$ regardless of $\varphi$, even if $\varphi$ is a formula on two free variables, and symmetric on these free variables. For a parse tree node $u \in U$, $S(u)$ denotes the *subsolution of $S$ at $u$*, which is obtained from $S$ be removing every graph feature from every set $S_i$ that is not present in $G(u)$. A solution is *feasible* if the corresponding assignment is satisfying. We denote by $\boldsymbol{sat}(G, \varphi)$ the set of feasible solutions, i.e., $Y \in \boldsymbol{sat}(G, \varphi) \Leftrightarrow (Y, G) \models \varphi$.

Sets of solutions, and $\boldsymbol{sat}(G, \varphi)$ in particular, are sets of $n$-tuples of sets. We write $M \sqcup N$ for the disjoint union of sets $M$ and $N$ (undefined if $M \cap N \neq \emptyset$). Let $X$, $Y$ be solutions, and $A$, $B$ sets of solutions. We say that $X$ and $Y$ *interfere* if there are $i, j \in [n]$ such that

$X_i \cap Y_j \neq \emptyset$. By extension, $A$ and $B$ interfere if some $X \in A$ interferes with some $Y \in B$. If $A$ and $B$ do not interfere, $A \uplus B$ denotes the set of all combinations of each $X \in A$ and $Y \in B$, i.e.,

$$A \uplus B = \{(X_i \sqcup Y_i)_{i\in[n]} \mid X \in A, Y \in B\}.$$

Note that the above operators are not defined for every combination of operands. A *semi-homomorphism* from $\langle \mathcal{S}, \uplus, \sqcup, \emptyset, \emptyset\rangle$ to some evaluation structure $\langle \mathcal{R}, \oplus, \otimes, \mathbf{0}, \mathbf{1}\rangle$ is a mapping $f : \mathcal{S} \to \mathcal{R}$ that acts like a homomorphism where applicable, i.e., $f(\emptyset) = \mathbf{0}$, $f(\emptyset) = \mathbf{1}$, $f(A \uplus B) = f(A) \oplus f(B)$ if $A$ and $B$ do not interfere, and $f(A \sqcup B) = f(A) \otimes f(B)$ if $A \cap B = \emptyset$.

Values of solution sets are expressed in terms of *evaluation structures*. An evaluation structure is an algebra $\langle \mathcal{R}, \oplus, \otimes, \mathbf{0}, \mathbf{1}\rangle$ such that $\langle \mathcal{R}, \oplus, \mathbf{0}\rangle$ and $\langle \mathcal{R}, \otimes, \mathbf{1}\rangle$ are monoids. An *evaluation* $v$ is a function that maps hypergraphs generated by $\mathcal{G}_w$ to an (ordered) evaluation structure $\mathcal{R}$. An evaluation $v$ is an *MS-evaluation* if there exists a semi-homomorphism $h$ and a CMS formula $\varphi \in \Phi_R$ such that $v(G) = h(\boldsymbol{sat}(G, \varphi))$ for every hypergraph $G$.

A *linear CMS minimization problem* $P$ consists of a formula $\varphi = \varphi(P) \in \Phi_R$ with $n$ free variables $X$ that characterizes feasible solutions, and an *instance* of the problem consists of an $r$-order hypergraph $G$ and a sequence $c$ of $n$ cost functions with $c_i : \mathit{type}(X_i) \to \mathbb{R}$. The value $c(Y)$ of a solution $Y$ for $P$ is defined as $\sum_{i\in[n]} \sum_{y\in Y_i} c_i(y)$. An optimal solution of $P$ is a feasible solution $Y^*$ such that for each feasible solution $Y'$ of $P$, we have $c(Y^*) \leq c(Y')$. Our task is to find the value of an optimal feasible solution of $P$, or equivalently, to compute the value of $v(G) = \min\{c(Y) \mid Y \in \boldsymbol{sat}(G, \varphi)\}$. This evaluation can be expressed as $v(G) = h(\boldsymbol{sat}(G, \varphi))$ with $h(A) = \min\{c(Y) \mid Y \in A\}$, and is thus an MS-evaluation. Note that $h$ is a semi-homomorphism, and the corresponding evaluation structure is $\langle \mathbb{R} \cup \{\infty\}, +, \min, 0, \infty\rangle$. The class of all linear CMS minimization problems is called LinCMS. Let $m = |\boldsymbol{sat}(G, \varphi)|$, and $\boldsymbol{sat}(G, \varphi) = \{Y^1, \ldots, Y^m\}$. Let $\Pi = \Pi(G, \varphi) \subseteq S_n([m])$ be the set of permutations such that $(c(Y^{\pi(i)}))_{i\in[m]}$ is nondecreasing for $\pi \in \Pi$. For a problem $P \in \text{LinCMS}$, $k$-*val*$(P)$ is again a problem that gets the same input as $P$ itself, plus some $k \in \mathbb{N}$. It asks for the sequence $(c(Y^{\pi(i)}))_{i\in[k']}$ for some $\pi \in \Pi$, where $k' = \min\{k, m\}$. The problem $k$-*sol*$(P)$ gets the same input as $k$-val$(P)$, but asks for $(Y^{\pi(i)})_{i\in[k']}$ for any $\pi \in \Pi$. Depending on $\pi$, different outputs for $k$-sol$(P)$ are possible, but there is only one valid output for $k$-val$(P)$. Also note that 1-val$(P)$ is equivalent to $P$ itself. For simplicity, we assume $k \leq m$ for the remainder of this article.

## 3 The second-best solution

In the following section, we describe how to solve 2-sol$(P)$ in linear time for each $P$ in LinCMS. In a later section we generalize these results to an arbitrary (constant) number of solutions, and to LOGSPACE and PRAM models of computation.

A hypergraph $G$ generated by $\mathcal{G}_w$, and a formula $\varphi \in \Phi_R$, $\boldsymbol{sat}(G, \varphi)$ can be evaluated by a tree automaton on a parse tree of $G$ [10, 17]. Our notion of full parse trees enables us to unify Lemmas 2.4 to 2.6 from Courcelle and Mosbah [10] as follows.

▶ **Lemma 1.** *Let $\varphi \in \Phi_R$, and let $T$ be a full parse tree rooted in $r$ that represents a hypergraph $G$. If $r$ is not a leaf, it has two child nodes $u_1$, $u_2$ representing $G_1 = G(u_1)$, $G_2 = G(u_2)$, respectively, and there exist $l \in \mathbb{N}$ and $\psi_1^k, \psi_2^k \in \Phi_R$ for each $k \in [l]$ such that*

$$\boldsymbol{sat}(G, \varphi) = \bigsqcup \left\{ \boldsymbol{sat}(G_1, \psi_1^k) \uplus \boldsymbol{sat}(G_2, \psi_2^k) \mid k \in [l] \right\}, \tag{1}$$

*where $l$ only depends on the hypergraph operator associated with $r$.*

In the situation of Lemma 1, we call $\psi_i^k$ a *child formula* of $\varphi$ with respect to the corresponding operator. For $k \in [l]$, we call $(\psi_1^k, \psi_2^k)$ a *fitting pair of child formulas*. A solution $S \in \boldsymbol{sat}(G, \varphi)$ has a unique fitting pair $(\psi_1, \psi_2)$ of child formulas with $S \in (\boldsymbol{sat}(G_1, \psi_1) \sqcup \boldsymbol{sat}(G_2, \psi_2))$, which follows from the fact that all unions in Equation (1) are disjoint. All child formulas $\psi$ with respect to a $\theta_{i,j,m}$ node $u$ can be chosen such that $src_i \notin S_k$ for any $S \in \boldsymbol{sat}(G(u'), \psi)$, where $u'$ is the proper child of $u$. Further, Courcelle and Mosbah demonstrated that every MS-evaluation $v$ can be computed by a similar tree automaton. The running time required to compute $v$ on the entire hypergraph $G$ is $\mathcal{O}(|G| \cdot \mu)$, where $\mu$ is the time required to compute $f(A) \oplus f(B)$ and $f(A) \otimes f(B)$ for valid combinations $A$, $B$. In the uniform cost model, the operators of the evaluation structure $\langle \mathbb{R} \cup \{\infty\}, +, \min, 0, \infty \rangle$, addition and selecting the smaller of two real numbers, require $\mathcal{O}(1)$ time. Problems in LinCMS can therefore be solved in linear time. Applying the semi-homomorphism $h$ with $v(G) = h(\boldsymbol{sat}(G, \varphi))$ to Equation (1) yields

$$v(G) = h(\boldsymbol{sat}(G, \varphi)) = \min \left\{ h(\boldsymbol{sat}(G_1, \psi_1^k)) + h(\boldsymbol{sat}(G_2, \psi_2^k)) \mid k \in [l] \right\}. \tag{2}$$

A different linear-time approach for this special case had been proposed earlier by Arnborg, Lagergren and Seese [3]. The basic algorithm of Courcelle and Mosbah computes $h(\boldsymbol{sat}(G(u), \psi))$ for every parse tree node $u$ in a bottom-up manner, and every $\psi \in \Phi_R$. Conceptually, we perform a depth-first search on $T$, starting at its root $r$. Every time we finish a node $v$, we *evaluate* it, i.e., we compute the evaluation $h(\boldsymbol{sat}(G(v), \phi))$ for every formula $\psi \in \Phi_R$ based on the child formulas of $\psi$ by Equation (2). Since the number of formulas and the number of child formulas per formula are fixed, the overall running time is linear in the size of $T$.

Courcelle and Mosbah also propose an improved algorithm, called the *CM algorithm* in this article, that determines in a top-down preprocessing phase the set of formulas that are reachable from $\varphi$ at $r$ via the child formula relation. We call these the *relevant formulas* of a parse tree node $u$.

Let $x$ be a graph feature that is part of at least one solution for $\varphi$ at $r$, and let $v_1$, $v_2$ be the child nodes of $r$. If $r$ is an $\oplus$ or $\sigma$ node, the graph features of $G(v_1)$, $G(v_2)$ are disjoint, so exactly one of them contains $x$. If $r$ is a $\theta$ node that fuses $src_j$ into $src_i$, let $v_2$ be the the child node such that $G(v_2)$ only consists of $src_i$. As can be seen from the proof of Lemma 2.5 in [10], subformulas of $\varphi$ can be chosen such that $src_i$ is not part of any solution at $v_1$. Taken together, $x$ is part of some solutions at either $v_1$ or $v_2$, but not both. Applying the argument iteratively to this child node, there is exactly one leaf of the full parse tree where some solutions contain $x$. We denote this leaf by $u(x)$.

Let $u$ be a parse tree node, $\psi_1$, $\psi_2$ two formulas relevant for $u$. The CM algorithm only computes $h(G(u), \psi_1)$ and $h(G(u), \psi_2)$, which correspond to the values of optimal solutions on $G(u)$ for the problems characterized by $\psi_1$ and $\psi_2$, respectively. We do not have any information about the solutions themselves besides their values. In particular, we do not know for any hypergraph feature if it appears in one of those solutions. Since we do not require optimal solutions to be unique, we also do not know if they represent the same solution even in the case $h(\boldsymbol{sat}(G(u), \psi_1)) = h(\boldsymbol{sat}(G(u), \psi_2))$.

In the next section, we need to test if optimal solutions for $\psi_1$ and $\psi_2$ are the same by only looking at their evaluations. To do so, we establish for each parse tree node $u$ a mapping from the relevant formulas of $u$ to solution IDs in $[|\Phi_R|]$ , with the following discriminating property. Each relevant formula $\psi$ of $u$ is mapped to an optimal solution for $\psi$ on $G(u)$ such that two formulas are assigned the same solution if and only if they are assigned the same solution ID. Solution IDs can be computed along with solution values while keeping the linear time bound:

▶ **Lemma 2.** *Given solution IDs for all relevant formulas of all child nodes of a parse tree node $u$, solution IDs for $u$ can be computed in constant time.*

**Proof.** Deferred to the full version. ◀

The compression function can be stored with $u$, and a mapping from solution ID to solution at leaf nodes, requiring only constant space. Even if $\boldsymbol{sat}(G(u), \psi)$ contains multiple optimal solutions, it is now possible to refer to *the optimal solution*, which is the one defined recursively in terms of matching solution IDs. This particular optimal solution can be found by a simple depth-first search based algorithm in linear time, starting at $\varphi$ at the root of the parse tree. For each $(u, \psi)$, we find a fitting pair of child formulas $\rho_1$, $\rho_2$ for child nodes $v_1$, $v_2$ of $u$, respectively, such that the assigned solution IDs match, process $(v_1, \rho_1)$ and $(v_2, \rho_2)$ independently, and output the associated subsolution at leaf nodes.

To solve 2-val$(P)$, we adapt the evaluation structure. Instead of values in $\mathbb{R} \cup \{\infty\}$, we use pairs $(x, y) \in \mathcal{R} = (\mathbb{R} \cup \{\infty\})^2$, where $x$ and $y$ represents the values of an optimal and second-best solution. We define two new binary operators $+_2$ and $\min_2$ over $\mathcal{R}$, with $(x_1, y_1) +_2 (x_2, y_2) = (x_1 + x_2, \min(x_1 + y_2, x_1 + y_1))$ and $\min_2((x_1, y_1), (x_2, y_2)) = (a, b)$, where $a$, $b$ are the smallest and second-smallest element of the multiset $\{x_1, y_1, x_2, y_2\}$, respectively.

▶ **Lemma 3.** *Let $P$ be a LinCMS problem characterized by the formula $\varphi \in \Phi_R$ and cost functions $c$. Given a parse tree $T$ of a hypergraph $G$, the CM algorithm solves 2-val$(P)$ in linear time when used in conjunction with the evaluation structure $\langle \mathcal{R}, +_2, \mathit{min}_2, (0, 0), (\infty, \infty) \rangle$.*

**Proof.** For each leaf $u$ of $T$, we solve 2-val$(P)$ on $G(u)$ directly. Let $S^1 \in \boldsymbol{sat}(G, \varphi)$ be optimal. The mapping $v(G) = (c(S_1), \min(c(\boldsymbol{sat}(G, \varphi) \setminus \{S^1\})))$ can be written as $h(\boldsymbol{sat}(G, \varphi))$ with $h(A \uplus B) = h(A) \min_2 h(B)$ and $h(A \sqcup B) = h(A) +_2 h(B)$. The operators $\min_2$ and $+_2$ can be evaluated in constant time, resulting in linear total time using the CM algorithm. ◀

Solution IDs can be trivially generalized to the new evaluation structure, allowing us to refer to *the* optimal and second-best solution, and to reconstruct these two solutions in linear time.

▶ **Corollary 4.** *Let $P$ be a LinCMS problem, and let $w \in \mathbb{N}$ be fixed. Given a parse tree $T$ of a hypergraph $G$ with bounded treewidth, we can solve 2-sol$(P)$ on $G$ in time $\mathcal{O}(|G|)$.*

## 4 Dynamizing the second-best solution

In this section, we introduce the evaluation tree data structure that stores intermediate results of the algorithm from the previous section. The data structure allows for a trivial query for the values of the two best solutions in constant time, and a query for the solutions themselves in linear time. We demonstrate how to perform a query for a pivot feature, and how to update an evaluation tree to match the two subproblems with respect to this pivot feature as described in Section 2.1. Both operations start in the root of the given tree, enabling us to use the path persistence technique described in Section 2.2.

An *evaluation tree* is a tree with the same structure as the parse tree. We store with every node of the evaluation tree the result of all evaluations of the CM algorithm as described in Section 3, as well as all solution ID information. In addition, we also store the solution sets $\boldsymbol{sat}(G(u), \psi)$ themselves for each leaf node $u$. We first describe the query and update procedures for problems characterized by CMS formulas with exactly one free variable. For this purpose, we identify solutions $(S_1)$ with their first set $S_1$. For the sake of simplicity, we also identify nodes of the parse tree with their twins in the evaluation tree of the current subproblem. We assume that the hypergraph $G$ is represented as a full parse tree $T$.

Let $\varphi \in \Phi_R$ be a CMS formula that characterizes a subproblem $P$, and let $x$ be a pivot feature of $P$. If $x$ is a source vertex of $G$, the two subproblems of $P$ can again be characterized by formulas in $\Phi_R$, namely $\varphi \wedge (x \in X_1)$ and $\varphi \wedge (x \notin X_1)$, respectively. However, if $x$ is a hyperedge or a vertex that is not a source vertex of $G$, there are no such formulas in general. Therefore, not all subproblems can be characterized by a CMS formula, and we have to generalize the mapping $\boldsymbol{sat}$ to cover those subproblems. We define the set $\boldsymbol{sat}_P(G(u), \psi)$ to contain all solutions in $\boldsymbol{sat}(G(u), \psi)$ that satisfy the constraints imposed by the binary subproblem tree. Note that $h(\boldsymbol{sat}_P(G, \varphi))$ is the solution of 2-val($P$).

Let $S^1$ and $S^2$ be the optimal and second-best solution of $P$, respectively. To find a pivot feature $x$ for $P$, we maintain a current parse tree node $u_j$, two formulas $\psi_j^1$, $\psi_j^2$, and the invariant that the subsolutions $S^1(u_j)$, $S^2(u_j)$ of $S^1$, $S^2$ at $u_j$ are in $\boldsymbol{sat}_P(G(u_j), \psi_j^1)$, $\boldsymbol{sat}_P(G(u_j), \psi_j^2)$, respectively, and that $S^1(u_j) \neq S^2(u_j)$. In the $j$-th iteration, we choose $u_{j+1}$ to be a child node of $u_j$ in the parse tree. Initially, $u_1$ is the root of $T$, and $\psi_1^1 = \psi_1^2 = \varphi$. The invariant holds trivially because of $h(\boldsymbol{sat}_P(G(u_1), \varphi)) = h(\boldsymbol{sat}_P(G, \varphi))$, and because the optimal solution differs from the second-best one by definition.

If $u_j$ is a leaf, we construct the fixed-size subsolutions $S^1(u)$ and $S^2(u)$ explicitly. If the invariant holds, these solutions are distinct, and we can find a pivot feature in constant time. Otherwise, $u_j$ is an inner node with children $v_1$ and $v_2$. Let $\rho^1$ ($\rho^2$) be a fitting pair of child formulas for $\psi_j^1$ ($\psi_j^2$) for which solution IDs match. If the invariant holds for $j$, the subsolutions $S^1(u_j)$ and $S^2(u_j)$ differ, so their subsolutions at $v_1$ ($S^1(v_1)$ and $S^2(v_1)$) or those at $v_2$ have to differ as well. We choose $u_{j+1}$, $\psi_{j+1}^1$ and $\psi_{j+1}^2$ accordingly. The invariant for $j+1$ then holds by construction.

▶ **Lemma 5.** *Given an evaluation tree of depth $d$ for a subproblem $P$, the above algorithm finds a pivot feature for $P$ in time $\mathcal{O}(d)$.*

**Proof.** Deferred to the full version. ◀

Next, we describe the update process to transform the evaluation tree for $P$ into the evaluation tree for one of its two subproblems. The process is symmetric for both subproblems, so we only describe it for the subproblem $P'$ that requires solutions to contain the pivot feature.

First, we execute the query algorithm to find a pivot feature $x$, and push each node it visits to a stack. The pivot query algorithm can only terminate at leaf nodes, so the last node $u$ that is pushed to the stack is a leaf. Recall that for each leaf node $u$ and each relevant formula $\psi$, we store $\boldsymbol{sat}_P(G(u), \psi)$ explicitly. We enumerate this solution set and remove every solution that does not contain $x$ to obtain $\mathcal{S}(\psi) = \boldsymbol{sat}_{P'}(G(u), \psi)$. To re-evaluate $\psi$, we apply $h$ to $\mathcal{S}(\psi)$.

Any other node $u$ on the stack is an inner node of the full parse tree, and an ancestor of $u(x)$. Re-evaluation works the same as the original evaluation in the CM algorithm, by evaluating Equation (2) with operators $\min_2$, $+_2$ instead of min, + as in Lemma 3.

▶ **Lemma 6.** *Given an evaluation tree of depth $d$ for a subproblem $P$, the above algorithm updates the evaluation tree to match a subproblem $P'$ of $P$ in time $\mathcal{O}(d)$.*

**Proof.** Deferred to the full version. ◀

Let $S$ be the optimal or second-best solution of $P$, whichever remains feasible in $P'$. We do not require optimal solutions for subproblems to be unique, so we might find two optimal solutions for $P'$ that both differ from $S$. Alternatively, $S$ might turn up as the second-best solution for $P'$. Although these results would be valid outcomes of the update procedure, they

would break the $k$-best algorithm of Section 2.1: If we lose track of $S$ by choosing two other solutions as optimal and second-best, we cannot detect whether we find $S$ in a subproblem of $P'$ again, leading to (the value of) $S$ being output twice. This happens because solution IDs are not suitable to efficiently check two solutions with respect to different evaluation trees for equality. Therefore, we need to make sure that $S$ is the optimal solution of $P'$ as encoded by the solution IDs for $P$ and $P'$. Fortunately, the update procedure can trivially enforce this property.

Now consider a problem characterized by $N$ free variables, with $N > 1$. We may consider $N$ to be constant, as it only depends on the problem definition. Recall that solutions $S$ and $S'$ are distinct when there is a discriminating index $i$ with $S_i \neq S'_i$. The pivot query algorithm has to take this into account. For inner nodes, it relies only on solution IDs to choose a successor of the current parse tree node. Leaf nodes still have a fixed number of feasible solutions for a fixed-size hypergraph. The same arguments as above yield running time $\mathcal{O}(d)$. Similarly, we only have to adapt the processing of nodes for the update algorithm. For $N = 1$, we always imposed the new subproblem constraint on the first and only set of a solution, because the discriminating index was always the same. Now, we have to take into account the discriminating index as determined during the pivot query phase. Using the same arguments, updates can still be performed in time $\mathcal{O}(d)$. We now state our main result.

▶ **Theorem 7.** *Let $P$ be a LinCMS problem. Then computing the values of the $k$ best solutions for $P$ on a graph $G$ requires $\mathcal{O}(|G| + k \log |G|)$ time and space.*

**Proof.** In linear time, we compute a shallow tree decomposition, transform it into a parse tree $T$ of depth $\mathcal{O}(\log |G|)$, and apply the CM algorithm to $T$ by Lemma 3. With the results, we initialize a binary subproblem tree. We perform a best-first search to find the $k$ best subproblems with respect to their second-best solution as described in Section 2.1, which requires us to solve $\mathcal{O}(k)$ subproblems. Using the persistent tree technique from Section 2.2 in conjunction with the update operation above, we need $\mathcal{O}(d)$ additional time and space per subproblem. ◀

## 5 Fixed numbers of solutions

We generalize here our algorithm for the second best solution to any fixed number of solutions, and to low-space and parallel complexity classes. Similar results could be obtained by expressing the $k$-best solution problem (for a constant $k$) in MSO with $k$ set variables, all constrained to be solutions, to be distinct, and with minimum total weight; however, this would give a significantly worse dependence on $k$ than our solution and would require additional complication to recover the differences between the solutions.

For fixed $k \in \mathbb{N}$, operators $\min_k$ and $+_k$ can be defined analogously to $\min_2$ and $+_2$, respectively. Evaluating $\min_k(x, y)$ and $x +_k y$ for $k$-tuples $x, y \in \mathbb{R}^k$ still requires constant time. Hence, the CM algorithm with the evaluation structure $\langle (\mathbb{R} \cup \infty)^k, +_k, \min_k, (0, \ldots, 0), (\infty, \ldots, \infty) \rangle$ solves $k$-val$(P)$ for any problem $P$ in LinCMS. Let $T$ be a parse tree with depth $d$, and let $\leq_{\text{post}}$ be a postordering of $T$. To solve $k$-val$(P)$, we evaluate the nodes of $T$ according to $\leq_{\text{post}}$. As soon as a parse tree node has been evaluated, the evaluations of its children can be dropped. Thus, the number of stored evaluations never exceeds $d + 1$.

▶ **Theorem 8.** *Let $P$ be a LinCMS problem, and let $k, w \in \mathbb{N}$ be fixed. Given a graph $G$ with treewidth $w$, the problem $k$-val$(P)$ on $G$ can be solved using logarithmic memory space.*

**Proof.** A shallow tree decomposition $T$ can be computed with logarithmic memory space [13]. We use a depth-first search from an arbitrarily chosen root node to process the bags of $T$ in

postorder, processing each bag by replacing it with its fixed-size portion of the parse tree, which is then evaluated bottom-up. The evaluations of all parse tree nodes corresponding to a bag require constant space, and we store $\mathcal{O}(\log |G|)$ of them at a time. ◀

In the PRAM model (see e.g. [1]), the CM algorithm can be parallelized as follows.

▶ **Theorem 9.** *Let $P$ be a LinCMS problem, and let $k, w \in \mathbb{N}$ be fixed. In the EREW PRAM model, given a shallow tree decomposition $T$ of graph $G$ with treewidth $w$, the problem $k$-val$(P)$ on $G$ can be solved in time $\mathcal{O}(\log |G|)$ by $\mathcal{O}(|G|)$ processors.*

**Proof.** We allocate one processor $p(u)$ for each node $u$ of $T$, which computes the portion of the parse tree corresponding to its bag and evaluates all nodes of that portion. Processor $p(u)$ waits until all processors $p(v)$ of child nodes $v$ of $u$ have finished. The processor of the root node of $T$ therefore waits $\mathcal{O}(\log |G|)$ time. Only processor $p(u)$ writes solutions for node $u$, and only the parent $u'$ of $u$ reads them, according to the EREW model. ◀

Finally, using the algorithm of Bodlaender [5] on $\mathcal{O}(|G|^{3w+4})$ processors to compute a shallow tree decomposition, we obtain the following.

▶ **Corollary 10.** *Let $P$ be a LinCMS problem, and let $k, w \in \mathbb{N}$ be fixed. In the CRCW PRAM model, given a graph $G$ with treewidth $w$, the problem $k$-val$(P)$ on $G$ can be solved in time $\mathcal{O}(\log |G|)$ by $\mathcal{O}(|G|^{3w+4})$ processors.*

## References


1 Selim G. Akl. *Design and analysis of parallel algorithms.* Prentice Hall, 1989.

2 Stefan Arnborg, Bruno Courcelle, Andrzej Proskurowski, and Detlef Seese. An algebraic theory of graph reduction. *J. ACM*, 40(5):1134–1164, 1993. doi:10.1145/174147.169807.

3 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991. doi:10.1016/0196-6774(91)90006-K.

4 Michel Bauderon and Bruno Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20(2-3):83–127, 1987. doi:10.1007/BF01692060.

5 Hans L. Bodlaender. Nc-algorithms for graphs with small treewidth. In Jan van Leeuwen, editor, *Graph-Theoretic Concepts in Computer Science, 14th International Workshop, WG '88, Amsterdam, The Netherlands, June 15-17, 1988, Proceedings*, volume 344 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 1988. doi:10.1007/3-540-50728-0_32.

6 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.

7 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.

8 Chandra R. Chegireddy and Horst W. Hamacher. Algorithms for finding k-best perfect matchings. *Discrete Applied Mathematics*, 18(2):155–165, 1987. doi:10.1016/0166-218X(87)90017-5.

9 Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics*, pages 193–242. Elsevier, 1990.

10 Bruno Courcelle and Mohamed Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1&2):49–82, 1993. doi:10.1016/0304-3975(93)90064-Z.

**11** Edo S. Van der Poort, Marek Libura, Gerard Sierksma, and Jack A. A. van der Veen. Solving the $k$-best traveling salesman problem. *Computers & OR*, 26(4):409–425, 1999. doi:10.1016/S0305-0548(98)00070-7.

**12** James R. Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, 1989. doi:10.1016/0022-0000(89)90034-2.

**13** Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of bodlaender and courcelle. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 143–152. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.21.

**14** David Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, 1998. doi:10.1137/S0097539795290477.

**15** David Eppstein. $K$-best enumeration. *Bull. EATCS*, 115, 2015. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/322.

**16** David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997. doi:10.1145/265910.265914.

**17** S. Feferman and R. L. Vaught. The first order properties of products of algebraic systems. *Fund. Math.*, 47:57–103, 1959.

**18** Greg N. Frederickson. An optimal algorithm for selection in a min-heap. *Inf. Comput.*, 104(2):197–214, 1993. doi:10.1006/inco.1993.1030.

**19** Harold N. Gabow. Two algorithms for generating weighted spanning trees in order. *SIAM J. Comput.*, 6(1):139–150, 1977. doi:10.1137/0206011.

**20** H. W. Hamacher and M. Queyranne. $K$ best solutions to combinatorial optimization problems. *Ann. Oper. Res.*, 4(1-4):123–143, 1985. doi:10.1007/BF02022039.

**21** Naoki Katoh, Toshihide Ibaraki, and Hisashi Mine. An efficient algorithm for K shortest simple paths. *Networks*, 12(4):411–427, 1982. doi:10.1002/net.3230120406.

**22** Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 645–654. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.67.

**23** Jin Y. Yen. Finding the $K$ shortest loopless paths in a network. *Manag. Sci.*, 17(11):712–716, 1971. URL: http://www.jstor.org/stable/2629312.

# DynASP2.5: Dynamic Programming on Tree Decompositions in Action*†

**Johannes K. Fichte[1], Markus Hecher[2], Michael Morak[3], and Stefan Woltran[4]**

1 **Institut für Informationssysteme, TU Wien, Vienna, Austria and Universität Potsdam, Potsdam, Germany**
fichte@dbai.tuwien.ac.at
2 **Institut für Informationssysteme, TU Wien, Vienna, Austria and Universität Potsdam, Potsdam, Germany**
hecher@dbai.tuwien.ac.at
3 **Institut für Informationssysteme, TU Wien, Vienna, Austria**
morak@dbai.tuwien.ac.at
4 **Institut für Informationssysteme, TU Wien, Vienna, Austria**
woltran@dbai.tuwien.ac.at

## Abstract

Efficient, exact parameterized algorithms are a vibrant theoretical research area. Recent solving competitions, such as the PACE challenge, show that there is also increasing practical interest in the parameterized algorithms community. An important research question is whether such algorithms can be built to efficiently solve specific problems in practice, that is, to be competitive with established solving systems. In this paper, we consider Answer Set Programming (ASP), a logic-based declarative modeling and problem solving framework. State-of-the-art ASP solvers generally rely on SAT-based algorithms. In addition, DynASP2, an ASP solver that is based on a classical dynamic programming on tree decompositions, has recently been introduced. DynASP2 outperforms modern ASP solvers when the goal is to count the number of solutions of programs that have small treewidth. However, for quickly finding one solutions, DynASP2 proved uncompetitive. In this paper, we present a new algorithm and implementation, called DynASP2.5, that shows competitive behavior compared to state-of-the-art ASP solvers on problems like Steiner tree for low-treewidth graphs, even when the task is to find just one solution. Our implementation is based on a novel approach that we call multi-pass dynamic programming.

**1998 ACM Subject Classification** I.2.8 Problem Solving, Control Methods, and Search, D.1.6 Logic Programming, F.4.1 Mathematical Logic, I.2.3 Deduction and Theorem Proving, I.2.4 Knowledge Representation Formalisms and Methods

**Keywords and phrases** Parameterized algorithms, Fixed-parameter linear time, Tree decompositions, Multi-pass dynamic programming

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.17

## 1 Introduction

Answer set programming (ASP) is a logic-based declarative modeling language and problem solving framework [13], where a program is defined by a set of rules over propositional atoms

* Research was supported by the Austrian Science Fund (FWF), Grant Y698.
† For an extended version see [9], https://arxiv.org/abs/1706.09370.


and is interpreted under an extended stable model semantics [15]. Problems are usually modeled in ASP in such a way that the solutions (called *answer sets*) of a program directly correspond to the solutions of the considered problem instance. Computational problems for disjunctive, propositional ASP, such as deciding whether a program has an answer set, are complete for the second level of the polynomial hierarchy [8]. In consequence, finding answer sets usually involves a Sat part (finding a model of the program) and an Unsat part (minimality check). A variety of ASP solvers based on techniques of Sat solvers are readily available [3, 11] and have proven to be very successful in solving competitions.

Recently, a dynamic programming based solver (*DynASP2*) that builds upon ideas from parameterized algorithms has been proposed [10]. The running time of the underlying algorithm is double exponential in the treewidth of the input program and linear in its size (a so-called *fixed-parameter linear* algorithm). DynASP2 roughly works as follows. First, it computes a tree decomposition of the incidence graph of the given input program. Second, it solves the program via dynamic programming (DP) on the tree decomposition, traversing the tree exactly once bottom-up. Both the Sat and Unsat tasks are considered in a single pass. Once the root node has been reached, complete solutions for the input program can be constructed, if any exist. The exhaustive nature of DP algorithms, where all potential solutions are computed locally for each node of the tree decomposition, works well when all solutions are indeed needed, e.g., for counting answer sets. However, this approach is not competitive when the task is to construct just one answer set, since space requirements can be quite extensive, resulting in long running times. Another downside is the fact that DP algorithms on tree decompositions may exhibit running times that vary considerably, even on tree decompositions of the exact same width [2].

In this paper, we propose a multi-pass algorithm, called $\texttt{M-DP}_{\mathbb{SINC}}$, for dynamic programming on tree decompositions, as well as a new implementation (DynASP2.5). In contrast to classical DP algorithms for problems on the second level of the polynomial hierarchy, $\texttt{M-DP}_{\mathbb{SINC}}$ traverses the given tree decomposition multiple times in a bottom-up fashion. During the first pass, it computes and stores sets of atoms that are relevant for the Sat part (finding a model of the program) up to the root. In the second pass, it computes and stores sets of atoms that are relevant for the Unsat part (checking for minimality). Finally, in the third pass, it links those sets from past two passes together that might lead to an answer set. This allows us to discard candidates that do not lead to answer sets early on.

In addition, we present technical improvements (including working on non-nice tree decompositions) and employ dedicated customization techniques for selecting tree decompositions. These improvements are the main ingredients to speed up the solving process for DP algorithms. Experiments indicate that DynASP2.5 is competitive, even for quickly finding some answer set, when solving the Steiner tree problem on graphs of low treewidth. In particular, DynASP2.5 are able to solve instances that have an upper bound on the incidence treewidth of 14 (whereas DynASP2 could solve instances of treewidth at most 9) on our benchmark set[1].

**Contributions.** Our main contributions can be summarized as follows:

1. We establish a novel fixed-parameter linear algorithm ($\texttt{M-DP}_{\mathbb{SINC}}$), which works in multiple passes and computes Sat and Unsat parts separately.
2. We present an implementation (DynASP2.5)[2] and an experimental evaluation.

[1] The set is available at `https://github.com/daajoe/dynasp_experiments/tree/ipec2017`.
[2] The sources of our solver are available at `https://github.com/daajoe/dynasp/releases/tag/v2.5.0`.

**Related Work.** An ASP solver (DynASP2) that is based on single pass DP on tree decompositions was recently introduced [10]. The solver is dedicated to counting answer sets for the full ground ASP language. The present paper extends these results by a multi-pass DP algorithm. In contrast to systems that use encodings in Monadic Second-Order (MSO) [14], our approach directly treats ASP. Bliem et al. [5] introduced a multi-pass approach and an implementation (D-FLAT^2) for DP on tree decompositions solving subset minimization tasks. Their approach allows to specify DP algorithms by means of ASP. In D-FLAT^2 one can see ASP as a meta-language to describe what needs to be done at each node of the tree decomposition, whereas our work presents an algorithm dedicated to find some answer set of a program. Further, we require specialized adaptations to the ASP problem semantics, including three valued evaluation of atoms, handling of non-nice tree decompositions, and optimizations in join nodes to be competitive. We use in our solver the heuristic tree decomposition library htd [1]. For other systems we refer to the PACE challenge [7].

## 2 Formal Background

**Tree Decompositions.** Let $G = (V, E)$ be a graph, $T = (N, F, n)$ a rooted tree, and $\chi : N \to 2^V$ a function that maps each node $t \in N$ to a set of vertices. We call the sets $\chi(\cdot)$ *bags* and $N$ the set of nodes. Then, the pair $\mathcal{T} = (T, \chi)$ is a *tree decomposition (TD)* of $G$ if the following conditions hold: (i) for every vertex $v \in V$ there is a node $t \in N$ with $v \in \chi(t)$; (ii) for every edge $e \in E$ there is a node $t \in N$ with $e \subseteq \chi(t)$; and (iii) for any three nodes $t_1, t_2, t_3 \in N$, if $t_2$ lies on the unique path from $t_1$ to $t_3$, then $\chi(t_1) \cap \chi(t_3) \subseteq \chi(t_2)$. We call $\max\{|\chi(t)| - 1 \mid t \in N\}$ the *width* of the TD. The *treewidth* $tw(G)$ of a graph $G$ is the minimum width over all possible TDs of $G$. For some arbitrary but fixed integer $k$ and a graph of treewidth at most $k$, we can compute a TD of width $\leqslant k$ in time $2^{\mathcal{O}(k^3)} \cdot |V|$ [6]. Given a TD $(T, \chi)$ with $T = (N, \cdot, \cdot)$, for a node $t \in N$ we say that type$(t)$ is *leaf* if $t$ has no children; *join* if $t$ has children $t'$ and $t''$ with $t' \neq t''$ and $\chi(t) = \chi(t') = \chi(t'')$; *int* ("introduce") if $t$ has a single child $t'$, $\chi(t') \subseteq \chi(t)$ and $|\chi(t)| = |\chi(t')| + 1$; *rem* ("removal") if $t$ has a single child $t'$, $\chi(t') \supseteq \chi(t)$ and $|\chi(t')| = |\chi(t)| + 1$. If every node $t \in N$ has at most two children, type$(t) \in \{$*leaf*, *join*, *int*, *rem*$\}$, and bags of leaf nodes and the root are empty, then the TD is called *nice*. For every TD, we can compute a nice TD in linear time without increasing the width [6]. Later, we traverse a TD bottom up. Therefore, let post-order$(T, t)$ be the sequence of nodes in post-order of the induced subtree $T' = (N', \cdot, t)$ of $T$ rooted at $t$.

**Answer Set Programming (ASP).** ASP is a declarative modeling and problem solving framework that combines techniques of knowledge representation and database theory. Two of the main advantages of ASP are its expressiveness and, when using non-ground programs, its advanced declarative problem modeling capability. Prior to solving, non-ground programs are usually compiled into ground ones by a grounder. There are classes of non-ground programs that preserve the treewidth of the input instance after grounding [4]. In this paper, we restrict ourselves to ground ASP programs. For a comprehensive introduction, see, e.g., [13]. Let $\ell$, $m$, $n$ be non-negative integers such that $\ell \leq m \leq n$, $a_1, \ldots, a_n$ distinct propositional atoms, and $l \in \{a_1, \neg a_1\}$. A *choice rule* is an expression of the form $\{a_1; \ldots; a_\ell\} \leftarrow a_{\ell+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n$ with the intuitive meaning that some subset of $\{a_1, \ldots, a_\ell\}$ is true if all atoms $a_{\ell+1}, \ldots, a_m$ are true and there is no evidence that any atom of $a_{m+1}, \ldots, a_n$ is true. A *disjunctive rule* is of the form $a_1 \vee \cdots \vee a_\ell \leftarrow a_{\ell+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n$, which, intuitively, means that at least one atom of $a_1, \ldots, a_\ell$ must be true if all atoms $a_{\ell+1}, \ldots, a_m$ are true and there is no evidence that any atom of $a_{m+1}, \ldots, a_n$ is true. A *rule* $r$

is either a disjunctive or a choice rule. Let $H_r := \{a_1, \ldots, a_\ell\}$, $B_r^+ := \{a_{\ell+1}, \ldots, a_m\}$, and $B_r^- := \{a_{m+1}, \ldots, a_n\}$. Usually, for a rule $r$, if $B_r^- \cup B_r^+ = \emptyset$, we simply write $H_r$ instead of $H_r \leftarrow$. For a rule $r$, let $\mathrm{at}(r) := H_r \cup B_r^+ \cup B_r^-$ denote its *atoms* and $B_r := B_r^+ \cup \{\neg b \mid b \in B_r^-\}$ its *body*. A *program* $P$ is a set of rules, where $\mathrm{at}(P) := \bigcup_{r \in P} \mathrm{at}(r)$ denotes its atoms. A set $M \subseteq \mathrm{at}(P)$ *satisfies* a rule $r$ if (i) $r$ is a disjunctive rule and $(H_r \cup B_r^-) \cap M \neq \emptyset$ or $B_r^+ \not\subseteq M$ or (ii) $r$ is a choice rule. Note that choice rules are always satisfied. $M$ is a (classical) model of $P$, denoted by $M \vDash P$, if $M$ satisfies every rule $r \in P$. The *reduct* of a rule $r$ with respect to $M$, denoted by $r^M$, is defined (i) for a choice rule $r$ as the set $\{a \leftarrow B_r^+ \mid a \in H_r \cap M, B_r^- \cap M = \emptyset\}$ of rules and (ii) for a disjunctive rule $r$ as the singleton $\{H_r \leftarrow B_r^+ \mid B_r^- \cap M = \emptyset\}$. $P^M := \bigcup_{r \in P} r^M$ is called the *(GL) reduct* of $P$ with respect to $M$. A set $M \subseteq \mathrm{at}(P)$ is an *answer set* of a program $P$, if $M \vDash P$ and there does not exist a proper subset $M' \subsetneq M$, such that $M' \vDash P^M$.

▶ **Example 1.** Consider program $P$, consisting of the following nine rules:
$P = \{\overbrace{\{e_{ab}\}}^{r_{ab}}; \overbrace{\{e_{bc}\}}^{r_{bc}}; \overbrace{\{e_{cd}\}}^{r_{cd}}; \overbrace{\{e_{ad}\}}^{r_{ad}}; \overbrace{a_b \leftarrow e_{ab}}^{r_b}; \overbrace{a_d \leftarrow e_{ad}}^{r_d}; \overbrace{a_c \leftarrow a_b, e_{bc}}^{r_{c1}}; \overbrace{a_c \leftarrow a_d, e_{cd}}^{r_{c2}}; \overbrace{\leftarrow \neg a_c}^{r_{\neg c}}\}$. The set $A = \{e_{ab}, e_{bc}, a_b, a_c\}$ is an answer set of $P$, as $\{e_{ab}, e_{bc}, a_b, a_c\}$ is a minimal model of the reduct $P^A = \{e_{ab} \leftarrow;\ e_{bc} \leftarrow;\ a_b \leftarrow e_{ab};\ a_d \leftarrow e_{ad};\ a_c \leftarrow a_b, e_{bc};\ a_c \leftarrow a_d, e_{cd}\}$. Now, consider program $R = \{a \vee c \leftarrow b;\ b \leftarrow c, \neg g;\ c \leftarrow a;\ b \vee c \leftarrow e;\ h \vee i \leftarrow g, \neg c;\ a \vee b;\ g \leftarrow \neg i;\ c;\ \{d\} \leftarrow g\}$. The set $B = \{b, c, d, g\}$ is an answer set of $R$, since $B$ is a minimal model of the reduct $R^B = \{a \vee c \leftarrow b;\ c \leftarrow a;\ b \vee c \leftarrow e;\ a \vee b;\ g;\ c;\ d \leftarrow g\}$.

In this paper, we mainly consider the *output answer set problem*, that is, output an answer set for an ASP program. The decision version of this problem is $\Sigma_2^p$-complete.

**Graph Representations of Programs.** In order to use TDs for ASP solving, we need dedicated graph representations of programs. The *incidence graph* $I(P)$ of $P$ is the bipartite graph that has the atoms and rules of $P$ as vertices and an edge $a\,r$ if $a \in \mathrm{at}(r)$ for some rule $r \in P$ [10]. The *semi-incidence graph* $S(P)$ of $P$ is a graph that has the atoms and rules of $P$ as vertices and (i) an edge $a\,r$ if $a \in \mathrm{at}(r)$ for some rule $r \in P$ as well as (ii) an edge $a\,b$ for distinct atoms $a, b \in H_r$ where $r \in P$ is a choice rule. Since, for every program $P$, the incidence graph $I(P)$ is a subgraph of the semi-incidence graph, we have that $tw(I(P)) \leq tw(S(P))$. Further, by definition of TDs and the construction of a semi-incidence graph, head atoms of each choice rule occur together in at least one bag of the TD.

**Sub-programs.** Let $\mathcal{T} = (T, \chi)$ be a nice TD of the semi-incidence graph $S(P)$ of a program $P$. Also, let $T = (N, \cdot, n)$ and $t \in N$. The *bag-program* is defined as $P_t := P \cap \chi(t)$. The set $\mathrm{at}_{\leq t} := \{a \mid a \in \mathrm{at}(P) \cap \chi(t'), t' \in \text{post-order}(T, t)\}$ is called *atoms below* $t$, the *program below* $t$ is defined as $P_{\leq t} := \{r \mid r \in P_{t'}, t' \in \text{post-order}(T, t)\}$, and the *program strictly below* $t$ is $P_{<t} := P_{\leq t} \setminus P_t$. Since $\chi(n) = \emptyset$, it holds that $P_{\leq n} = P_{<n} = P$ and $\mathrm{at}_{\leq n} = \mathrm{at}(P)$.

## 3 A Single Pass DP Algorithm

DynASP2 [10], a dynamic programming-based ASP solver, splits the input program $P$ into bag-programs based on the structure of a given nice tree decomposition for $P$ and evaluates each bag-program in turn, storing results in tables for each TD node. The algorithm works as shown in Figure 1 (following the red DynASP2 arrow) and executes the following steps:

1. Construct a graph representation $G(P)$ of the given input program $P$.
2. Compute a TD $\mathcal{T}$ of the graph $G(P)$ by means of some heuristic, thereby decomposing $P$ into several smaller bag-programs and fixing an ordering in which $P$ will be evaluated.

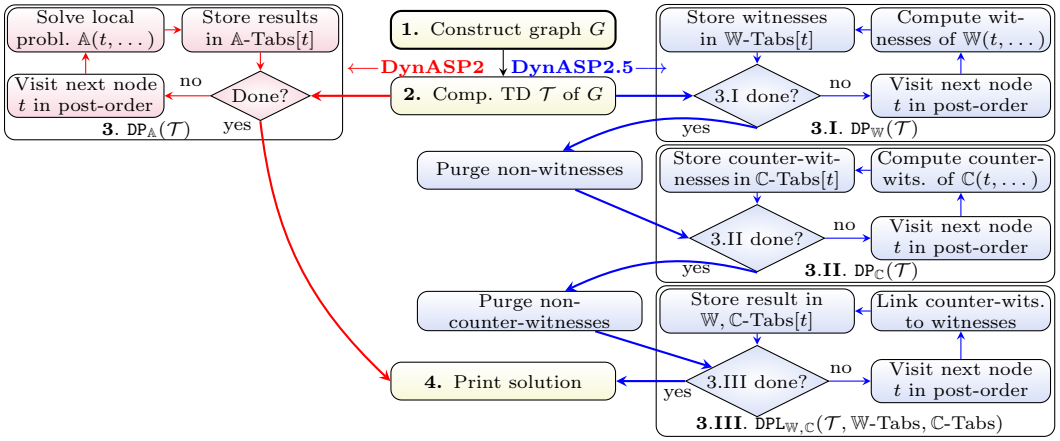



**Figure 1** Control flow for DP-based ASP solvers DynASP2 (left) and DynASP2.5 (right).

**Listing 1:** Algorithm $\mathtt{DP}_{\mathbb{A}}(\mathcal{T})$ for Dynamic Programming on TD $\mathcal{T}$ for ASP [9].

**In:** Table algorithm $\mathbb{A}$, nice TD $\mathcal{T} = (T, \chi)$ with $T = (N, \cdot, n)$ of $G(P)$ according to $\mathbb{A}$.
**Out:** $\mathbb{A}$-Tabs: maps each TD node $t \in T$ to some computed table $\tau_t$.

```
for iterate t in post-order(T,n) do
    Child-Tabs_t := {A-Tabs[t'] | t' is a child of t in T}
    A-Tabs[t] ← A(t, χ(t), P_t, at_≤t, Child-Tabs_t)
```

3. Algorithm $\mathtt{DP}_{\mathbb{A}}(\mathcal{T})$ (see Listing 1 above) specifies the general scheme for this step, assuming that an algorithm $\mathbb{A}$, which strongly depends on the graph representation, is given. Such an algorithm $\mathbb{A}$ is called a *table algorithm* that specifies, how the individual bag-programs for each tree node are evaluated. $\mathtt{DP}_{\mathbb{A}}(\mathcal{T})$ works as follows: Traverse the tree decomposition $\mathcal{T}$ in post-order. For every node $t \in T$ in the tree decomposition $\mathcal{T} = ((T, E, n), \chi)$, run $\mathbb{A}$ to compute the table for node $t$ (denoted $\mathbb{A}$-Tabs$[t]$). Intuitively, each tuple, which we refer to as *row*, in the table represents a witness for the existence of a solution for the bag-program at node $t$. $\mathbb{A}$-Tabs$[t]$ is computed by taking, as input, the tables of the child nodes of $t$, and extending them according to the bag-program $P_t$. Each row in $\mathbb{A}$-Tabs$[t]$ consists of a *witness set* (a set of atoms relevant for the SAT part of the problem), and a family of *counter-witness sets* (sets of atoms relevant for the UNSAT part) [10]. This directly follows the definition of answer sets, namely, being models of $P$ and subset-minimal with respect to $P^M$.
4. For root node $n$, check if a "solution row" exists in table $\mathbb{A}$-Tabs$[n]$ and print the solution to the output ASP problem.

With the above general algorithm in mind, we are now ready to propose $\mathbb{SINC}$, a new table algorithm for solving ASP on the semi-incidence graph (see Listing 2). $\mathtt{DP}_{\mathbb{SINC}}$ merges two earlier algorithms for the primal and incidence graph [10].

As in the general approach, $\mathbb{SINC}$ computes and stores witness sets, and their corresponding counter-witness sets. However, in addition, for each witness set and counter-witness set, respectively, we need to store so-called *satisfiability states* (or *sat-states*, for short), since the atoms of a rule may no longer be contained in one single bag of the TD of the semi-incidence graph. Therefore, we need to remember in each TD node, how much of a rule is already satisfied. The following describes this in more detail.

**Listing 2:** Table algorithm $\mathbb{SINC}(t, \chi_t, P_t, \text{at}_{\leq t}, \text{Child-Tabs}_t)$.

**In:** Bag $\chi_t$, bag-program $P_t$, atoms-below $\text{at}_{\leq t}$, child tables Child-Tabs$_t$ of $t$. **Out:** Tab. $\tau_t$.

1 **if** type$(t) =$ *leaf* **then** $\tau_t \leftarrow \{\langle \emptyset, \emptyset, \emptyset \rangle\}$ ; `/* For Abbreviations see below. */`

2 **else if** type$(t) =$ *int,* $a \in \chi_t \setminus P_t$ *is introduced and* $\tau' \in$ *Child-Tabs*$_t$ **then**

3 $\quad \tau_t \leftarrow \{\langle M_a^+, \sigma \cup \text{SatPr}(\dot{P}_t^{(t)}, M_a^+),\ \{\langle C_a^+, \rho \cup \text{SatPr}(\dot{P}_t^{(t,M_a^+)}, C_a^+)\rangle \mid \langle C, \rho\rangle \in \mathcal{C}\} \cup$

4 $\quad \{\langle C, \rho \cup \text{SatPr}(\dot{P}_t^{(t,M_a^+)}, C)\rangle \mid \langle C, \rho\rangle \in \mathcal{C}\} \cup \{\langle M, \sigma \cup \text{SatPr}(\dot{P}_t^{(t,M_a^+)}, M)\rangle\}\rangle \mid \langle M, \sigma, \mathcal{C}\rangle \in \tau'\}$

5 $\quad \cup\ \{\langle M, \sigma \cup \text{SatPr}(\dot{P}_t^{(t)}, M),\ \{\langle C, \rho \cup \text{SatPr}(\dot{P}_t^{(t,M)}, C)\rangle \mid \langle C, \rho\rangle \in \mathcal{C}\}\ \rangle \mid \langle M, \sigma, \mathcal{C}\rangle \in \tau'\}$

6 **else if** type$(t) =$ *int,* $r \in \chi_t \cap P_t$ *is introduced and* $\tau' \in$ *Child-Tabs*$_t$ **then**

7 $\quad \tau_t \leftarrow \{\langle M, \sigma \cup \text{SatPr}(\{\dot{r}\}^{(t)}, M),\ \{\langle C, \rho \cup \text{SatPr}(\{\dot{r}\}^{(t,M)}, C)\rangle \mid \langle C, \rho\rangle \in \mathcal{C}\}\rangle \mid \langle M, \sigma, \mathcal{C}\rangle \in \tau'\}$

8 **else if** type$(t) =$ *rem,* $a \notin \chi_t$ *is removed atom and* $\tau' \in$ *Child-Tabs*$_t$ **then**

9 $\quad \tau_t \leftarrow \{\langle M_a^-, \sigma,\ \{\langle C_a^-, \rho\rangle \mid \langle C, \rho\rangle \in \mathcal{C}\}\rangle \mid \langle M, \sigma, \mathcal{C}\rangle \in \tau'\}$

10 **else if** type$(t) =$ *rem,* $r \notin \chi_t$ *is removed rule and* $\tau' \in$ *Child-Tabs*$_t$ **then**

11 $\quad \tau_t \leftarrow \{\langle M, \sigma_r^-,\ \{\langle C, \rho_r^-\rangle \mid \langle C, \rho\rangle \in \mathcal{C}, r \in \rho\}\rangle \mid \langle M, \sigma, \mathcal{C}\rangle \in \tau', r \in \sigma\}$

12 **else if** type$(t) =$ *join and* $\tau', \tau'' \in$ *Child-Tabs*$_t$ *with* $\tau' \neq \tau''$ **then**

13 $\quad \tau_t \leftarrow \{\langle M, \sigma' \cup \sigma'',\ \{\langle C, \rho' \cup \rho''\rangle \mid \langle C, \rho'\rangle \in \mathcal{C}', \langle C, \rho''\rangle \in \mathcal{C}''\} \cup \{\langle M, \rho \cup \sigma''\rangle \mid \langle M, \rho\rangle \in \mathcal{C}'\} \cup$

14 $\quad \{\langle M, \sigma' \cup \rho\rangle \mid \langle M, \rho\rangle \in \mathcal{C}''\}\rangle \mid \langle M, \sigma', \mathcal{C}'\rangle \in \tau', \langle M, \sigma'', \mathcal{C}''\rangle \in \tau''\}$

For set $S$ and element $s$, we denote $S_s^+ \leftarrow S \cup \{s\}$ and $S_s^- \leftarrow S \setminus \{s\}$.

By definition of TDs and the semi-incidence graph, for every atom $a$ and every rule $r$ of a program, it is true that if atom $a$ occurs in rule $r$, then $a$ and $r$ occur together in at least one bag of the TD. As a consequence, the table algorithm encounters every occurrence of an atom in any rule. In the end, on removal of $r$, we have to ensure that $r$ is among the rules that are already satisfied. However, we need to keep track of whether a witness set satisfies a rule, because not all atoms that occur in a rule occur together in a bag. Hence, when our algorithm traverses the TD and an atom is removed we still need to store this sat-state, as setting this atom to a certain truth value influences the satisfiability of the rule. Since the semi-incidence graph contains a clique on every set $A$ of atoms that occur together in a choice rule head, those atoms $A$ occur together in a bag in every TD of the semi-incidence graph. For that reason, we do not need to incorporate choice rules into the satisfiability state, in contrast to the algorithm for the incidence graph [10].

In Algorithm $\mathbb{SINC}$ (detailed in Listing 2), a row $u$ in the table $\tau_t$ is a triple $\langle M, \sigma, \mathcal{C}\rangle$. The set $M \subseteq \text{at}(P) \cap \chi(t)$ represents a witness set. The family $\mathcal{C}$ of rows represents counter-witnesses, which we will discuss in more detail below. The sat-state $\sigma$ for $M$ represents rules of $\chi(t)$ satisfied by a superset of $M$. Hence, $M$ witnesses a model $M' \supseteq M$ where $M' \vDash P_{<t} \cup \sigma$. For that reason, a witness set together with its sat-state is called a *witness*. We use the binary operator $\cup$ to combine sat-states, which ensures that rules satisfied in at least one operand remain satisfied. For a node $t$, our algorithm considers a local-program depending on the bag $\chi(t)$. Intuitively, this provides a local view on the program.

▶ **Definition 2.** Let $P$ be a program, $\mathcal{T} = (\cdot, \chi)$ a TD of $S(P)$, $t$ a node of $\mathcal{T}$ and $R \subseteq P_t$. The *local-program* $R^{(t)}$ is obtained from $R \cup \{\leftarrow B_r \mid r \in R \text{ is a choice rule, } H_r \subsetneq \text{at}_{\leq t}\}$[3] by removing all literals $a$ and $\neg a$ from every rule where $a \notin \chi(t)$.

▶ **Example 3.** Observe $P_{t_4}^{(t_4)} = \{\leftarrow e_{bc}, r_b\}$ and $P_{t_5}^{(t_5)} = \{c \leftarrow\}$ for $P_{t_4}$ and $P_{t_5}$ of Figure 2.

Since the local-program $P^{(t)}$ depends on the considered node $t$, we may have different local-programs for for node $t$ and its child $t'$. In particular, the programs $\{r\}^{(t)}$ and $\{r\}^{(t')}$ might already differ for a rule $r \in \chi(t) \cap \chi(t')$. In consequence for satisfiability with respect

[3] We require to add $\{\leftarrow B_r \mid r \in R \text{ is a choice rule, } H_r \subsetneq \text{at}_{\leq t}\}$ in order to decide satisfiability for corner cases of choice rules involving counter-witnesses of Line 3 in Listing 2.

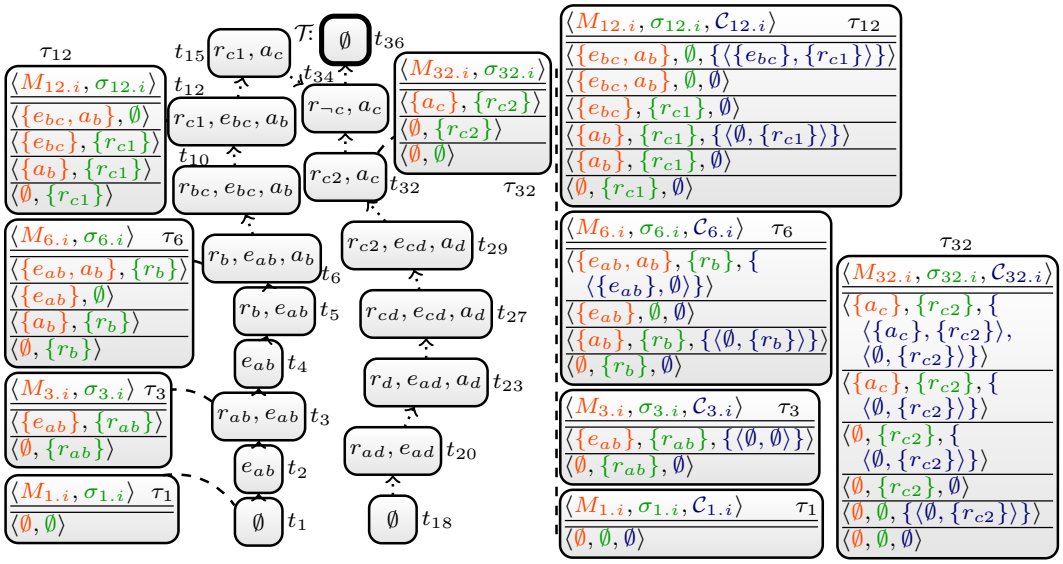



**Figure 2** A TD $\mathcal{T}$ of the semi-incidence graph $S(P)$ for program $P$ from Example 1 (center). Selected DP tables after $\mathtt{DP}_{\mathbb{MOD}}$ (left) and after $\mathtt{DP}_{\mathbb{SINC}}$ (right) for nice TD $\mathcal{T}$.

to sat-states, we need to keep track of a representative of a rule. We achieve this by a function $\dot{R}^{(t)} : R \to 2^{R^{(t)}}$ that maps a rule in $R \subseteq P_t$ for bag-program $P_t$ to its local-program, i.e., $\dot{R}^{(t)}(r) := \{r\}^{(t)}$ for $r \in R$. When we compute newly satisfied rules for witness set $M$ and a set $R$ of rules, we use the function $\dot{R}^{(t)}$. Formally, $\mathrm{SatPr}(\dot{R}^{(t)}, M) := \{r \mid (r, S) \in \dot{R}^{(t)}, M \vDash S\}$ for $M \subseteq \chi(t) \setminus P_t$ using program $S = \dot{R}^{(t)}(r)$ constructed by $\dot{R}^{(t)}$ and $r \in R$.

Example 4 provides an explanation of the part of $\mathbb{SINC}$ that deals with witnesses only. Therefore, the resulting algorithm $\mathbb{MOD}$ computes only models and is obtained from $\mathbb{SINC}$, by taking only the first two row positions into account (red and green text in Listing 2). The remaining position (blue text) can be seen as an algorithm $\mathbb{CMOD}$ that computes counter-witnesses (see [9, Ex. 4]). Note that we discuss selected cases, and we assume that each row in a table $\tau_t$ is identified by a number, i.e., row $i$ corresponds to $u_{t.i} = \langle M_{t.i}, \sigma_{t.i} \rangle$.

▶ **Example 4.** Consider program $P$ from Example 1, TD $\mathcal{T} = (\cdot, \chi)$ in Figure 2, and the tables $\tau_1, \ldots, \tau_{34}$, which illustrate computation results obtained during post-order traversal of $\mathcal{T}$ by $\mathtt{DP}_{\mathbb{MOD}}$. Figure 2 (left) does not show every intermediate node of TD $\mathcal{T}$. Table $\tau_1 = \{\langle \emptyset, \emptyset \rangle\}$ as $\mathrm{type}(t_1) = \mathit{leaf}$ (see Line 1 in Listing 2). Table $\tau_3$ is obtained via introducing rule $r_{ab}$, after introducing atom $e_{ab}$ ($\mathrm{type}(t_2) = \mathrm{type}(t_3) = \mathit{int}$). It contains two rows due to two possible truth assignments using atom $e_{ab}$ (Line 3–5). Observe that rule $r_{ab}$ is satisfied in both rows $M_{3.1}$ and $M_{3.2}$, since the head of choice rule $r_{ab}$ is in $\mathrm{at}_{\leq t_3}$ (see Line 7 and Definition 2). Intuitively, whenever a rule $r$ is proven to be satisfied, sat-state $\sigma_{t.i}$ marks $r$ as *satisfied* since an atom of a rule of $S(P)$ might only occur in one TD bag. Consider table $\tau_4$ with $\mathrm{type}(t_4) = \mathit{rem}$ and $r_{ab} \in \chi(t_3) \setminus \chi(t_4)$. By definition of TDs of $S(P)$, we have encountered every occurrence of any atom in $r_{ab}$. Thus, $\mathbb{MOD}$ enforces that only rows where $r_{ab}$ is marked satisfied in $\tau_3$, are considered for table $\tau_4$. The resulting table $\tau_4$ consists of rows of $\tau_3$ with $\sigma_{4.i} = \emptyset$, where rule $r_{ab}$ is proven satisfied ($r_{ab} \in \sigma_{3.1}, \sigma_{3.2}$, see Line 11). Note that between nodes $t_6$ and $t_{10}$, an atom and rule remove as well as an atom and rule introduce node is placed. Observe that the second row $u_{6.2} = \langle M_{6.2}, \sigma_{6.2} \rangle \in \tau_6$ does not have a "successor row" in $\tau_{10}$, since $r_b \notin \sigma_{6.2}$. Intuitively, join node $t_{34}$ joins only common witness sets in $\tau_{17}$ and $\tau_{33}$ with $\chi(t_{17}) = \chi(t_{33}) = \chi(t_{34})$. In general, a join node marks rules satisfied, which are marked satisfied in at least one child (see Line 13–14).

**Listing 3:** Algorithm $\texttt{DPL}_{\mathbb{W},\mathbb{C}}(\mathcal{T}, \mathbb{W}\text{-Tabs}, \mathbb{C}\text{-Tabs})$ for linking counter-witnesses to witnesses.

**In:** Nice TD $\mathcal{T} = (T, \chi)$ with $T = (N, \cdot, n)$ of a graph $S(P)$, mappings $\mathbb{W}$-Tabs$[\cdot]$, $\mathbb{C}$-Tabs$[\cdot]$.
**Out:** $\mathbb{W}, \mathbb{C}$-Tabs: maps $t \in T$ to some pair $(\tau_t^{\mathbb{W}}, \tau_t^{\mathbb{C}})$ with $\tau_t^{\mathbb{W}} \in \mathbb{W}\text{-Tabs}[t], \tau_t^{\mathbb{C}} \in \mathbb{C}\text{-Tabs}[t]$.

1 $\text{Child-Tabs}_t := \{\mathbb{W}, \mathbb{C}\text{-Tabs}[t'] \mid t' \text{ is a child of } t \text{ in } T\}$

`/* Get for a node t tables of (preceding) combined child rows (CCR) */`

2 $\text{CCR}_t := \hat{\Pi}_{\tau' \in \text{Child-Tabs}_t} \tau'$ `/* For Abbreviations see below. */`

`/* Get for a row u⃗ its combined child rows (origins) */`

3 $\text{orig}_t(\vec{u}) := \{S \mid S \in \text{CCR}_t, \vec{u} \in \tau, \tau = \mathbb{W}(t, \chi(t), P_t, \text{at}_{\leq t}, f_w(S))\}$

`/* Get for a table S of combined child rows its successors (evolution) */`

4 $\text{evol}_t(S) := \{\vec{u} \mid \vec{u} \in \tau, \tau = \mathbb{C}(t, \chi(t), P_t, \text{at}_{\leq t}, \tau'), \tau' \in S\}$

5 **for** iterate $t$ *in* post-order$(T,n)$ **do**

`/* Compute counter-witnesses (≺-smaller rows) for a witness set M */`

6 $\text{subs}_{\prec}(f, M, S) := \{\vec{u} \mid \vec{u} \in \mathbb{C}\text{-Tabs}[t], \vec{u} \in \text{evol}_t(f(S)), \vec{u} = \langle C, \cdots \rangle, C \prec M\}$

`/* Link each witness u⃗ to its counter-witnesses and store the results */`

7 $\mathbb{W}, \mathbb{C}\text{-Tabs}[t] \leftarrow \{(\vec{u}, \text{subs}_{\subseteq}(f_w, M, S) \cup \text{subs}_{\subseteq}(f_{cw}, M, S)) \mid \vec{u} \in \mathbb{W}\text{-Tabs}[t], \vec{u} = \langle M, \cdots \rangle, S \in \text{orig}_t(u)\}$

For set $I = \{1, \ldots, n\}$ and sets $S_i$, we define $\Pi_{i \in I} S_i := S_1 \times \cdots \times S_n = \{(s_1, \ldots, s_n) : s_i \in S_i\}$. Moreover, for $\Pi_{i \in I} S_i$, let $\hat{\Pi}_{i \in I} S_i := \{\{\{s_1\}, \ldots, \{s_n\}\} \mid (s_1, \ldots, s_n) \in \Pi_{i \in I} S_i\}$. If for each $S \in \hat{\Pi}_{i \in I} S_i$ and $\{s_i\} \in S$, $s_i$ is a pair with a witness and a counter-witness part, let $f_w(S) := \bigcup_{\{(W_i, C_i)\} \in S} \{\{W_i\}\}$ and $f_{cw}(S) := \bigcup_{\{(W_i, C_i)\} \in S} \{\{C_i\}\}$ restrict $S$ to the witness or counter-witness parts, respectively.

Since we already explained how to obtain models, we only briefly describe how to compute counter-witnesses. Family $\mathcal{C}$ consists of rows $(C, \rho)$, where $C \subseteq \text{at}(P) \cap \chi(t)$ is a counter-witness set to $M$. Similar to the sat-state $\sigma$, the sat-state $\rho$ for $C$ under $M$ represents whether rules of the GL reduct $P_t^M$ are satisfied by a superset of $C$. A counter-witness set together with its sat-state is called a *counter-witness*. Thus, $C$ witnesses the existence of $C' \subsetneq M'$ satisfying $C' \vDash (P_{<t} \cup \rho)^{M'}$ since $M$ witnesses a model $M' \supseteq M$ where $M' \vDash P_{<t}$. In consequence, there exists an answer set of $P$ if the root table contains $\langle \emptyset, \emptyset, \emptyset \rangle$.

In order to decide the satisfiability of counter-witness sets, we require local-reducts similar to local-programs (see Definition 2 and below).

▶ **Definition 5.** Let $P$ be a program, $\mathcal{T} = (\cdot, \chi)$ be a TD of $S(P)$, $t$ be a node of $\mathcal{T}$, $R \subseteq P_t$ and $M \subseteq \text{at}(P)$. We define *local-reduct* $R^{(t,M)}$ as $[R^{(t)}]^M$ and $\dot{R}^{(t,M)} : R \to 2^{R^{(t,M)}}$ as $\dot{R}^{(t,M)}(r) := \{r\}^{(t,M)}$ for any $r \in R$.

Note that one can now easily refine $\text{SatPr}(\cdot, \cdot)$ such that it takes as first argument arbitrary functions mapping from rules to programs. In particular, one can then pass the function $\dot{R}^{(t,M)}$ for a set $R$ of rules, and a witness set $M \subseteq \chi(t) \setminus P_t$, as used in Listing 2.

▶ **Proposition 6** ($\star$, c.f. [10]). *Let $P$ be a program and $k := tw(S(P))$. Then, the algorithm $\texttt{DP}_{\mathbb{SINC}}$ runs in time $\mathcal{O}(2^{2^{k+2}} \cdot \|S(P)\|)$ and is correct.*

## 4 DynASP2.5: Implementing a III Pass DP Algorithm

The classical DP algorithm $\texttt{DP}_{\mathbb{SINC}}$ (Step 3 of Figure 1) follows a single pass approach. It computes both witnesses and counter-witnesses in one step by traversing the given TD exactly once. In particular, it exhaustively stores all potential counter-witnesses, even those where the associated witness does not lead to a solution at the root node. In addition, there

[5] Due to space limitations, proofs of statements marked with "$\star$" have been omitted.

can be a high number of duplicates among the counter-witnesses, which are stored separately. In this section, we propose a multi-pass algorithm, $\texttt{M-DP}_{\mathbb{SINC}}$, for DP on TDs and a new implementation (DynASP2.5), which tackles this issue by adapting and extending concepts for DP on TDs presented in [5]. Our novel algorithm allows for an early cleanup (purging) of witnesses that do not lead to answer sets. As a consequence, this (i) avoids to construct expendable counter-witnesses. Moreover, multiple passes enable us to store witnesses and counter-witnesses separately which, in turn, (ii) avoids storing duplicates of counter-witnesses and (iii) allows for highly space-efficient data structures (pointers) in practice when linking witnesses and counter-witnesses together. Figure 1 (following the blue arrows) presents the control flow of the new multi-pass approach *DynASP2.5*, where $\texttt{M-DP}_{\mathbb{SINC}}$ introduces a much more elaborate computation in Step 3 (cf. Figure 1).

## 4.1 The Algorithm

Algorithm $\texttt{M-DP}_{\mathbb{SINC}}$ executed as Step 3 runs $\texttt{DP}_{\mathbb{MOD}}$, $\texttt{DP}_{\mathbb{CMOD}}$ and new Algorithm $\texttt{DPL}_{\mathbb{MOD},\mathbb{CMOD}}$ in three respective passes (3.I, 3.II, and 3.III) as follows:

**3.I.** First, we run the algorithm $\texttt{DP}_{\mathbb{MOD}}$, which computes, via a bottom-up traversal, a table $\mathbb{MOD}$-Tabs$[t]$ of witnesses for every node $t$ in the tree decomposition. Then, via a top-down traversal, it purges those witnesses from tables $\mathbb{MOD}$-Tabs$[t]$ that do not extend to a witness in the table for the parent node; these witnesses can never be used to construct a model (nor answer set) of the program.

**3.II.** For this step, let $\mathbb{CMOD}$ be a table algorithm computing only counter-witnesses of $\mathbb{SINC}$ (blue parts of Listing 2). We execute $\texttt{DP}_{\mathbb{CMOD}}$, which computes all counter-witnesses for all the witnesses at once and stores the resulting tables in $\mathbb{CMOD}$-Tabs$[\cdot]$. For every node $t$, table $\mathbb{CMOD}$-Tabs$[t]$ contains possible counter-witnesses for subset-minimality. Again, irrelevant rows are purged.

**3.III.** Finally, via a bottom-up traversal, for every node $t$ in the TD, witnesses and counter-witnesses are linked using algorithm $\texttt{DPL}_{\mathbb{MOD},\mathbb{CMOD}}$ (see Listing 3). $\texttt{DPL}_{\mathbb{MOD},\mathbb{CMOD}}$ takes previous results and maps rows in $\mathbb{MOD}$-Tabs$[t]$ to a set of rows in $\mathbb{CMOD}$-Tabs$[t]$.

We already explained the table algorithms $\texttt{DP}_{\mathbb{MOD}}$ and $\texttt{DP}_{\mathbb{CMOD}}$ in the previous section. The main part of our multi-pass algorithm is the algorithm $\texttt{DPL}_{\mathbb{MOD},\mathbb{CMOD}}$ based on the general algorithm $\texttt{DPL}_{\mathbb{W},\mathbb{C}}$ (Listing 3) with $\mathbb{W} = \mathbb{MOD}$, $\mathbb{C} = \mathbb{CMOD}$, which links those separate tables together. Before we quickly discuss the core of $\texttt{DPL}_{\mathbb{W},\mathbb{C}}$ in Line 5–7, note that Line 2–4 introduce auxiliary definitions. Line 2 combines rows of the child nodes of given node $t$, which is achieved by a product over sets where we drop the order and keep sets only. For a row $\vec{u}$, Line 3 determines its preceding combined rows that lead to $\vec{u}$, using table algorithm $\mathbb{W}$. Via algorithm $\mathbb{C}$, Line 4 derives the succeeding rows (called *evolution* rows) of a certain child row combination $\tau'$ (called *origin* row). In the actual implementation, origin and evolution rows are not computed, but represented via pointer data structures, directly linking to $\mathbb{W}$-Tabs$[\cdot]$ and $\mathbb{C}$-Tabs$[\cdot]$, respectively. Then, the table algorithm $\texttt{DPL}_{\mathbb{W},\mathbb{C}}$ applies a post-order traversal and links witnesses to counter-witnesses in Line 7. $\texttt{DPL}_{\mathbb{W},\mathbb{C}}$ searches for origins (orig) of a witness $\vec{u}$, uses the counter-witnesses ($f_{cw}$) linked to these origins, and then determines the evolution (procedure evol) in order to derive counter-witnesses (procedure subs) of $\vec{u}$.

▶ **Example 7.** Let $k$ be some integer and $P_k$ be the program that consists of the rules $r_c := \{a_1, \cdots, a_k\} \leftarrow f$, $r_2 := \leftarrow \neg a_2$, …, $r_k := \leftarrow \neg a_k$, and $r_f := \leftarrow \neg f$ and $r_{cf} := \{f\} \leftarrow$. The rules $r_2, \ldots, r_k$ simulate that only specific subsets of $\{a_1, \cdots, a_k\}$ are allowed. Rules $r_f$ and $r_{cf}$ enforce that $f$ is set to true. Let $\mathcal{T} = (T, \chi, t_3)$ be a TD of the semi-incidence graph $S(P_k)$ of program $P_k$ where $T = (V, E)$ with $V = \{t_1, t_2, t_3\}$, $E = \{(t_1, t_2), (t_2, t_3)\}$,

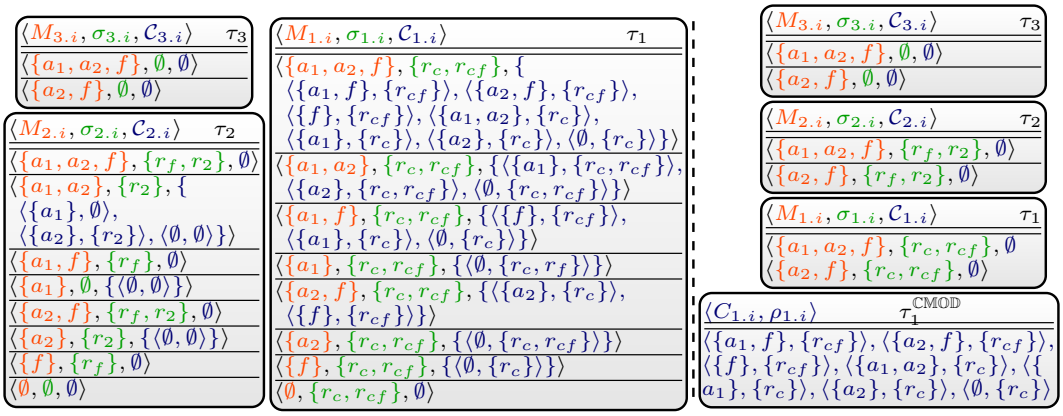



**Figure 3** Selected DP tables after DP$_{\mathbb{SINC}}$ (left) and after M-DP$_{\mathbb{SINC}}$ (right) for TD $\mathcal{T}$.

$\chi(t_1) = \{a_1, \cdots, a_k, f, r_c, r_{cf}\}$, $\chi(t_2) = \{a_1, \cdots, a_k, r_2, \cdots, r_k, r_f\}$, and $\chi(t_3) = \emptyset$. Figure 3 (left) illustrates the tables for program $P_2$ after DP$_{\mathbb{SINC}}$, whereas Figure 3 (right) presents tables after M-DP$_{\mathbb{SINC}}$ was run, which, mainly due to cleanup, are exponentially smaller in $k$. Observe that in Pass 3.II, M-DP$_{\mathbb{SINC}}$ "temporarily" materializes counter-witnesses for $\tau_1$ only, presented in table $\tau_1^{\text{CMOD}}$. Hence, using multi-pass algorithm M-DP$_{\mathbb{SINC}}$ results in an exponential speedup. Note that we can extend the program such that we have the same effect for a TD of minimum width and even if we take the incidence graph. The program $P_k$ and the TD $\mathcal{T}$ also reveal that a different TD of the same width, where $f$ occurs very early in the bottom-up traversal, would result in a smaller table $\tau_1$ even when running DP$_{\mathbb{SINC}}$.

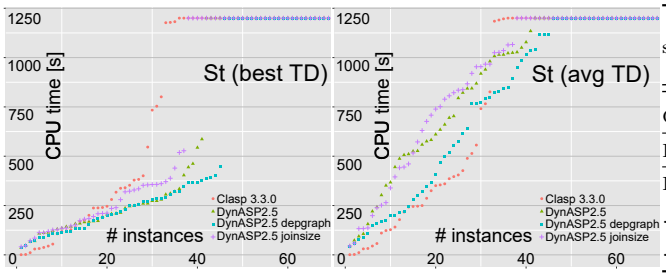
▶ **Theorem 8** ($\star$). *For a program $P$ of semi-incidence treewidth $k = tw(S(P))$, the algorithm* M-DP$_{\mathbb{SINC}}$ *is correct and runs in time $\mathcal{O}(2^{2^{k+2}} \cdot \|P\|)$.*

## 4.2 Implementation Details

Efficient implementations of dynamic programming algorithms on TDs are not a by-product of computational complexity theory and involve tuning and sophisticated algorithm engineering. For that reason, we present additional details about implementing the M-DP$_{\mathbb{SINC}}$ algorithm into our prototypical multi-pass solver DynASP2.5.

Even though *normalizing* a TD (computing a nice TD) can be achieved without increasing its width, a normalization may artificially introduce additional atoms. Normalization causes several additional intermediate join nodes among such artificially introduced atoms requiring a significant amount of total unnecessary computation in practice. That is why, we use non-nice tree decompositions. In order to still ensure the theoretical runtime bounds, we limit the number of children per node to a constant. Moreover, linking counter-witnesses to witnesses efficiently is crucial. The main challenge is to deal with situations where a witness might be linked to a different family of counter-witnesses depending on different predecessors of the row (hidden in set notation of Line 9 in Listing 3). In these cases, DynASP2.5 eagerly creates a "clone" in form of a very light-weighted proxy to the original row and ensures that only the original row (if at all required) serves as a counter-witness during Pass 3. Together with efficient caches of counter-witnesses, DynASP2.5 reduces the overhead caused by clones in practice.

Dedicated data structures are vital. In DynASP2.5, sets of witnesses and satisfied rules are represented via constant-size bit vectors. We use 32-bit integers to represent whether




| solver\time | # TO | Avg *solved* runtime TD | 3.I | 3.II | 3.III | Σ |
|---|---|---|---|---|---|---|
| Clasp 3.3.0 | 34 | - | - | - | - | 334.5 |
| DynASP2 | 68 | - | - | - | - | - |
| DynASP2.5 | 28 | 0.5 | 3.5 | 16.6 | 264.4 | 285.0 |
| ...depgraph | 24 | 10.9 | 3.4 | 15.3 | 251.3 | 280.9 |
| ...joinsize | 32 | 0.7 | 3.5 | 16.5 | 278.3 | 299.0 |

**Figure 4** Cactus plots showing best and average runtime among five TDs (left). Number of Timeouts (TO) and average runtime among solved instances (right).

an atom is set to true or a rule is satisfied in the respective bit positions according to the bag. A restriction to 32-bit integers seems reasonable as we assume, because of practical memory limitations, that our approach works well on TDs of width $\leq 20$. Since state-of-the-art computers handle constant-sized integers extremely efficiently, DynASP2.5 allows for efficient projections and joins of rows, as well as subset checks. In order to not recompute counter-witnesses (in Pass 3.II) for different witnesses, we use a three-valued notation of counter-witness sets consisting of atoms set to true (T), false (F), or false but true in the witness set (TW) to build the reduct. Note that only atoms occurring in negations or choice rules are among the (TW)-atoms, since only these atoms "affect" the corresponding reducts.

Minimum width is not the only optimization goal when computing TDs by means of heuristics. Instead, using customized TDs that not only optimize the width, but also some other, relevant feature, works seemingly well in practice [2]. While DynASP2.5 (M-DP$_{\mathbb{SINC}}$) does not take additional TD features into account, we also implemented a variant (*DynASP2.5 depgraph*), which prefers one out of ten TDs that intuitively speaking avoids to introduce head atoms of some rule $r$ in node $t$, without having encountered every body atom of $r$ below $t$, similar to atom dependencies in the program [12]. The variant *DynASP2.5 joinsize* minimizes bag sizes of child nodes of join nodes, c.f. [1].

## 4.3 Experimental Evaluation

We performed experiments to investigate the runtime behavior of DynASP2.5 and its variants, in order to evaluate whether our multi-pass approach can be beneficial and has practical advantages over the classical single pass approach (DynASP2). Further, we considered the dedicated ASP solver clasp 3.3.0. Clearly, we *cannot* hope to solve programs with graph representations of high treewidth. However, programs involving real-world graphs such as graph problems on transit graphs admit TDs of acceptable width to perform DP on TDs. To get a first intuition, we focused on the Steiner tree problem (ST) for our benchmarks.

We mainly inspected the CPU time using the average over five runs per instance (five fixed seeds allow for some variance in the heuristic TD computation). For each run, we limited the environment to 16 GB RAM and 1200 seconds CPU time. We used clasp with improvements for unsatisfiable cores [3] enabled and solution printing/recording disabled. We also benchmarked clasp with branch-and-bound, which, however, timed out on almost every instance. The left plot in Figure 4 shows the result of always selecting the best among five TDs, whereas the right plot shows the average running time. The table in Figure 4 reports average running times (TD computation and Passes 3.I, 3.II, 3.III) among the *solved* instances and the total number of timeouts (TO). We consider an instance to time out when all five TDs exceeded the limit. For the variants depgraph and joinsize, runtimes for

computing and selecting among ten TDs are included. Our empirical benchmark results confirm that DynASP2.5 exhibits competitive runtime behavior even for TDs of treewidth around 14. Compared to clasp, DynASP2.5 is capable of additionally delivering the number of optimal solutions. In particular, the depgraph variant shows promising runtimes.

## 5 Conclusion

In this paper, we presented a novel approach for ASP solving based on ideas from parameterized complexity. Our algorithms run in linear time assuming bounded treewidth of the input program. Our solver applies DP in three passes, thereby avoiding redundancies. Experimental results indicate that our ASP solver is competitive for certain classes of instances with small treewidth, where the latest version of the well-known solver clasp hardly keeps up. An interesting question for future research is whether a linear amount of passes (incremental DP) can improve the runtime behavior.

## References


1 Michael Abseher, Nysret Musliu, and Stefan Woltran. htd - A free, open-source framework for (customized) tree decompositions and beyond. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, volume 10335 of *Lecture Notes in Computer Science*, pages 376–386. Springer, 2017. doi:10.1007/978-3-319-59776-8_30.

2 Michael Abseher, Nysret Musliu, and Stefan Woltran. Improving the efficiency of dynamic programming on tree decompositions via machine learning. *J. Artif. Intell. Res.*, 58:829–858, 2017. doi:10.1613/jair.5312.

3 Mario Alviano and Carmine Dodaro. Anytime answer set optimization via unsatisfiable core shrinking. *TPLP*, 16(5-6):533–551, 2016. doi:10.1017/S147106841600020X.

4 B. Bliem, M. Moldovan, M. Morak, and S. Woltran. The impact of treewidth on ASP grounding and solving. In *IJCAI'17*, The AAAI Press, pages 852–858, 2017.

5 Bernhard Bliem, Günther Charwat, Markus Hecher, and Stefan Woltran. D-flat$^2$: Subset minimization in dynamic programming on tree decompositions made easy. *Fundam. Inform.*, 147(1):27–61, 2016. doi:10.3233/FI-2016-1397.

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 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The pace 2017 parameterized algorithms and computational experiments challenge: The second iteration. In *IPEC'17*, LIPIcs, pages 30:1—-30:13, 2017. doi:10.4230/LIPIcs.IPEC.2017.30.

8 Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.*, 15(3-4):289–323, 1995. doi:10.1007/BF01536399.

9 J. K. Fichte, M. Hecher, and S. Woltran. DynASP2.5: Dynamic Programming on Tree Decompositions in Action. *CoRR*, arXiv:1706.09370, 2017.

10 Johannes Klaus Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Answer set solving with bounded treewidth revisited. In Marcello Balduccini and Tomi Janhunen, editors, *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings*, volume 10377 of *Lecture Notes in Computer Science*, pages 132–145. Springer, 2017. doi:10.1007/978-3-319-61660-5_13.

**11** Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, 187:52–89, 2012. doi:10.1016/j.artint.2012.04.001.

**12** Georg Gottlob, Francesco Scarcello, and Martha Sideri. Fixed-parameter complexity in AI and nonmonotonic reasoning. *Artif. Intell.*, 138(1-2):55–86, 2002. doi:10.1016/S0004-3702(02)00182-0.

**13** T. Janhunen and I. Niemelä. The answer set programming paradigm. *AI Magazine*, 37(3):13–24, 2016. doi:10.1609/aimag.v37i3.2671.

**14** Joachim Kneis, Alexander Langer, and Peter Rossmanith. Courcelle's theorem - A game-theoretic approach. *Discrete Optimization*, 8(4):568–594, 2011. doi:10.1016/j.disopt.2011.06.001.

**15** Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artif. Intell.*, 138(1-2):181–234, 2002. doi:10.1016/S0004-3702(02)00187-X.

# Finding Connected Secluded Subgraphs*

**Petr A. Golovach[1], Pinar Heggernes[2], Paloma T. Lima[3], and Pedro Montealegre[4]**

1 **Department of Informatics, University of Bergen, Norway**
petr.golovach@ii.uib.no
2 **Department of Informatics, University of Bergen, Norway**
pinar.heggernes@ii.uib.no
3 **Department of Informatics, University of Bergen, Norway**
paloma.lima@ii.uib.no
4 **Facultad de Ingeniería y Ciencias, Universidad Adolfo Ibáñez, Santiago, Chile**
p.montealegre@uai.cl

## Abstract

Problems related to finding induced subgraphs satisfying given properties form one of the most studied areas within graph algorithms. Such problems have given rise to breakthrough results and led to development of new techniques both within the traditional P vs NP dichotomy and within parameterized complexity. The Π-Subgraph problem asks whether an input graph contains an induced subgraph on at least $k$ vertices satisfying graph property Π. For many applications, it is desirable that the found subgraph has as few connections to the rest of the graph as possible, which gives rise to the Secluded Π-Subgraph problem. Here, input $k$ is the size of the desired subgraph, and input $t$ is a limit on the number of neighbors this subgraph has in the rest of the graph. This problem has been studied from a parameterized perspective, and unfortunately it turns out to be W[1]-hard for many graph properties Π, even when parameterized by $k + t$. We show that the situation changes when we are looking for a connected induced subgraph satisfying Π. In particular, we show that the Connected Secluded Π-Subgraph problem is FPT when parameterized by just $t$ for many important graph properties Π.

**1998 ACM Subject Classification** G.2.2 Graph Theory, F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** Secluded subgraph, forbidden subgraphs, parameterized complexity

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.18

## 1 Introduction

Vertex deletion problems are central in parameterized algorithms and complexity, and they have contributed hugely to the development of new algorithmic methods. The Π-Deletion problem, with input a graph $G$ and an integer $\ell$, asks whether at most $\ell$ vertices can be deleted from $G$ so that the resulting graph satisfies graph property Π. Its dual, the Π-Subgraph problem, with input $G$ and $k$, asks whether $G$ contains an induced subgraph on at least $k$ vertices satisfying Π. The problems were introduced already in 1980 by Yannakakis and Lewis [11], who showed their NP-completeness for almost all interesting graph properties Π. During the last couple of decades, these problems have been studied extensively with respect to parameterized complexity and kernelization, which has resulted in numerous new techniques and methods in these fields [4, 5].

* This work is supported by Research Council of Norway via project "CLASSIS".




In many network problems, the size of the *boundary* between the subgraph that we are looking for and the rest of the graph makes a difference. A small boundary limits the exposure of the found subgraph, and notions like isolated cliques have been studied in this respect [7, 8, 10]. Several measures for the boundary have been proposed; in this work we use the open neighborhood of the returned induced subgraph. For a set of vertices $U$ of a graph $G$ and a positive integer $t$, we say that $U$ is *$t$-secluded* if $|N_G(U)| \leq t$. Analogously, an induced subgraph $H$ of $G$ is *$t$-secluded* if the vertex set of $H$ is $t$-secluded. For a given graph property $\Pi$, we get the following formal definition of the problem Secluded $\Pi$-Subgraph.

Secluded $\Pi$-Subgraph
*Input:* A graph $G$ and nonnegative integers $k$ and $t$.
*Task:* Decide whether $G$ contains a $t$-secluded induced subgraph $H$ on at least $k$ vertices, satisfying $\Pi$.

Lewis and Yannakakis [11] showed that $\Pi$-Subgraph is NP-complete for every hereditary nontrivial graph property $\Pi$. This immediately implies that Secluded $\Pi$-Subgraph is NP-complete for every such $\Pi$. As a consequence, the interest has shifted towards the parameterized complexity of the problem, which has been studied by van Bevern et al. [14] for several classes $\Pi$. Unfortunately, in most cases Secluded $\Pi$-Subgraph proves to be W[1]-hard, even when parameterized by $k+t$. In particular, it is W[1]-hard to decide whether a graph $G$ has a $t$-secluded independent set of size $k$ when the problem is parameterized by $k+t$ [14]. In this extended abstract, we show that the situation changes when the secluded subgraph we are looking for is required to be connected, in which case we are able to obtain positive results that apply to many properties $\Pi$. In fact, connectivity is central in recently studied variants of secluded subgraphs, like Secluded Path [2, 9] and Secluded Steiner Tree [6]. However, in these problems the boundary measure is the closed neighborhood of the desired path or the steiner tree, connecting a given set of vertices. The following formal definition describes the problem that we study in this extended abstract, Connected Secluded $\Pi$-Subgraph. For generality, we define a weighted problem.

Connected Secluded $\Pi$-Subgraph
*Input:* A graph $G$, a weight function $\omega\colon V(G) \to \mathbb{Z}_{>0}$, a nonnegative integer $t$ and a positive integer $w$.
*Task:* Decide whether $G$ contains a connected $t$-secluded induced subgraph $H$ with $\omega(V(H)) \geq w$, satisfying $\Pi$.

Observe that Connected Secluded $\Pi$-Subgraph remains NP-complete for all hereditary nontrivial graph properties $\Pi$, following the results of Yannakakis [15]. It can be also seen that Connected Secluded $\Pi$-Subgraph parameterized by $w$ is W[1]-hard even for unit weights, if it is W[1]-hard with parameter $k$ to decide whether $G$ has a connected induced subgraph on at least $k$ vertices, satisfying $\Pi$ (see, e.g., [5, 13]).

It is thus more interesting to consider parameterization by $t$. We consider Connected Secluded $\Pi$-Subgraph for all graph properties $\Pi$ that are characterized by finite sets $\mathcal{F}$ of forbidden induced subgraphs and refer to this variant of the problem as Connected Secluded $\mathcal{F}$-Free Subgraph. We show that the problem is fixed parameter tractable when parameterized by $t$ by proving the following theorem.

▶ **Theorem 1.** Connected Secluded $\mathcal{F}$-Free Subgraph *can be solved in time* $2^{2^{2^{\mathcal{O}(t \log t)}}} \cdot n^{\mathcal{O}(1)}$.

In this extended abstract, we only sketch the proofs and omit some of them due to space constraints.

## 2 Preliminaries

We consider only finite undirected graphs. We use $n$ to denote the number of vertices and $m$ the number of edges of the considered graphs unless it creates confusion. A graph $G$ is identified by its vertex set $V(G)$ and edge set $E(G)$. For $U \subseteq V(G)$, we write $G[U]$ to denote the subgraph of $G$ induced by $U$. We write $G - U$ to denote the graph $G[V(G) \setminus U]$; for a single-element $U = \{u\}$, we write $G-u$. A set of vertices $U$ is *connected* if $G[U]$ is a connected graph. For a vertex $v$, we denote by $N_G(v)$ the *(open) neighborhood* of $v$ in $G$, i.e., the set of vertices that are adjacent to $v$ in $G$. For a set $U \subseteq V(G)$, $N_G(U) = (\cup_{v \in U} N_G(v)) \setminus U$. We denote by $N_G[v] = N_G(v) \cup \{v\}$ the *closed neighborhood* of $v$; respectively, $N_G[U] = \cup_{v \in U} N_G[v])$. The *degree* of a vertex $v$ is $d_G(v) = |N_G(v)|$. A set of vertices $S \subset V(G)$ of a connected graph $G$ is a *separator* if $G - S$ is disconnected. A vertex $v$ is a *cut vertex* if $\{v\}$ is a separator.

A graph property is *hereditary* if it is preserved under vertex deletion, or equivalently, under taking induced subgraphs. A graph property is *trivial* if either the set of graphs satisfying it, or the set of graphs that do not satisfy it, is finite. Let $F$ be a graph. We say that a graph $G$ is *$F$-free* if $G$ has no induced subgraph isomorphic to $F$. For a set of graphs $\mathcal{F}$, a graph $G$ is *$\mathcal{F}$-free* if $G$ is $F$-free for every $F \in \mathcal{F}$. Let $\Pi$ be the property of being $\mathcal{F}$-free. Then, depending on whether $\mathcal{F}$ is a finite or an infinite set, we say that $\Pi$ is *characterized by a finite / infinite set of forbidden induced subgraphs.*

We use the *recursive understanding* technique introduced by Chitnis et al. [3] for graph problems to solve Connected Secluded $\Pi$-Subgraph when $\Pi$ is defined by forbidden induced subgraphs or $\Pi$ is the property to be a forest. This powerful technique is based on the following idea. Suppose that the input graph has a vertex separator of bounded size that separates the graph into two sufficiently big parts. Then we solve the problem recursively for one of the parts and replace this part by an equivalent graph such that the replacement keeps all essential (partial) solutions of the original part. By such a replacement we obtain a graph of smaller size. Otherwise, if there is no separator of bounded size separating graphs into two big parts, then either the graph has bounded size or it is highly connected, and we exploit these properties. We need the following notions and results from Chitnis et al. [3].

Let $G$ be a graph. A pair $(A, B)$, where $A, B \subseteq V(G)$ and $A \cup B = V(G)$, is a *separation of $G$ of order* $|A \cap B|$ if $G$ has no edge $uv$ with $u \in A \setminus B$ and $v \in B \setminus A$, i.e., $A \cap B$ is an $(A, B)$-separator. Let $q$ and $k$ be nonnegative integers. A graph $G$ is *(q,k)-unbreakable* if for every separation $(A, B)$ of $G$ of order at most $k$, $|A \setminus B| \leq q$ or $|B \setminus A| \leq q$. Combining Lemmas 19, 20 and 21 of [3], we obtain the following.

▶ **Lemma 2** ([3]). *Let $q$ and $k$ be nonnegative integers. There is an algorithm with running time $2^{\mathcal{O}(\min\{q,k\}\log(q+k))} \cdot n^3 \log n$ that, for a graph $G$, either finds a separation $(A, B)$ of order at most $k$ such that $|A \setminus B| > q$ and $|B \setminus A| > q$, or correctly reports that $G$ is $((2q + 1)q \cdot 2^k, k)$-unbreakable.*

We conclude this section by noting that the following variant of Connected Secluded $\Pi$-Subgraph is FPT when parameterized by $k + t$. We will rely on this result in the subsequent sections, however we believe that it is also of interest on its own.

Connected Secluded Colored $\Pi$-Subgraph of Exact Size

*Input:* A graph $G$, coloring $c\colon V(G) \to \mathbb{N}$, a weight function $\omega\colon V(G) \to \mathbb{Z}_{\geq 0}$ and nonnegative integers $k$, $t$ and $w$.

*Task:* Decide whether $G$ contains a connected $t$-secluded induced subgraph $H$ such that $(H, c')$, where $c'(v) = c|_{V(H)}(v)$, satisfies $\Pi$, $|V(H)| = k$ and $\omega(V(H)) \geq w$.

We say that a mapping $c\colon V(G) \to \mathbb{N}$ is a *coloring* of $G$; note that we do not demand a coloring to be proper. Analogously, we say that $\Pi$ is a *property of colored graphs* if $\Pi$ is a property on pairs $(G, c)$, where $G$ is a graph and $c$ is a coloring. Notice that if some vertices of the input graph have labels, then we can assign to each label (or a combination of labels if a vertex can have several labels) a specific color and assign some color to unlabeled vertices. Then we can redefine a considered graph property with the conditions imposed by labels as a property of colored graphs. Observe that we allow zero weights. Our next theorem presents two possible running times for the mentioned cases. The latter running times will be useful when $k \gg t$.

▶ **Theorem 3.** *If property* $\Pi$ *can be recognized in time* $f(n)$*, then* Connected Secluded Colored $\Pi$-Subgraph of Exact Size *can be solved both in time* $2^{k+t} \cdot f(k) \cdot n^{\mathcal{O}(1)}$*, and in time* $2^{\mathcal{O}(\min\{k,t\}\log(k+t))} \cdot f(k) \cdot n^{\mathcal{O}(1)}$.

In particular, the theorem implies that if $\Pi$ can be recognized in polynomial time, then Connected Secluded Colored $\Pi$-Subgraph of Exact Size can be solved both in time $2^{k+t} \cdot n^{\mathcal{O}(1)}$, and in time $2^{\mathcal{O}(\min\{k,t\}\log(k+t))} \cdot n^{\mathcal{O}(1)}$.

## 3 Solving Connected Secluded $\mathcal{F}$-Free Subgraph

In this section we prove Theorem 1. Throughout this section, we assume that we are given a fixed finite set $\mathcal{F}$ of graphs.

Recall that to apply the recursive understanding technique introduced by Chitnis et al. [3], we should be able to recurse when the input graph contains a separator of bounded size that separates the graph into two sufficiently big parts. To do this, we have to combine partial solutions in both parts. A danger in our case is that a partial solution in one part might contain a subgraph of a graph in $\mathcal{F}$. We have to avoid creating subgraphs belonging to $\mathcal{F}$ when we combine partial solutions. To achieve this goal, we need some definitions and auxiliary combinatorial results.

Let $p$ be a nonnegative integer. A pair $(G, x)$, where $G$ is a graph and $x = (x_1, \ldots, x_p)$ is a $p$-tuple of distinct vertices of $G$, is called a *$p$-boundaried graph* or simply a *boundaried graph*. Respectively, $x = (x_1, \ldots, x_p)$ is a *boundary*. Note that a boundary is an ordered set. Hence, two $p$-boundaried graphs that differ only by the order of the vertices in theirs boundaries are distinct. Observe also that a boundary could be empty. We say that $(G, x)$ is a *properly $p$-boundaried graph* if each component of $G$ has at least one vertex of the boundary. Slightly abusing notation, we may say that $G$ is a ($p$-) boundaried graph assuming that a boundary is given.

Two $p$-boundaried graphs $(G_1, x^{(1)})$ and $(G_2, x^{(2)})$, where $x^{(h)} = (x_1^{(h)}, \ldots, x_p^{(h)})$ for $h = 1, 2$, are *isomorphic* if there is an isomorphism of $G_1$ to $G_2$ that maps each $x_i^{(1)}$ to $x_i^{(2)}$ for $i \in \{1, \ldots, p\}$. We say that $(G_1, x^{(1)})$ and $(G_2, x^{(2)})$ are *boundary-compatible* if for any distinct $i, j \in \{1, \ldots, p\}$, $x_i^{(1)}x_j^{(1)} \in E(G_1)$ if and only if $x_i^{(2)}x_j^{(2)} \in E(G_2)$.

Let $(G_1, x^{(1)})$ and $(G_2, x^{(2)})$ be boundary-compatible $p$-boundaried graphs and let $x^{(h)} = (x_1^{(h)}, \ldots, x_p^{(h)})$ for $h = 1, 2$. We define the *boundary sum* $(G_1, x^{(1)}) \oplus_b (G_2, x^{(2)})$ (or simply $G_1 \oplus_b G_2$) as the (non-boundaried) graph obtained by taking vertex disjoint copies of $G_1$ and $G_2$ and identifying $x_i^{(1)}$ and $x_i^{(2)}$ for each $i \in \{1, \ldots, p\}$.

Let $G$ be a graph and let $y = (y_1, \ldots, y_p)$ be a $p$-tuple of vertices of $G$. For an $s$-boundaried graph $(H, x)$ with the boundary $x = (x_1, \ldots, x_s)$ and pairwise distinct $i_1, \ldots, i_s \in \{1, \ldots, p\}$, we say that $H$ is an *induced boundaried subgraph of $G$ with respect to* $(y_{i_1}, \ldots, y_{i_s})$ if $G$ contains an induced subgraph $H'$ isomorphic to $H$ such that the corresponding isomorphism of $H$ to $H'$ maps $x_j$ to $y_{i_j}$ for $j \in \{1, \ldots, s\}$ and $V(H') \cap \{y_1, \ldots, y_p\} = \{y_{i_1}, \ldots, y_{i_s}\}$.

We construct the set of boundaried graphs $\mathcal{F}_b$ as follows. For each $F \in \mathcal{F}$, each separation $(A, B)$ of $F$ and each $p = |A \cap B|$-tuple $x$ of the vertices of $(A \cap B)$, we include $(F[A], x)$ in $\mathcal{F}_b$ unless it already contains an isomorphic boundaried graph. We say that two properly $p$-boundaried graphs $(G_1, x^{(1)})$ and $(G_2, x^{(2)})$, where $x^{(h)} = (x_1^{(h)}, \ldots, x_p^{(h)})$ for $h = 1, 2$, are *equivalent (with respect to $\mathcal{F}_b$)* if

- **(i)** $(G_1, x^{(1)})$ and $(G_2, x^{(2)})$ are boundary-compatible,
- **(ii)** for any $i, j \in \{1, \ldots, p\}$, $x_i^{(1)}$ and $x_j^{(1)}$ are in the same component of $G_1$ if and only if $x_i^{(2)}$ and $x_j^{(2)}$ are in the same component of $G_2$,
- **(iii)** for any pairwise distinct $i_1, \ldots, i_s \in \{1, \ldots, p\}$, $G_1$ contains an $s$-boundaried induced subgraph $H \in \mathcal{F}_b$ with respect to the $s$-tuple $(x_{i_1}^{(1)}, \ldots, x_{i_s}^{(1)})$ if and only if $H$ is an induced subgraph of $G_2$ with respect to the $s$-tuple $(x_{i_1}^{(2)}, \ldots, x_{i_s}^{(2)})$.

It is straightforward to verify that the introduced relation is indeed an equivalence relation on the set of properly $p$-boundaried graphs. The following property of the equivalence with respect to $\mathcal{F}_b$ is crucial for our algorithm.

▶ **Lemma 4.** *Let $(G, x)$, $(H_1, y^{(1)})$ and $(H_2, y^{(2)})$ be boundary-compatible $p$-boundaried graphs, $x = (x_1, \ldots, x_p)$ and $y^{(h)} = (y_1^{(h)}, \ldots, y_p^{(h)})$ for $h = 1, 2$. If $(H_1, y^{(1)})$ and $(H_2, y^{(2)})$ are equivalent with respect to $\mathcal{F}_b$, then $(G, x) \oplus_b (H_1, y^{(1)})$ is $\mathcal{F}$-free if and only if $(G, x) \oplus_b (H_2, y^{(2)})$ is $\mathcal{F}$-free.*

It also should be noted that the equivalence of two properly $p$-boundaried graphs can be checked in polynomial time.

For each nonegative integer $p$, we consider a set $\mathcal{G}_p$ of properly $p$-boundaried graphs obtained by picking a graph with minimum number of vertices in each equivalence class. We show that the size of $\mathcal{G}_p$ and the size of each graph in the set is upper bounded by some functions of $p$, and this set can be constructed in time that depends only on $p$ assuming that $\mathcal{F}_b$ is fixed.

▶ **Lemma 5.** *For every positive integer $p$, $|\mathcal{G}_p| = 2^{\mathcal{O}(p^2)}$, and for every $H \in \mathcal{G}'_p$, $|V(H)| = p^{\mathcal{O}(1)}$, where the constants hidden in the $\mathcal{O}$-notations depend on $\mathcal{F}$ only. Moreover, for every $p$-boundaried graph $G$, the number of $p$-boundaried graphs in $\mathcal{G}_p$ that are compatible with $G$ is $2^{\mathcal{O}(p \log p)}$.*

Consider now the class $\mathcal{C}$ of $p$-boundaried graphs, such that a $p$-boundaried graph $(G, (x_1, \ldots, x_p\}) \in \mathcal{C}$ if and only if it holds that for every component $H$ of $G - \{x_1, \ldots, x_p\}$, $N_G(V(H)) = \{x_1, \ldots, x_p\}$. We consider our equivalence relation with respect to $\mathcal{F}_b$ on $\mathcal{C}$ and define $\mathcal{G}'_p$ as follows. In each equivalence class, we select a graph $(G, (x_1, \ldots, x_p\}) \in \mathcal{C}$ such that both the number of components of $G - \{x_1, \ldots, x_p\}$ is minimum and the number of vertices of $G$ is minimum subject to the first condition, and then include it in $\mathcal{G}'_p$. Similarly to Lemma 5 we show the following.

▶ **Lemma 6.** *For every positive integer $p$, $|\mathcal{G}'_p| = 2^{\mathcal{O}(p^2)}$, and for each $H \in \mathcal{G}_p$, $|V(H)| = p^{\mathcal{O}(1)}$, and the constants hidden in the $\mathcal{O}$-notations depend on $\mathcal{F}$ only. Moreover, for any $p$-boundaried graph $G$, the number of $p$-boundaried graphs in $\mathcal{G}'_p$ that are compatible with $G$ is $p^{\mathcal{O}(1)}$.*

Lemmas 5 and 6 immediately imply that $\mathcal{G}_p$ and $\mathcal{G}'_p$ can be constructed by brute force.

▶ **Lemma 7.** *The sets $\mathcal{G}_p$ and $\mathcal{G}'_p$ can be constructed in time $2^{p^{\mathcal{O}(1)}}$.*

To apply the recursive understanding technique, we also have to solve a special variant of Connected Secluded Π-Subgraph tailored for recursion. First, we define the following auxiliary problem for a positive integer $w$.

Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph

*Input:* A graph $G$, sets $I, O, B \subseteq V(G)$ such that $I \cap O = \emptyset$ and $I \cap B = \emptyset$, a weight function $\omega\colon V(G) \to \mathbb{Z}_{\geq 0}$ and a nonnegative integer $t$.

*Task:* Find a $t$-secluded $\mathcal{F}$-free induced connected subgraph $H$ of $G$ of maximum weight or weight at least $w$ such that $I \subseteq V(H)$, $O \subseteq V(G) \setminus V(H)$ and $N_G(V(H)) \subseteq B$ and output $\emptyset$ if such a subgraph does not exist.

Notice that Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph is an optimization problem and a *solution* is either an induced subgraph $H$ of maximum weight or of weight at least $w$, or $\emptyset$. Observe also that we allow zero weights for technical reasons.

We recurse if we can separate graphs by a separator of bounded size into two big parts and we use the vertices of the separator to combine partial solutions in both parts. This leads us to the following problem. Let $(G, I, O, B, \omega, t)$ be an instance of Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph and let $T \subseteq V(G)$ be a set of *border terminals*. We say that an instance $(G', I', O', B', \omega', t')$ is obtained by a *border complementation* if there is a partition $(X, Y, Z)$ of $T$ (some sets could be empty), where $X = \{x_1, \ldots, x_p\}$, such that $Y = \emptyset$ if $X = \emptyset$, $I \cap T \subseteq X$, $O \cap T \subseteq Y \cup Z$ and $Y \subseteq B$, and there is a $p$-boundaried graph $(H, y) \in \mathcal{G}_p$ such that $(H, y)$ and $(G, (x_1, \ldots, x_p))$ are boundary-compatible, and the following holds:

- **(i)** $G'$ is obtained from $(G, (x_1, \ldots, x_p)) \oplus_b (H, y)$ (we keep the notation $X = \{x_1, \ldots, x_p\}$ for the set of vertices obtained by the identification in the boundary sum) by adding edges joining every vertex of $V(H)$ with every vertex of $Y$,
- **(ii)** $I' = I \cup V(H)$,
- **(iii)** $O' = O \cup Y \cup Z$,
- **(iv)** $B' = B \setminus X$,
- **(v)** $\omega'(v) = \omega(v)$ for $v \in V(G)$ and $\omega'(v) = 0$ for $v \in V(H) \setminus X$,
- **(vi)** $t' \leq t$.

We also say that $(G', I', O', B', w', t')$ is a *border complementation of* $(G, I, O, B, w, t)$ *with respect to* $(X = \{x_1, \ldots, x_p\}, Y, Z, H)$. We say that $(X = \{x_1, \ldots, x_p\}, Y, Z, H)$ is *feasible* if it holds that $Y = \emptyset$ if $X = \emptyset$, $I \cap T \subseteq X$, $O \cap T \subseteq Y \cup Z$ and $Y \subseteq B$, and the $p$-boundaried graph $H \in \mathcal{G}_p$ and $(G, (x_1, \ldots, x_p))$ are boundary-compatible.

Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph

*Input:* A graph $G$, sets $I, O, B \subseteq V(G)$ such that $I \cap O = \emptyset$ and $I \cap B = \emptyset$, a weight function $\omega\colon V(G) \to \mathbb{Z}_{\geq 0}$, a nonnegative integer $t$, and a set $T \subseteq V(G)$ of border terminals of size at most $2t$.

*Task:* Output a solution for each instance $(G', I', O', B', w', t')$ of Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph that can be obtained from $(G, I, O, B, w, t)$ by a border complementation distinct from the border complementation with respect to $(\emptyset, \emptyset, T, \emptyset)$, and for the border complementation with respect to $(\emptyset, \emptyset, T, \emptyset)$ output a nonempty solution if it has weight at least $w$ and output $\emptyset$ otherwise.

Two instances $(G_1, I_1, O_1, B_1, \omega_1, t, T)$ and $(G_2, I_2, O_2, B_2, \omega_2, t, T)$ of Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph (note that $t$ and $T$ are the same) are said to be *equivalent* if

- **(i)** $T \cap I_1 = T \cap I_2$, $T \cap O_1 = T \cap O_2$ and $T \cap B_1 = T \cap B_2$,
- **(ii)** for the border complementations $(G_1', I_1', O_1', B_1', \omega_1', t')$ and $(G_2', I_2', O_2', B_2', \omega_2', t')$ of the instances $(G_1, I_1, O_1, B_1, \omega_1, t')$ and $(G_2, I_2, O_2, B_2, \omega_2, t')$ respectively of Maximum or

$w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph with respect to every feasible $(X = \{x_1, \ldots, x_p\}, Y, Z, H)$ and $t' \leq t$, it holds that

**(a)** if $(G'_1, I'_1, O'_1, B'_1, \omega'_1, t')$ has a nonempty solution $R_1$, then $(G'_2, I'_2, O'_2, B'_2, \omega'_2, t')$ has a nonempty solution $R_2$ with $w'_2(V(R_2)) \geq \min\{\omega'_1(V(R_1)), w\}$ and, vice versa,

**(b)** if $(G'_2, I'_2, O'_2, B'_2, \omega'_2, t')$ has a nonempty solution $R_2$, then $(G'_1, I'_1, O'_1, B'_1, \omega'_1, t')$ has a nonempty solution $R_1$ with $\omega'_1(V(R_1)) \geq \min\{\omega'_2(V(R_2)), w\}$.

Strictly speaking, if $(G_1, I_1, O_1, B_1, \omega_1, t, T)$ and $(G_2, I_2, O_2, B_2, \omega_2, t, T)$ are equivalent, then a solution of the first problem is not necessarily a solution of the second. Nevertheless, Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph is an auxiliary problem and in the end we use it to solve an instance $(G, \omega, t, w)$ of Connected Secluded $\mathcal{F}$-Free Subgraph by calling the algorithm for Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph for $(G, \emptyset, \emptyset, V(G), \omega, t, \emptyset)$. Clearly, $(G, \omega, t, w)$ is a yes-instance if and only if a solution for the corresponding instance of Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph contains a connected subgraph $R$ with $\omega(V(R)) \geq w$. It allows us to not distinguish equivalent instances of Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph and their solutions.

## 3.1 High connectivity phase

In this section we solve Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph for $(q,t)$-unbreakable graphs. The following lemma shows that we can separately lists all graphs $R$ in a solution of Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph with $|V(R) \cap V(G)| \leq q$ and all graphs $R$ with $|V(G) \setminus V(R)| \leq q + t$.

▶ **Lemma 8.** *Let $(G, I, O, B, \omega, t, T)$ be an instance of* Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph *where $G$ is a $(q,t)$-unbreakable graph for a positive integer $q$. Then for each nonempty graph $R$ in a solution of* Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph*, either $|V(R) \cap V(G)| \leq q$ or $|V(G) \setminus V(R)| \leq q + t$.*

To list $R$ with $|V(R) \cap V(G)| \leq q$, we use Theorem 3. To list $R$ with $|V(G) \setminus V(R)| \leq q+t$, we use *important separators* defined by Marx in [12]. The main observation in this second case is that if $|V(G) \setminus V(R)| \leq q + t$, then there is a hitting set $S$ of size at most $q + t$ for all copies of graphs of $\mathcal{F}$ that lies outside $R$ in the corresponding graph. Moreover, hitting sets of size at most $q+t$ can be enumerated in FPT time. Then we can use important separators between the closed neighborhood of $I$ and $S \cup O$ to find $R$. It gives us the following crucial lemma.

▶ **Lemma 9.** Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph *for $(q,t)$-unbreakable graphs can be solved in time $2^{(q+t\log(q+t)))} \cdot n^{\mathcal{O}(1)}$ if the sets $\mathcal{G}_p$ for $p \leq 2t$ are given.*

## 3.2 The FPT algorithm for Connected Secluded $\mathcal{F}$-Free Subgraph

In this section we construct an FPT algorithm for Connected Secluded $\mathcal{F}$-Free Subgraph parameterized by $t$. We do this by solving Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph in FPT-time for general case.

▶ **Lemma 10.** Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph *can be solved in time $2^{2^{2^{\mathcal{O}(t \log t)}}} \cdot n^{\mathcal{O}(1)}$.*

**Sketch of the Proof.** Given $\mathcal{F}$, we construct the set $\mathcal{F}_b$. Then we use Lemma 7 to construct the sets $\mathcal{G}_p$ for $p \in \{0,\ldots,t\}$ in time $2^{t^{\mathcal{O}(1)}}$.

By Lemma 5, there is a constant $c$ that depends only on $\mathcal{F}$ such that for every nonnegative $p$ and for any $p$-boundaried graph $G$, there are at most $2^{cp\log p}$ $p$-boundaried graphs in $\mathcal{G}_p$ that are compatible with $G$ and there are at most $p^c$ $p$-boundaried graphs in $\mathcal{G}'_p$ that are compatible with $G$. We define

$$q = 2^{((t+1)t3^{2t}2^{c2t\log(2t)}+2t)} \cdot 2((t+1)t3^{2t}2^{c2t\log(2t)}+2t)^c t^c + (t+1)t3^{2t}2^{c2t\log(2t)} + 2t. \quad (1)$$

The choice of $q$ will become clear later in the proof. Notice that $q = 2^{2^{\mathcal{O}(t\log t)}}$.

Consider an instance $(G, I, O, B, \omega, t, T)$ of Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph.

We use the algorithm from Lemma 2 for $G$. This algorithm in time $2^{2^{\mathcal{O}(t\log t)}} \cdot n^{\mathcal{O}(1)}$ either finds a separation $(U, W)$ of $G$ of order at most $t$ such that $|U \setminus W| > q$ and $|W \setminus U| > q$ or correctly reports that $G$ is $((2q+1)q \cdot 2^t, t)$-unbreakable. In the latter case we solve the problem using Lemma 9 in time $2^{2^{2^{\mathcal{O}(t\log t)}}} \cdot n^{\mathcal{O}(1)}$. Assume from now that there is a separation $(U, W)$ of order at most $t$ such that $|U \setminus W| > q$ and $|W \setminus U| > q$.

Recall that $|T| \leq 2t$. Then $|T \cap (U \setminus W)| \leq t$ or $|T \cap (W \setminus U)| \leq t$. Assume without loss of generality that $|T \cap (W \setminus U)| \leq t$. Let $\tilde{G} = G[W]$, $\tilde{I} = I \cap W$, $\tilde{O} = O \cap W$, $\tilde{\omega}$ is the restriction of $\omega$ to $W$, and define $\tilde{T} = (T \cap W) \cup (U \cap W)$. Since $|U \cap W| \leq t$, $|\tilde{T}| \leq 2t$.

If $|W| \leq (2q+1)q \cdot 2^t$, then we solve Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph for the instance $(\tilde{G}, \tilde{I}, \tilde{O}, \tilde{B}, \tilde{\omega}, t, \tilde{T})$ by brute force in time $2^{2^{2^{\mathcal{O}(t\log t)}}}$ trying all possible subset of $W$ and at most $t+1$ values of $0 \leq t' \leq t$. Otherwise, we solve $(\tilde{G}, \tilde{I}, \tilde{O}, \tilde{B}, \tilde{\omega}, t, \tilde{T})$ recursively. Let $\mathcal{R}$ be the set of nonempty induced subgraphs $R$ that are included in the obtained solution for $(\tilde{G}, \tilde{I}, \tilde{O}, \tilde{B}, \tilde{\omega}, t, \tilde{T})$.

For $R \in \mathcal{R}$, define $S_R$ to be the set of vertices of $W \setminus V(R)$ that are adjacent to the vertices of $R$ in the graph obtained by the border complementation for which $R$ is a solution of the corresponding instance of Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph. Note that $|S_R| \leq t$. If $\mathcal{R} \neq \emptyset$, then let $S = \tilde{T} \cup_{R\in\mathcal{R}} S_R$, and $S = \tilde{T}$ if $\mathcal{R} = \emptyset$. Since Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph is solved for at most $t+1$ of values of $t' \leq t$, at most $3^{2t}$ three-partitions $(X, Y, Z)$ of $\tilde{T}$ and at most $2^{c2t\log(2t)}$ choices of a $p$-boundaried graph $H \in \mathcal{F}_b$ for $p = |X|$, we have that $|\mathcal{R}| \leq (t+1)3^{2t}2^{c2t\log(2t)}$. Taking into account that $|T'| \leq 2t$,

$$|S| \leq (t+1)t3^{2t}2^{c2t\log(2t)} + 2t. \quad (2)$$

Let $\hat{B} = (B \cap U) \cup (B \cap S)$. We claim that the instances $(G, I, O, B, \omega, t, T)$ and $(G, I, O, \hat{B}, \omega, t, T)$ of Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph are equivalent.

Since, $(G, I, O, B, \omega, t, T)$ and $(G, I, O, \hat{B}, \omega, t, T)$ of Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph are equivalent, we can consider $(G, I, O, \hat{B}, \omega, t, T)$. Now we apply some reduction rules that produce equivalent instances of Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph or report that we have no solution. The ultimate aim of these rules is to reduce the size of $G$.

Let $Q$ be a component of $G[W] - S$. Notice that for any nonempty graph $R$ in a solution of $(G, I, O, \hat{B}, w, t, T)$, either $V(Q) \subseteq V(R)$ or $V(Q) \cap V(R) = \emptyset$, because $N_{G[W]}(V(R)) \subseteq S$. Moreover, if $V(Q) \cap V(R) = \emptyset$, then $N_{G[W]}[V(Q)] \cap V(R) = \emptyset$. Notice also that if $v \in N_{G[W]}(V(Q))$ is a vertex of $R$, then $V(Q) \subseteq V(R)$. These observation are crucial for the following reduction rules.

▶ **Reduction Rule 3.1.** For a component $Q$ of $G[W]-S$ do the following in the given order:
- if $N_{G[W]}[V(Q)] \cap I \neq \emptyset$ and $V(Q) \cap O \neq \emptyset$, then return $\emptyset$ and stop,
- if $N_{G[W]}[V(Q)] \cap I \neq \emptyset$, then set $I = I \cup V(Q)$,
- if $V(Q) \cap O \neq \emptyset$, then set $O = O \cup N_{G[W]}[V(Q)]$.

The rule is applied to each component $Q$ exactly once. Notice that after application of the rule, for every component $Q$ of $G[W]-S$, we have that either $V(Q) \subseteq I$ or $V(Q) \subseteq O$ or $V(Q) \cap (I \cup O \cup \hat{B}) = \emptyset$.

Suppose that $Q_1$ and $Q_2$ are components of $G[W]-S$ such that $N_{G[W]}(V(Q_1)) = N_{G[W]}(V(Q_2))$ and $|N_{G[W]}(V(Q_1))| = |N_{G[W]}(V(Q_2))| > t$. Then if $V(Q_1) \subseteq V(R)$ for a nonempty graph $R$ in a solution of $(G, I, O, \hat{B}, \omega, t, T)$, then at least one vertex of $N_{G[W]}(V(Q_1))$ is in $R$ as $R$ have at most $t$ neighbors outside $R$. This gives the next rule.

▶ **Reduction Rule 3.2.** For components $Q_1$ and $Q_2$ of $G[W]-S$ such that $N_{G[W]}(V(Q_1)) = N_{G[W]}(V(Q_2))$ and $|N_{G[W]}(V(Q_1))| = |N_{G[W]}(V(Q_2))| > t$ do the following in the given order:
- if $(V(Q_1) \cup V(Q_2)) \cap I \neq \emptyset$ and $(V(Q_1) \cup V(Q_2)) \cap O \neq \emptyset$, then return $\emptyset$ and stop,
- if $(V(Q_1) \cup V(Q_2)) \cap I \neq \emptyset$, then set $I = I \cup (V(Q_1) \cup V(Q_2))$,
- if $(V(Q_1) \cup V(Q_2)) \cap O \neq \emptyset$, then set $O = O \cup N_{G[W]}[V(Q_1) \cup V(Q_2)]$.

We apply the rule for all pairs of components $Q_1$ and $Q_2$ with $N_{G[W]}(V(Q_1)) = N_{G[W]}(V(Q_2))$ and $|N_{G[W]}(V(Q_1))| = |N_{G[W]}(V(Q_2))| > t$, and for each pair the rule is applied once.

If $V(Q) \subseteq O$ for a component $Q$ of $G[W]-S$, then $N_{G[W]}(V(Q)) \subseteq O$. It immediately implies that the vertices of $Q$ are irrelevant and can be removed.

▶ **Reduction Rule 3.3.** If there is a component $Q$ of $G[W]-S$ such that $N_{G[W]}(V(Q)) \subseteq O$, then set $G = G - V(Q)$, $W = W \setminus V(Q)$ and $O = O \setminus V(Q)$.

Notice that for each component $Q$, we have that either $V(Q) \subseteq I$ or $V(Q) \subseteq W \setminus (I \cup O \cup \hat{B})$.

To define the remaining rules, we construct the sets $\mathcal{G}'_p$ for $p \in \{0, \ldots, |S|\}$ in time $2^{2^{\mathcal{O}(t \log t)}}$ using Lemma 7.

Let $Q$ be a component of $G[W]-S$ and let $N_{G[W]}(V(Q)) = \{x_1, \ldots, x_p\}$. Let $G'$ be the graph obtained from $G$ by the deletion of the vertices of $V(Q)$ and let $x = (x_1, \ldots, x_p)$. Let $(H, y)$ be a connected $p$-boundaried graph of the same weight as $G[N_{G[W]}[V(Q)]]$. Then by Lemma 4, we have that the instance of Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph obtained from $(G, I, O, \hat{B}, \omega, t, T)$ by the replacement of $G$ by $(G', x) \oplus_b (H, y)$ is equivalent to $(G, I, O, \hat{B}, \omega, t, T)$. We use it in the remaining reduction rules.

Suppose again that $Q_1$ and $Q_2$ are components of $G[W]-S$ such that $N_{G[W]}(V(Q_1)) = N_{G[W]}(V(Q_2))$ and $|N_{G[W]}(V(Q_1))| = |N_{G[W]}(V(Q_2))| > t$. Then, as we already noticed, if $V(Q_1) \cup V(Q_2) \subseteq V(R)$ for a nonempty graph $R$ in a solution of $(G, I, O, \hat{B}, \omega, t, T)$, then at least one vertex of $N_{G[W]}(V(Q_1))$ is in $R$. It means that if we are constructing a solution $R$, then the restriction of the size of the neighborhood of $R$ ensures the connectivity between $Q_1$ and $Q_2$ if we decide to include these components in $R$. Together with Lemma 4 this shows that the following rule is safe.

▶ **Reduction Rule 3.4.** Let $L = \{x_1, \ldots, x_p\} \subseteq S$, $p > t$, and let $x = (x_1, \ldots, x_p)$. Let also $Q_1, \ldots, Q_r, r \geq 1$, be the components of $G[W]-S$ with $N_{G[W]}(V(Q_i)) = L$ for all $i \in \{1, \ldots, r\}$. Let $Q = G[\cup_{i=1}^{r} N_{G[W]}[V(Q_i)]]$ and $w' = \sum_{i=1}^{r} \omega(V(Q_i))$. Find a $p$-boundaried

graph $(H, y) \in \mathcal{G}'_p$ that is equivalent to $(Q, x)$ with respect to $\mathcal{F}_b$ and denote by $A$ the set of nonboundary vertices of $H$. Then do the following.

- Delete the vertices of $V(Q_1), \ldots, V(Q_r)$ from $G$ and denote the obtained graph $G'$.
- Set $G = (G', x) \oplus_b (H, y)$ and $W = (W \setminus \cup_{i=1}^{r} V(Q_i)) \cup A$.
- Select arbitrarily $u \in A$ and modify $\omega$ as follows:
  - keep the weight same for every $v \in V(G')$ including the boundary vertices $x_1, \ldots, x_p$,
  - set $\omega(v) = 0$ for $v \in A \setminus \{u\}$,
  - set $\omega(u) = w'$.
- If $V(Q_1) \subseteq I$, then set $I = I \setminus (\cup_{i=1}^{r} V(Q_i)) \cup A$.

The rule is applied exactly once for each inclusion maximal sets of components $\{Q_1, \ldots, Q_r\}$ having the same neighborhood of size at least $t + 1$.

We cannot apply this trick if we have several components $Q_1, \ldots, Q_r$ of $G[W] - S$ with the same neighborhood $N_{G[W]}(V(Q_i))$ if $|N_{G[W]}(V(Q_i))| \leq t$. Now it can happen that there are $i, j \in \{1, \ldots, r\}$ such that $V(Q_i) \subseteq V(R)$ and $N_{G[W]}[V(Q_j)] \cap V(R) = \emptyset$ for $R$ in a solution of $(G, I, O, \hat{B}, \omega, t, T)$. But if $N_{G[W]}[V(Q_j)] \cap V(R) = \emptyset$ , then by the connectivity of $R$ and the fact that $G[W] - S$ does not contain border terminals, we have that $R = Q_i$. Notice that $I = \emptyset$ in this case and, in particular, it means that $R$ is a solution for an instance of Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph obtained by the border complementation with respect to $(\emptyset, \emptyset, T, \emptyset)$. Recall that we output $R$ in this case only if its weight is at least $w$. Still, we can modify Reduction Rule 3.4 for the case when there are components $Q$ of $G[W] - S$ such that $V(Q) \subseteq I$. Notice that if there are components $Q_0, \ldots, Q_r$ of $G[W] - S$ with the same neighborhood and $V(Q_0) \subseteq I$, then for any nonempty $R$ in a solution of $(G, I, O, \hat{B}, \omega, t, T)$, either $R = Q_0$ or $\cup_{i=0}^{r} V(Q_i) \subseteq V(R)$. Applying Lemma 4, we obtain that the following rule is safe.

▶ **Reduction Rule 3.5.** Let $L = \{x_1, \ldots, x_p\} \subseteq S$, $p \leq t$, and let $x = (x_1, \ldots, x_p)$. Let also $Q_0, \ldots, Q_r$,$r \geq 0$, be the components of $G[W] - S$ with $N_{G[W]}(V(Q_i)) = L$ for all $i \in \{0, \ldots, r\}$ such that $V(Q_0) \subseteq I$. Let $Q = G[\cup_{i=1}^{r} N_{G[W]}[V(Q_i)]]$ and $w' = \sum_{i=1}^{r} \omega(V(Q_i))$. Find a $p$-boundaried graph $(H_0, y) \in \mathcal{G}'_p$ that is equivalent to $(Q_0, x)$ with respect to $\mathcal{F}_b$ and denote by $A_0$ the set of nonboundary vertices of $H_0$, and find a $p$-boundaried graph $(H, y) \in \mathcal{G}'_p$ that is equivalent to $(Q, x)$ with respect to $\mathcal{F}_b$ and denote by $A$ the set of nonboundary vertices of $H$. Then do the following.

- Delete the vertices of $V(Q_0), \ldots, V(Q_r)$ from $G$ and denote the obtained graph $G'$.
- Set $G = (((G', x) \oplus_b (H_0, y)), y) \oplus_b (H, y)$ and $W = (W \setminus \cup_{i=0}^{r} V(Q_i)) \cup A_0 \cup A$.
- Select arbitrarily $u \in A_0$ and $v \in A$ and modify $\omega$ as follows:
  - keep the weight same for every $z \in V(G')$ including the boundary vertices $x_1, \ldots, x_p$,
  - set $\omega(z) = 0$ for $z \in (A_0 \setminus \{u\}) \cup (A \setminus \{v\})$,
  - set $\omega(u) = \omega(V(Q_0))$ and $\omega(v) = w'$.
- If $V(Q_i) \subseteq I$ for some $i \in \{1, \ldots, r\}$, then set $I = I \setminus (\cup_{i=1}^{r} V(Q_i)) \cup A$.

Assume now that we have an inclusion maximal set of components $\{Q_1, \ldots ,Q_r\}$ of $G[W] - S$ with the same neighborhoods $N_{G[W]} = \{x_1, \ldots, x_p\}$ such that the $p$-boundaried graphs $(G[N_{G[W]}[V(Q_i)]], (x_1, \ldots, x_p))$ and $(G[N_{G[W]}[V(Q_j)]], (x_1, \ldots, x_p))$ are equivalent with respect to $\mathcal{F}_b$ for each $i, j \in \{1, \ldots, p\}$. Suppose also that $V(Q_i) \cap I = \emptyset$ for $i \in \{1, \ldots, r\}$. Let $\omega(V(Q_1)) \geq \omega(V(Q_i))$ for every $i \in \{1, \ldots, r\}$. Recall that if $R$ is a nonempty graph in a solution, then either $R = Q_i$ for some $i \in \{1, \ldots, r\}$ or $\cup_{i=1}^{r} V(Q_i) \subseteq V(R)$. Recall also that $R$ is a solution for the instance of Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph obtained by a border complementation with respect to $(\emptyset, \emptyset, T, \emptyset)$ and we output it only if $\omega(V(R)) \geq w$. Since all $(G[N_{G[W]}[V(Q_i)]], (x_1, \ldots, x_p))$

are equivalent, we can assume that if $R = Q_i$, then $i = 1$, because $Q_1$ has maximum weight. Then by Lemma 4, our final reduction rule is safe.

▶ **Reduction Rule 3.6.** Let $L = \{x_1, \ldots, x_p\} \subseteq S$, $p \leq t$, and let $x = (x_1, \ldots, x_p)$. Let also $Q_0, \ldots, Q_r, r \geq 0$, be the components of $G[W] - S$ with $N_{G[W]}(V(Q_i)) = L$ for all $i \in \{0, \ldots, r\}$ such that $\omega(V(Q_0)) \geq \omega(V(Q_i))$ for every $i \in \{1, \ldots, r\}$ and the $p$-boundaried graphs $(G[N_{G[W]}[V(Q_i)]], (x_1, \ldots, x_p))$ are pairwise equivalent with respect to $\mathcal{F}_b$ for $i \in \{0, \ldots, r\}$. Let $Q = G[\cup_{i=1}^{r} N_{G[W]}[V(Q_i)]]$ and $w' = \min\{w - 1, \sum_{i=1}^{r} \omega(V(Q_i))\}$. Find a $p$-boundaried graph $(H_0, y) \in \mathcal{G}'_p$ that is equivalent to $(Q_0, x)$ with respect to $\mathcal{F}_b$ and denote by $A_0$ the set of nonboundary vertices of $H_0$, and find a $p$-boundaried graph $(H, y) \in \mathcal{G}'_p$ that is equivalent to $(Q, x)$ with respect to $\mathcal{F}_b$ and denote by $A$ the set of nonboundary vertices of $H$. Then do the following.

- Delete the vertices of $V(Q_0), \ldots, V(Q_r)$ from $G$ and denote the obtained graph $G'$.
- Set $G = (((G', x) \oplus_b (H_0, y)), y) \oplus_b (H, y)$ and $W = (W \setminus \cup_{i=0}^{r} V(Q_i)) \cup A_0 \cup A$.
- Select arbitrarily $u \in A_0$ and $v \in A$ and modify $\omega$ as follows:
  - keep the weight same for every $z \in V(G')$ including the boundary vertices $x_1, \ldots, x_p$,
  - set $\omega(z) = 0$ for $z \in (A_0 \setminus \{u\}) \cup (A \setminus \{v\})$,
  - set $\omega(u) = \omega(V(Q_0))$ and $\omega(v) = w'$.

The Reduction Rule 3.6 is applied for each inclusion maximal sets of components $\{Q_0, \ldots, Q_r\}$ satisfying the conditions of the rule such that Reduction Rule 3.5 was not applied to these components before.

Denote by $(G^*, I^*, O^*, B^*, \omega^*, t, T)$ the instance of Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph obtained from $(G, I, O, \hat{B}, \omega, t, T)$ by Reduction Rules 3.1-3.6. Notice that all modifications were made for $G[W]$. Denote by $W^*$ the set of vertices of the graph obtained from the initial $G[W]$ by the rules. Observe that there are at most $2^{|S|}$ subsets $L$ of $S$ such that there is a component $Q$ of $G[W] - S$ with $N_{G[W]}(V(Q)) = L$. If $|L| > t$, then all $Q$ with $N_{G[W]}(V(Q)) = L$ are replaced by one graph by Reduction Rule 3.4 and the number of vertices of this graph is at most $|L|^c$ by Lemma 5 and the definition of $c$. If $|L| \leq t$, then we either apply Reduction Rule 3.5 for all $Q$ with $N_{G[W]}(V(Q)) = L$ and replace these components by two graph with at most $|L|^c$ vertices or we apply Reduction Rule 3.6. For the latter case, observe that there are at most $t^c$ partitions of the components $Q$ with $N_{G[W]}(V(Q)) = L$ into equivalence classes with respect to $\mathcal{F}_b$ by Lemma 5. Then we replace each class by two graphs with at most $|L|^c$ vertices. Taking into account the vertices of $S$, we obtain the following upper bound for the size of $W^*$: $|W^*| \leq 2^{|S|}2|S|^c t^c + |S|$. By (1) and (2), $|W^*| \leq q$. Recall that $|W \setminus U| > q$. Therefore, $|V(G^*)| < |V(G)|$. We use it and solve Bordered Maximum or $w$-Weighted Connected Secluded $\mathcal{F}$-Free Subgraph for $(G^*, I^*, O^*, B^*, \omega^*, t, T)$ recursively.

Following the general scheme from [3], we show that the total running time is $2^{2^{2^{\mathcal{O}(t \log t)}}} \cdot n^{\mathcal{O}(1)}$. ◀

It remains to observe that Lemma 10 immediately implies Theorem 1.

## 4 Concluding remarks

In addition to our general result from the previous section, we are also able to show that Connected Secluded Π-Subgraph is FPT parameterized by $t$, when Π is defined by an *infinite* set of forbidden induced subgraphs, namely, the set of all cycles. In other words, a graph has the property Π considered if it is a forest. Using the recursive understanding technique, we proved that the problem can be solved in time $2^{2^{\mathcal{O}(t \log t)}} \cdot n^{\mathcal{O}(1)}$. We believe

that the same approach can be used for other graph properties $\Pi$ as well. Nevertheless, the drawback of applying the recursive understanding technique is that we get double or even triple-exponential dependence on the parameter in our FPT algorithms. It is natural to ask whether we can do better for some properties $\Pi$. This can in fact be done when $\Pi$ is the property of being a complete graph, a star, a path or a $d$-regular graph.

Finally, we conclude by briefly touching upon the kernelization question. For Connected Secluded $\Pi$-Subgraph, we hardly can hope to obtain polynomial kernels as it could be easily proved by applying the results of Bodlaender et al. [1] that, unless NP $\subseteq$ coNP/poly, Connected Secluded $\Pi$-Subgraph has no polynomial kernel when parameterized by $t$ if Connected Secluded $\Pi$-Subgraph is NP-complete. Nevertheless, Connected Secluded $\Pi$-Subgraph can have a polynomial *Turing kernel*. In particular, we are able to show that Connected Secluded $\Pi$-Subgraph has a polynomial Turing kernel if $\Pi$ is the property of being a star.

## References


1 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

2 Shiri Chechik, Matthew P. Johnson, Merav Parter, and David Peleg. Secluded connectivity problems. In *ESA 2013*, volume 8125 of *LNCS*, pages 301–312. Springer, 2013.

3 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM J. Comput.*, 45(4):1171–1229, 2016. doi:10.1137/15M1032077.

4 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.

5 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.

6 Fedor V. Fomin, Petr A. Golovach, Nikolay Karpov, and Alexander S. Kulikov. Parameterized complexity of secluded connectivity problems. *Theory Comput. Syst.*, 61(3):795–819, 2017. doi:10.1007/s00224-016-9717-x.

7 Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Isolation concepts for clique enumeration: Comparison and computational experiments. *Theor. Comput. Sci.*, 410(52):5384–5397, 2009. doi:10.1016/j.tcs.2009.05.008.

8 Hiro Ito and Kazuo Iwama. Enumeration of isolated cliques and pseudo-cliques. *ACM Trans. Algorithms*, 5(4):40:1–40:21, 2009. doi:10.1145/1597036.1597044.

9 Matthew P. Johnson, Ou Liu, and George Rabanca. Secluded path via shortest path. In *SIROCCO 2014*, volume 8576 of *LNCS*, pages 108–120. Springer, 2014.

10 Christian Komusiewicz, Falk Hüffner, Hannes Moser, and Rolf Niedermeier. Isolation concepts for efficiently enumerating dense subgraphs. *Theor. Comput. Sci.*, 410(38-40):3640–3654, 2009. doi:10.1016/j.tcs.2009.04.021.

11 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

12 Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006. doi:10.1016/j.tcs.2005.10.007.

13 Christos H. Papadimitriou and Mihalis Yannakakis. On limited nondeterminism and the complexity of the V-C dimension. *J. Comput. Syst. Sci.*, 53(2):161–170, 1996. doi:10.1006/jcss.1996.0058.

**14** René van Bevern, Till Fluschnik, George B. Mertzios, Hendrik Molter, Manuel Sorge, and Ondrej Suchý. Finding secluded places of special interest in graphs. 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 5:1–5:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.5.

**15** Mihalis Yannakakis. The effect of a connectivity requirement on the complexity of maximum subgraph problems. *J. ACM*, 26(4):618–630, 1979. doi:10.1145/322154.322157.

# FO Model Checking of Geometric Graphs*

**Petr Hliněný †1, Filip Pokrývka‡2, and Bodhayan Roy3**

1 **Faculty of Informatics, Masaryk University Brno, Czech Republic**
`hlineny@fi.muni.cz`
2 **Faculty of Informatics, Masaryk University Brno, Czech Republic**
`xpokryvk@fi.muni.cz`
3 **Faculty of Informatics, Masaryk University Brno, Czech Republic**
`b.roy@fi.muni.cz`

## Abstract

Over the past two decades the main focus of research into first-order (FO) model checking algorithms has been on sparse relational structures – culminating in the FPT algorithm by Grohe, Kreutzer and Siebertz for FO model checking of nowhere dense classes of graphs. On contrary to that, except the case of locally bounded clique-width only little is currently known about FO model checking of dense classes of graphs or other structures. We study the FO model checking problem for dense graph classes definable by geometric means (intersection and visibility graphs). We obtain new nontrivial FPT results, e.g., for restricted subclasses of *circular-arc, circle, box, disk, and polygon-visibility graphs*. These results use the FPT algorithm by Gajarský et al. for FO model checking of posets of bounded width. We also complement the tractability results by related hardness reductions.

**1998 ACM Subject Classification** F.1.3 Complexity Measures and Classes, F.4.1 Mathematical Logic – Model theory, G.2.2 Graph Theory – Graph algorithms

**Keywords and phrases** first-order logic, model checking, fixed-parameter tractability, intersection graphs, visibility graphs

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.19

## 1 Introduction

Algorithmic meta-theorems are results stating that all problems expressible in a certain language are efficiently solvable on certain classes of structures, e.g. of finite graphs. Note that the model checking problem for *first-order logic* – given a graph $G$ and an FO formula $\phi$, we want to decide whether $G$ satisfies $\phi$ (written as $G \models \phi$) – is trivially solvable in time $|V(G)|^{\mathcal{O}(|\phi|)}$. "Efficient solvability" hence in this context often means *fixed-parameter tractability* (FPT); that is, solvability in time $f(|\phi|) \cdot |V(G)|^{\mathcal{O}(1)}$ for some computable function $f$.

In the past two decades algorithmic meta-theorems for FO logic on sparse graph classes received considerable attention. While the algorithm of [5] for MSO on graphs of bounded clique-width implies fixed-parameter tractability of FO model checking on graphs of locally bounded clique-width via Gaifman's locality, one could go far beyond that. After the result of Seese [26] proving fixed-parameter tractability of FO model checking on graphs of bounded degree there followed a series of results [6, 10, 12] establishing the same conclusion

* The full version of this paper is made public as arXiv:1709.03701., `https://arxiv.org/abs/1709.03701`
† P. Hliněný is supported by the Czech Science Foundation project No. 17-00837S.
‡ F. Pokrývka is supported by the Czech Science Foundation project No. 17-00837S.


for increasingly rich sparse graph classes. This line of research culminated in the result of Grohe, Kreutzer and Siebertz [20], who proved that FO model checking is FPT on *nowhere dense* graph classes.

While the result of [20] is the best possible in the following sense – if a graph class $\mathcal{D}$ is *monotone* (closed on taking subgraphs) and not nowhere dense, then the FO model checking problem on $\mathcal{D}$ is as hard as that on all graphs; this does not exclude interesting FPT meta-theorems on *somewhere dense* non-monotone graph classes. Probably the first extensive work of the latter dense kind, beyond locally bounded clique-width, was that of Ganian et al. [16] studying subclasses of interval graphs for which FO model checking is FPT (when only bounded number of interval lengths is used). Another approach has been taken in the works of Bova, Ganian and Szeider [3] and Gajarský et al. [13], which studied FO model checking on posets – posets can be seen as typically quite dense special digraphs. Altogether, however, only very little is known about FO model checking of somewhere dense graph classes (except perhaps specialised [15]).

The result of Gajarský et al. [13] claims that FO model checking is FPT on posets of bounded width (size of a maximum antichain), and it happens to imply [16] in a stronger setting (see below). One remarkable message of [13] is the following (citation): *The result may also be used directly towards establishing fixed-parameter tractability for FO model checking of other graph classes. Given the ease with which it ( [13] ) implies the otherwise non-trivial result on interval graphs [16], it is natural to ask what other (dense) graph classes can be interpreted in posets of bounded width.* Inspired by the geometric case of interval graphs, we propose to study dense graph classes defined in geometric terms, such as intersection and visibility graphs, with respect to tractability of their FO model checking problem.

The motivation for such study is a two-fold. First, intersection and visibility graphs present natural examples of non-monotone somewhere dense graph classes to which the great "sparse" FO tractability result of [20] cannot be (at least not easily) applied. Second, their supplementary geometric structure allows to better understand (as we have seen already in [16]) the boundaries of tractability of FO model checking on them, which is, to current knowledge, terra incognita for hereditary graph classes in general.

Our results mainly concern graph classes which are related to interval graphs. Namely, we prove (Theorem 3.1) that FO model checking is FPT on *circular-arc graphs* (these are interval graphs on a circle) if there is no long chain of arcs nested by inclusion. This directly extends the result of [16] and its aforementioned strengthening in [13] (with bounding chains of nested intervals instead of their lengths). We similarly show tractability of FO model checking of interval-overlap graphs, also known as *circle graphs*, of bounded independent set size (Theorem 4.1), and of restricted subclasses of *box and disk graphs* which naturally generalize interval graphs to two dimensions (Theorem 5.1).

On the other hand, for all of the studied cases we also show that whenever we relax our additional restrictions (parameters), the FO model checking problem becomes as hard on our intersection classes as on all graphs (Corollary 6.2). Some of our hardness claims hold also for the weaker $\exists$FO model checking problem (Proposition 6.3).

Another well studied dense graph class in computational geometry are *visibility graphs* of polygons, which have been largely explored in the context of recognition, partition, guarding and other optimization problems [17,25]. We consider some established special cases, involving *weak visibility, terrain* and *fan* polygons. We prove that FO model checking is FPT for the visibility graphs of a weak visibility polygon of a convex edge, with bounded number of reflex (non-convex) vertices (Theorem 7.2). On the other hand, without bounding reflex vertices, FO model checking remains hard even for the much more special case of polygons that are terrain and convex fans at the same time (Theorem 7.1).

As noted above, our fixed-parameter tractability proofs use the strong result [13] on FO model checking of posets of bounded width. We refer to Section 2 for a detailed explanation of the technical terms used here. Briefly, for a given graph $G$ from the respective class and a formula $\phi$, we show how to efficiently construct a poset $\mathcal{P}_G$ of bounded width and a related FO formula $\phi^I$ such that $G \models \phi$ iff $\mathcal{P}_G \models \phi^I$, and then solve the latter problem.

With respect to the previously known results, we remark that our graph classes are not sparse, as they all contain large complete or complete bipartite subgraphs. For some of them, namely unit circular-arc graphs, circle graphs of bounded independence number, and box graphs (with parameter $k = 2$ as in Theorem 5.1), it can be shown that they are of locally unbounded clique-width by an adaptation of the corresponding argument from [16].

Lastly, we particularly emphasize the seemingly simple tractable case (Corollary 4.2) of permutation graphs of bounded clique size: in relation to so-called stability notion (cf. [1]), already the hereditary class of triangle-free permutation graphs has the $n$-order property (i.e., is *not* stable), and yet FO model checking of this class is FPT. This example presents a natural hereditary and non-stable graph class with FPT FO model checking other than, say, graphs of bounded clique-width. We suggest that if we could fully understand the precise breaking point(s) of FP tractability of FO model checking on simply described intersection classes like the permutation graphs, then we would get much better insight into FP tractability of FO model checking of general hereditary graph classes.

Due to space restrictions, most of the proofs and some illustrating pictures have had to be removed from this short paper. The statements with removed proofs are marked by * and they can be found, for example, in the arXiv version.

## 2 Preliminaries

**Graphs and intersection graphs.** We work with *finite simple undirected graphs* and use standard graph theoretic notation. We refer to the vertex set of a graph $G$ as to $V(G)$ and to its edge set as to $E(G)$, and we write shortly $uv$ for an edge $\{u, v\}$. As it is common in the context of FO logic on graphs, vertices of our graphs can carry arbitrary labels.

Considering a family of sets $\mathcal{S}$ (in our case, of geometric objects in the plane), the *intersection graph of* $\mathcal{S}$ is the simple graph $G$ defined by $V(G) := \mathcal{S}$ and $E(G) := \{AB : A, B \in \mathcal{S}, A \cap B \neq \emptyset\}$. In respect of algorithmic questions, it is important to distinguish whether an intersection graph $G$ is given on the input as an abstract graph $G$, or alongside with its intersection representation $\mathcal{S}$.

One folklore example of a widely studied intersection graph class are *interval graphs* – the intersection graphs of intervals on the real line. Interval graphs enjoy many nice algorithmic properties, e.g., their representation can be constructed quickly, and generally hard problems like clique, independent set and chromatic number are solvable in polynomial time for them.

For a general overview and extensive reference guide of intersection graph classes we suggest to consult the online system ISGCI [7].

**FO logic.** The *first-order logic of graphs* (abbreviated as FO) applies the standard language of first-order logic to a graph $G$ viewed as a relational structure with the domain $V(G)$ and the single binary (symmetric) relation $E(G)$. That is, in graph FO we have got the standard predicate $x = y$, a binary predicate $edge(x, y)$ with the usual meaning $xy \in E(G)$, an arbitrary number of unary predicates $L(x)$ with the meaning that $x$ holds the label $L$, usual logical connectives $\wedge, \vee, \rightarrow$, and quantifiers $\forall x$, $\exists x$ over the vertex set $V(G)$.

For example, $\phi(x,y) \equiv \exists z\big(edge(x,z) \wedge edge(y,z) \wedge red(z)\big)$ states that the vertices $x, y$ have a common neighbour in $G$ which has got label 'red'. One can straightforwardly express in FO properties such as $k$-clique $\exists x_1, \ldots, x_k\big(\bigwedge_{i<j=1}^{k}(edge(x_i,x_j) \wedge x_i \neq x_j)\big)$ and $k$-dominating set $\exists x_1, \ldots, x_k \forall y\big(\bigvee_{i=1}^{k}(edge(x_i,y) \vee y = x_i)\big)$. Specially, an FO formula $\phi$ is *existential* (abbreviated as $\exists$FO) if it can be written as $\phi \equiv \exists x_1, \ldots, x_k\, \psi$ where $\psi$ is quantifier-free. For example, $k$-clique is $\exists$FO while $k$-dominating set is not.

Likewise, FO logic of posets treats a poset $\mathcal{P} = (P, \sqsubseteq)$ as a finite relational structure with the domain $P$ and the (antisymmetric) binary predicate $x \sqsubseteq y$ (instead of the predicate *edge*) with the usual meaning. Again, posets can be arbitrarily labelled by unary predicates.

**Parameterized model checking.** Instances of a parameterized problem can be considered as pairs $\langle I, k\rangle$ where $I$ is the main part of the instance and $k$ is the *parameter* of the instance; the latter is usually a non-negative integer. A parameterized problem is *fixed-parameter tractable (FPT)* if instances $\langle I, k\rangle$ of size $n$ can be solved in time $O(f(k) \cdot n^c)$ where $f$ is a computable function and $c$ is a constant independent of $k$. In *parameterized model checking*, instances are considered in the form $\langle (G, \phi), |\phi|\rangle$ where $G$ is a structure, $\phi$ a formula, the question is whether $G \models \phi$ and the parameter is the size of $\phi$.

When speaking about the FO model checking problem in this paper, we always implicitly consider the formula $\phi$ (precisely its size) as a parameter. We shall use the following result:

▶ **Theorem 2.1** ( [13]). *The FO model checking problem of (arbitrarily labelled) posets, i.e., deciding whether $\mathcal{P} \models \phi$ for a labelled poset $\mathcal{P}$ and FO $\phi$, is fixed-parameter tractable with respect to $|\phi|$ and the width of $\mathcal{P}$ (this is the size of the largest antichain in $\mathcal{P}$).*

We also present, for further illustration, a result on FO model checking of interval graphs with bounded nesting. A set $\mathcal{A}$ of intervals (interval representation) is called *proper* if there is no pair of intervals in $\mathcal{A}$ such that one is contained in the other. We call $\mathcal{A}$ a *$k$-fold proper* set of intervals if there exists a partition $\mathcal{A} = \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_k$ such that each $\mathcal{A}_j$ is a proper interval set for $j = 1, \ldots, k$. Clearly, $\mathcal{A}$ is $k$-fold proper if and only if there is no chain of $k+1$ inclusion-nested intervals in $\mathcal{A}$. From Theorem 2.1 one can, with help of relatively easy arguments (Lemma 3.2), derive the following:

▶ **Theorem 2.2** ( [13], cf. Proposition 2.4 and Lemma 3.2). *Let $G$ be an interval graph given alongside with its $k$-fold proper interval representation $\mathcal{A}$. Then FO model checking of $G$ is FPT with respect to the parameters $k$ and the formula size.*

**Parameterized hardness.** For some parameterized problems, like the $k$-clique on all graphs, we do not have nor expect any FPT algorithm. To this end, the theory of parameterized complexity of Downey and Fellows [8] defines complexity classes $W[t]$, $t \geq 1$, such that the $k$-clique problem is complete for $W[1]$ (the least class). Furthermore, theory also defines a larger complexity class $AW[*]$ containing all of $W[t]$. Problems that are $W[1]$-hard do not admit an FPT algorithm unless the established Exponential Time Hypothesis fails.

▶ **Theorem 2.3** ( [9]). *The FO model checking problem (where the formula size is the parameter) of all simple graphs is $AW[*]$-complete.*

Dealing with parameterized hardness of FO model checking, one should also mention the related *induced subgraph isomorphism problem*: for a given input graph $G$, and a graph $H$ as the parameter, decide whether $G$ has an induced subgraph isomorphic to $H$. Note that this includes the clique and independent set problems. Induced subgraph isomorphism

(parameterized by the subgraph size) is clearly a weaker problem than parameterized FO model checking, since one may "guess" the subgraph with $|V(H)|$ existential quantifiers and then verify it edge by edge. Consequently, every parameterized hardness result for induced subgraph isomorphism readily implies same hardness results for ∃FO and FO model checking.

**FO interpretations.** Interpretations are a standard tool of logic and finite model theory. To keep our paper short, we present here only a simplified description of them, tailored specifically to our need of interpreting geometric graphs in posets.

An *FO interpretation* is a pair $I = (\nu, \psi)$ of poset FO formulas $\nu(x)$ and $\psi(x, y)$ (of one and two free variables, respectively). For a poset $\mathcal{P}$, this defines a graph $G := I(\mathcal{P})$ such that $V(G) = \{v : \mathcal{P} \models \nu(v)\}$ and $E(G) = \{uv : u, v \in V(G), \mathcal{P} \models \psi(u, v) \vee \psi(v, u)\}$. Possible labels of the elements are naturally inherited from $\mathcal{P}$ to $G$. Moreover, for a graph FO formula $\phi$ the interpretation $I$ defines a poset FO formula $\phi^I$ recursively as follows: every occurrence of $edge(x, y)$ is replaced by $\psi(x, y) \vee \psi(y, x)$, every $\exists x\, \sigma$ is replaced by $\exists x\, (\nu(x) \wedge \sigma)$ and $\forall x\, \sigma$ by $\forall x\, (\nu(x) \to \sigma)$. Then, obviously, $\mathcal{P} \models \phi^I \iff G \models \phi$.

Usefulness of the concept is illustrated by the following trivial claim:

▶ **Proposition 2.4.*** *Let $\mathcal{P}$ be a class of posets such that the FO model checking problem of $\mathcal{P}$ is FPT, and let $\mathcal{G}$ be a class of graphs. Assume there is a computable FO interpretation $I$, and for every graph $G \in \mathcal{G}$ we can in polynomial time compute a poset $\mathcal{P} \in \mathcal{P}$ such that $G = I(\mathcal{P})$. Then the FO model checking problem of $\mathcal{G}$ is in FPT.*

## 3 Circular-arc Graphs

*Circular-arc graphs* are intersection graphs of arcs (curved intervals) on a circle. They clearly form a superclass of interval graphs, and they enjoy similar nice algorithmic properties as interval graphs, such as efficient construction of the representation [24], and easy computation of, say, maximum independent set or clique.

Since the FO model checking problem is $AW[*]$-complete on interval graphs [16], the same holds for circular-arc graphs in general. Furthermore, by [21, 23] already ∃FO model checking is $W[1]$-hard for interval and circular-arc graphs. A common feature of these hardness reductions (see more discussion in Section 6) is their use of unlimited chains of nested intervals/arcs. Analogously to Theorem 2.2, we prove that considering only *$k$-fold proper circular-arc* representations (the definition is the same as for $k$-fold proper interval representations) makes FO model checking of circular-arc graphs tractable.

▶ **Theorem 3.1.** *Let $G$ be a circular-arc graph given alongside with its $k$-fold proper circular-arc representation $\mathcal{A}$. Then FO model checking of $G$ is FPT with respect to the parameters $k$ and the formula size.*

Note that we can (at least partially) avoid the assumption of having a representation $\mathcal{A}$ in the following sense. Given an input graph $G$, we compute a circular-arc representation $\mathcal{A}$ using [24], and then we easily determine the least $k'$ such that $\mathcal{A}$ is $k'$-fold proper. However, without further considerations, this is not guaranteed to provide the minimum $k$ over all circular-arc representations of $G$, and not even $k'$ bounded in terms of the minimum $k$.

Our proof will be based on the following extension of the related argument from [13]:

▶ **Lemma 3.2** (parts from [13, Section 5])**.*** *Let $\mathcal{B}$ be a $k$-fold proper set of intervals for some integer $k > 0$, such that no two intervals of $\mathcal{B}$ share an endpoint. There exist formulas $\nu, \psi, \vartheta$ depending on $k$, and a labelled poset $\mathcal{P}$ of width $k + 1$ computable in polynomial time from $\mathcal{B}$, such that all the following hold:*

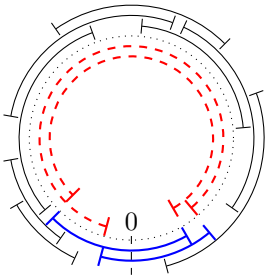



**Figure 1** An illustration; a proper circular-arc representation $\mathcal{A}$ (ordinary black and thick blue arcs), giving raise to a 2-fold proper interval set $\mathcal{B}$ (ordinary black and dashed red arcs), as in the proof of Theorem 3.1. The red arcs are complements of the corresponding blue arcs.

- *The domain of* $\mathcal{P}$ *includes (the intervals from)* $\mathcal{B}$*, and* $\mathcal{P} \models \nu(x)$ *iff* $x \in \mathcal{B}$*,*
- $\mathcal{P} \models \psi(x,y)$ *for intervals* $x, y \in \mathcal{B}$ *iff* $x \cap y \neq \emptyset$ *(edge relation of the interval graph of* $\mathcal{B}$*),*
- $\mathcal{P} \models \vartheta(x,y)$ *for intervals* $x, y \in \mathcal{B}$ *iff* $x \subseteq y$ *(containment of intervals).*

**Proof of Theorem 3.1.** We consider each arc of $\mathcal{A}$ in angular coordinates as $[\alpha, \beta]$ clockwise, where $\alpha, \beta \in [0, 2\pi)$. By standard arguments (a "small perturbation"), we can assume that no two arcs share the same endpoint, and no arc starts or ends in (the angle) 0. Let $\mathcal{A}_0 \subseteq \mathcal{A}$ denote the subset of arcs containing 0. Note that for every arc $[\alpha, \beta] \in \mathcal{A}_0$ we have $\alpha > \beta$, and we subsequently define $\mathcal{A}_1 := \big\{[\beta, \alpha] : [\alpha, \beta] \in \mathcal{A}_0\big\}$ as the set of their "complementary" arcs avoiding 0. For $a \in \mathcal{A}_0$ we shortly denote by $\bar{a} \in \mathcal{A}_1$ its complementary arc.

Now, the set $\mathcal{B} := (\mathcal{A} \setminus \mathcal{A}_0) \cup \mathcal{A}_1$ is an ordinary interval representation contained in the open line segment $(0, 2\pi)$. See Figure 1. Since each of $\mathcal{A} \setminus \mathcal{A}_0$ and $\mathcal{A}_1$ is $k$-fold proper by the assumption on $\mathcal{A}$, the representation $\mathcal{B}$ is $2k$-fold proper. Note the following facts; every two intervals in $\mathcal{A}_0$ intersect, and an interval $a \in \mathcal{A}_0$ intersects $b \in \mathcal{A} \setminus \mathcal{A}_0$ iff $b \not\subset \bar{a}$.

We now apply Lemma 3.2 to the set $\mathcal{B}$, constructing a (labelled) poset $\mathcal{P}$ of width at most $2k+1$. We also add a new label *red* to the elements of $\mathcal{P}$ which represent the arcs in $\mathcal{A}_1$. The final step will give a definition of an FO interpretation $I = (\nu, \psi_1)$ such that $I(\mathcal{P})$ will be isomorphic to the intersection graph $G$ of $\mathcal{A}$. Using the formulas $\psi, \vartheta$ from Lemma 3.2, the latter is also quite easy. As mentioned above, intersecting pairs of intervals from $\mathcal{A}$ can be described using intersection and containment of the corresponding intervals of $\mathcal{B}$:

$$\psi_1(x,y) \equiv \big(red(x) \wedge red(y)\big) \vee \big(\neg red(x) \wedge \neg red(y) \wedge \psi(x,y)\big) \vee \big(red(x) \wedge \neg red(y) \wedge \neg\vartheta(y,x)\big)$$

It is routine to verify that, indeed, $G \simeq I(\mathcal{P})$ (using the obvious bijection of $\mathcal{A}_0$ to $\mathcal{A}_1$).

We then finish simply by Theorem 2.1 and Proposition 2.4. ◀

## 4 Circle graphs

Another graph class closely related to interval graphs are *circle graphs*, also known as *interval overlap graphs*. These are intersection graphs of chords of a circle, and they can equivalently be characterised as having an *overlap* interval representation $\mathcal{C}$ such that $a, b \in \mathcal{C}$ form an edge, if and only if $a \cap b \neq \emptyset$ but neither $a \subseteq b$ nor $b \subseteq a$ hold (see Figure 2). A circle representation of a circle graph can be efficiently constructed [2].

Related *permutation graphs* are defined as intersection graphs of line segments with the ends on two parallel lines, and they form a complementation-closed subclass of circle graphs.

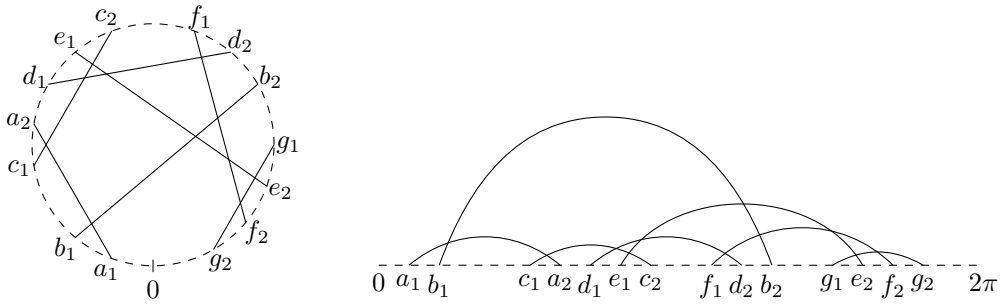



**Figure 2** "Opening" a circle representation (left; an intersecting system of chords of a circle) into an overlap representation (right; the depicted arcs to be flattened into intervals on the line).

Note another easy characterization: let $G$ be a graph and $G_1$ be obtained by adding one vertex adjacent to all vertices of $G$; then $G$ is a permutation graph if and only if $G_1$ is a circle graph. We will see in Section 6 that the ∃FO model checking problem is $W[1]$-hard for circle graphs, and the FO model checking problem is $AW[*]$-complete already for permutation graphs. However, there is also a positive result using a natural additional parameterization. The proof of it uses arguments similar to those of Theorem 3.1.

▶ **Theorem 4.1.*** *The FO model checking problem of circle graphs is FPT with respect to the formula and the maximum independent set size.*

An interesting question is whether 'independent set size' in Theorem 4.1 can also be replaced with 'clique size'. We think the right answer is 'yes', but we have not yet found the algorithm. At least, the answer is positive for the subclass of permutation graphs:

▶ **Corollary 4.2.*** *The FO model checking problem of permutation graphs is FPT with respect to the formula size, and either the maximum clique or the maximum independent set size.*

▶ **Corollary 4.3.*** *The subgraph isomorphism (not induced) problem of permutation graphs is FPT with respect to the subgraph size.*

## 5 Box and Disk graphs

*Box (intersection) graphs* are graphs having an intersection representation by rectangles in the plane, such that each rectangle (box) has its sides parallel to the x- and y-axes. The recognition problem of box graphs is NP-hard [28], and so it is essential that the input of our algorithm would consist of a box representation. *Unit-box graphs* are those having a representation by unit boxes.

The ∃FO model checking problem is $W[1]$-hard already for unit-box graphs [22], and we will furthermore show that it stays hard if we restrict the representation to a small area in Proposition 6.3. Here we give the following slight extension of Theorem 2.2:

▶ **Theorem 5.1.*** *Let $G$ be a box intersection graph given alongside with its box representation $\mathcal{B}$ such that the following holds: the projection of $\mathcal{B}$ to the $x$-axis is a $k$-fold proper set of intervals, and the projection of $\mathcal{B}$ to the $y$-axis consists of at most $k$ distinct intervals. Then FO model checking of $G$ is FPT with respect to the parameters $k$ and the formula size.*

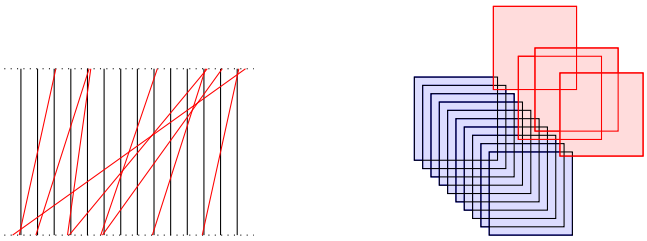


**Figure 3** Constructing witnesses of the consecutive neighbourhood representation property – as permutation graphs (left) and as unit-box graphs (right); cf. Corollary 6.2.

Furthermore, *disk graphs* are those having an intersection representation by disks in the plane. Their recognition problem is NP-hard already with unit disks [4], and the ∃FO model checking problem is $W[1]$-hard again for unit-disk graphs by [22]. Similarly to Theorem 5.1, we have identified a tractable case of FO model checking of unit-disk graph, based on restricting the y-coordinates of the disks. Due to space restrictions, we leave this case only for the full paper.

## 6 Hardness for intersection classes

Our aim is to provide a generic reduction for proving hardness of FO model checking (even without labels on vertices) using only a simple property which is easy to establish for many geometric intersection graph classes. We will then use it to derive hardness of FO for quite restricted forms of intersection representations studied in our paper (Corollary 6.2).

We say that a graph $G$ *represents consecutive neighbourhoods* of order $\ell$, if there exists a sequence $S = (v_1, v_2, \ldots, v_\ell) \subseteq V(G)$ of distinct vertices of $G$ and a set $R \subseteq V(G)$, $R \cap S = \emptyset$, such that for each pair $i, j$, $1 \leq i < j \leq \ell$, there is a vertex $w \in R$ whose neighbours in $S$ are precisely the vertices $v_i, v_{i+1} \ldots, v_j$. (Possible edges other than those between $R$ and $S$ do not matter.) A graph class $\mathcal{G}$ has the *consecutive neighbourhood representation* property if, for every integer $\ell > 0$, there exists an efficiently computable graph $G \in \mathcal{G}$ such that $G$ or its complement $\overline{G}$ represents consecutive neighbourhoods of order $\ell$.

Note that our notion of 'representing consecutive neighbourhoods' is related to the concepts of "$n$-order property" and "stability" from model theory (mentioned in Section 1). This is not a random coincidence, as it is known [1] that on monotone graph classes stability coincides with nowhere dense (which is the most general characterization allowing for FPT FO model checking on monotone classes). In our approach, we stress easy applicability of this notion to a wide range of geometric intersection graphs and, to certain extent, to ∃FO model checking.

The main result is as follows. A *duplication* of a vertex $v$ in $G$ is the operation of adding a *true twin* $v'$ to $v$, i.e., new $v'$ adjacent to $v$ and precisely to the neighbours of $v$ in $G$.

▶ **Theorem 6.1.*** *Let $\mathcal{G}$ be a class of unlabelled graphs having the consecutive neighbourhood representation property, and $\mathcal{G}$ be closed on induced subgraphs and duplication of vertices. Then the FO model checking of $\mathcal{G}$ is $AW[*]$-complete with respect to the formula size.*

Graphs witnessing the consecutive neighbourhood representation property can be easily constructed within our intersection classes, even with strong further restrictions. See some illustrating examples in Figure 3. So, we obtain the following hardness results:

▶ **Corollary 6.2.** * *The FO model checking problem is AW[∗]-complete with respect to the formula size, for each of the following geometric graph classes (all unlabelled):*

**(a)** *circular-arc graphs with a representation consisting or arcs of lengths from* $[\pi - \varepsilon, \pi + \varepsilon]$ *on the circle of diameter* 1*, for any fixed* $\varepsilon > 0$*,*

**(b)** *connected permutation graphs,*

**(c)** *unit-box graphs with a representation contained within a square of side length* $2 + \varepsilon$*, for any fixed* $\varepsilon > 0$*,*

**(d)** *unit-disk graphs (that is of diameter* 1*) with a representation contained within a rectangle of sides* $1 + \varepsilon$ *and* 2*, for any fixed* $\varepsilon > 0$*.*

It is worthwhile to notice that for each of the classes listed in Corollary 6.2, the $k$-clique and $k$-independent set problems are all easily FPT, and yet FO model checking is not.

Finally, we return to the weaker ∃FO model checking problem. In fact, this problem can be treated "the same" as the aforementioned parameterized induced subgraph isomorphism problem: precisely, one of them admits an FPT algorithm on any given (unlabelled) graph class if and only if the other does so.

The hardness construction in the proof of Theorem 6.1 can be turned into ∃FO, but only if vertex labels are allowed. Though, we can modify some of the constructions from Corollary 6.2 to capture also ∃FO without labels.

▶ **Proposition 6.3.** * *The ∃FO model checking problem is W[1]-hard with respect to the formula size, for both the following unlabelled geometric graph classes:*

**(a)** *circle graphs,*

**(b)** *unit-box graphs with a representation contained within a square of side length* 3*.*

One complexity question that remains open after Proposition 6.3 is about ∃FO on unlabelled permutation graphs (for labelled ones, this is $W[1]$-hard by the remark after Corollary 6.2). While induced subgraph isomorphism is generally NP-hard on permutation graphs by [21], we are not aware of results on the parameterized version, and we currently have no plausible conjecture about its parameterized complexity.

## 7 Polygonal visibility graphs

Given a polygon $W$ in the plane, two vertices $p_i$ and $p_j$ of $W$ are said to be *mutually visible* if the line segment $p_ip_j$ does not intersect the exterior of $W$. The *visibility graph* $G$ of $W$ is defined to have vertices $v_i$ corresponding to each vertex $p_i$ of $W$, and edge $(v_i, v_j)$ if and only if $p_i$ and $p_j$ are mutually visible. Our aim is to study the visibility graphs of some special established classes of polygons with respect to FO model checking.

If there is an edge $e$ of the polygon $W$, such that for any point $p$ of $W$, there is a point on $e$ that sees $p$, then $W$ is called a *weak visibility polygon*, and $e$ is called a *weak visibility edge* of $W$ (Figure 4a) [17,18]. A vertex $v_i$ of $W$ is called a *reflex vertex* if the interior angle of $W$ formed at $v_i$ by the two edges of $W$ incident to $v_i$ is more than $\pi$. Otherwise, $v_i$ is called a *convex vertex*. If both of the end vertices of an edge of $W$ are convex vertices, then the edge is called a *convex edge*.

If the boundary of $W$ consists only of an x-monotone polygonal arc touching the x-axis at its two extreme points, and an edge contained in the x-axis joining the two points, then it is called a *terrain* (Figure 4b) [11,17]. All terrains are weak visibility polygons with respect to their edge that lies on the x-axis. If all points of a $W$ are visible from a single vertex $v$ of the polygon, then $W$ is called a *fan* (Figure 4c) [17,19]. If $W$ is a fan with respect to a

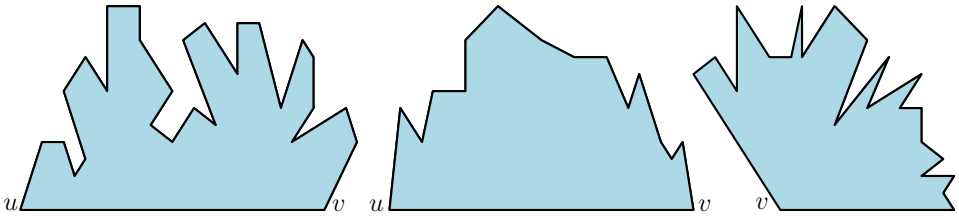



**Figure 4** From left to right: (a) a weak visibility polygon with respect to edge $uv$; (b) a terrain; (c) a convex fan visible from the vertex $v$.

convex vertex $v$, then $W$ is called a *convex fan* [25]. If $W$ is a convex fan with respect to a vertex $v$, then both of the edges of $W$ incident to $v$ are convex edges, and $W$ is also a weak visibility polygon with respect to any of them.

In this section we identify some interesting tractable and hard cases of the FO model checking problem on these visibility classes.

We first argue that the FO model checking problem of polygon visibility graphs stays hard even when the polygon is a terrain and a convex fan. Our approach is very similar to that in Theorem 6.1 above, that is, we show that a given FO model checking instance of general graphs can be interpreted in another instance of the visibility graph of a specially constructed polygon which is a terrain and a convex fan at the same time.

▶ **Theorem 7.1.*** *The FO model checking problem of unlabelled polygon visibility graphs (given alongside with the representing polygon) is AW[∗]-complete with respect to the formula size, even when the polygon is a terrain and a convex fan at the same time.*

Second, we prove that FO model checking of the visibility graph of a given weak visibility polygon of a convex edge is FPT when additionally parameterized by the number of reflex vertices. We remark that, for example, the independent set problem is NP-hard on polygonal visibility graphs [27], but Ghosh et al. [18] showed that the maximum independent set of the visibility graph of a given weak visibility polygon of a convex edge, is computable in quadratic time. In Theorem 7.1, we have seen that the latter result does not generalise to arbitrary FO properties, since FO model checking remains hard even for a very special subcase of weak visibility polygons. So, an additional parameterization is necessary.

▶ **Theorem 7.2.*** *Let $W$ be a given polygon weakly visible from one of its convex edges, with $k$ reflex vertices, and let $G$ be the visibility graph of $W$. Then FO model checking of $G$ is FPT with respect to the parameters $k$ and the formula size.*

While we cannot fit the whole algorithm in the short paper, we at least give an informal overview of how the algorithm works. As in the previous intersection graph cases, our aim is to construct, from given $W$, a poset $\mathcal{P}$ such that the width of $\mathcal{P}$ is bounded by a function of $k$ and that we have an FO interpretation of the visibility graph of $W$ in this $\mathcal{P}$.

Let $W$ be weakly visible from its convex edge $uv$, and denote by $C_{uv}$ the clockwise sequence of the vertices of $W$ from $u$ to $v$. The subsequence of $C_{uv}$ between two reflex vertices $v_a$ and $v_b$, such that all vertices in it are convex, is called an *ear* of $W$. The length of this sequence can be 0 as well. Additionally, the first (last) ear of $W$ is defined as the subsequence between $u$ and the first reflex vertex of $C_{uv}$ (between the last reflex vertex and $v$, respectively). We have got $k+1$ ears in $W$. With a slight abuse of terminology at $u, v$, we may simply say that an ear is a sequence of convex vertices between two reflex vertices.

The crucial idea of our construction of the poset $\mathcal{P}$ (which contains all vertices of $W$, in particular) is that the visibility edges between the internal (convex) vertices of the ears are

nicely structured: withing one ear $E_a$, they form a clique, and between two ears $E_a, E_b$, the visibility edges exhibit a "shifting pattern" not much different from the left and right ends of intervals in a proper interval representation (cf. Lemma 3.2). Consequently, we may "encode" all the edges between $E_a$ and $E_b$ with help of an extra subposet of $\mathcal{P}$ of fixed width, and since we have got only $k+1$ ears, this together gives a poset of width bounded in $k$.

The last step concerns visibility edges incident with one of the $k$ reflex vertices or $u, v$. These can be easily encoded in $\mathcal{P}$ with only $2(k+2)$ additional labels, without any assumption on the structure of $\mathcal{P}$: for each reflex vertex $x$ of $C_{uv}$, or $x \in \{u, v\}$, we assign one new label $L^0_x$ to $x$ itself and another new label $L^1_x$ to all the neighbours of $x$. Altogether, we can efficiently construct an FO interpretation of $G$ in $\mathcal{P}$ such that the formulas depend only on $k$. Then we may finish by Theorem 2.1.

## 8 Conclusions

We have identified several FP tractable cases of the FO model checking problem of geometric graphs, and complemented these by hardness results showing quite strict limits of FP tractability on the studied classes. Overall, this presents a nontrivial new contribution towards understanding on which (hereditary) dense graph classes can FO model checking be FPT.

All our tractability results rely on the FO model checking algorithm of [13], which is mainly of theoretical interest. However, in some cases one can employ, in the same way, the simple and practical ∃FO model checking algorithm of [14]. We would also like to mention the possibility of enhancing the result of [13] via interpreting posets in posets. While this might seem impossible, we actually have one positive indication of such an enhancement. It is known that interval graphs are $C_4$-free complements of comparability graphs (i.e., of posets) – the width of which is the maximum clique size of the original interval graph. Then, among $k$-fold proper interval graphs there are ones of unbounded clique size, which have FPT FO model checking by Theorem 2.2. This opens a promising possibility of an FP tractable subcase of FO model checking of posets of unbounded width, for future research.

Finally, we list two concrete open problems related to our results. We conjecture that FO model checking is FPT for

- circle graphs additionally parameterized by the maximum clique size,
- visibility graphs of weak visibility polygons additionally parameterized by the maximum independent set size.

## References


1 H. Adler and I. Adler. Interpreting nowhere dense graph classes as a classical notion of model theory. *Eur. J. Comb.*, 36:322–330, 2014.

2 A. Bouchet. Reducing prime graphs and recognizing circle graphs. *Combinatorica*, 7:243–254, 1987.

3 S. Bova, R. Ganian, and S. Szeider. Model checking existential logic on partially ordered sets. *ACM Trans. Comput. Log.*, 17(2):10:1–10:35, 2016.

4 H. Breu and D. G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Computational Geometry*, 9(1-2):3–24, 1998.

5 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.

6 A. Dawar, M. Grohe, and S. Kreutzer. Locally excluding a minor. In *LICS'07*, pages 270–279. IEEE Computer Society, 2007.

7 H. N. de Ridder et al. Information System on Graph Classes and their Inclusions (ISGCI). http://www.graphclasses.org.

8 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.

9 Rodney G. Downey, Michael R. Fellows, and Udayan Taylor. The parameterized complexity of relational database queries and an improved characterization of W[1]. In *First Conference of the Centre for Discrete Mathematics and Theoretical Computer Science, DMTCS 1996, New Zealand, December, 9-13, 1996*, pages 194–213. Springer-Verlag, Singapore, 1996.

10 Z. Dvořák, D. Kráľ, and R. Thomas. Deciding first-order properties for sparse graphs. In *FOCS'10*, pages 133–142. IEEE Computer Society, 2010.

11 S. Eidenbenz. In-approximability of finding maximum hidden sets on polygons and terrains. *Computational Geometry: Theory and Applications*, 21:139–153, 2002.

12 M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001.

13 J. Gajarský, P. Hliněný, D. Lokshtanov, J. Obdržálek, S. Ordyniak, M. S. Ramanujan, and S. Saurabh. FO model checking on posets of bounded width. In *FOCS'15*, pages 963–974. IEEE Computer Society, 2015. Full paper arXiv:1504.04115.

14 J. Gajarský, P. Hliněný, J. Obdržálek, and S. Ordyniak. Faster existential FO model checking on posets. In *ISAAC'14*, volume 8889 of *LNCS*, pages 441–451. Springer, 2014.

15 J. Gajarský, P. Hliněný, J. Obdržálek, D. Lokshtanov, and M. S. Ramanujan. A new perspective on FO model checking of dense graph classes. In *LICS '16*, pages 176–184. ACM, 2016.

16 R. Ganian, P. Hliněný, D. Kráľ, J. Obdržálek, J. Schwartz, and J. Teska. FO model checking of interval graphs. *Log. Methods Comput. Sci.*, 11(4:11):1–20, 2015.

17 S. K. Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, 2007.

18 S. K. Ghosh, A. Maheshwari, S. P. Pal, S. Saluja, and C. E. Veni Madhavan. Characterizing and recognizing weak visibility polygons. *Computational Geometry: Theory and Applications*, 3:213–233, 1993.

19 S. K. Ghosh, T. Shermer, B. K. Bhattacharya, and P. P. Goswami. Computing the maximum clique in the visibility graph of a simple polygon. *Journal of Discrete Algorithms*, 5:524–532, 2007.

20 M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. In *STOC'14*, pages 89–98. ACM, 2014.

21 P. Heggernes, P. van 't Hof, D. Meister, and Y. Villanger. Induced subgraph isomorphism on proper interval and bipartite permutation graphs. *Theoretical Computer Science*, 562:252–269, 2015.

22 D. Marx. Efficient approximation schemes for geometric problems? In *Algorithms - ESA 2005, 13th Annual European Symposium, Proceedings*, volume 3669 of *Lecture Notes in Computer Science*, pages 448–459. Springer, 2005.

23 D. Marx and I. Schlotter. Cleaning interval graphs. *Algorithmica*, 65(2):275–316, 2013.

24 R. M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147, 2003.

25 J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, 1987.

26 D. Seese. Linear time computable problems and first-order descriptions. *Math. Structures Comput. Sci.*, 6(6):505–526, 1996.

27 T. Shermer. Hiding people in polygons. *Computing*, 42:109–131, 1989.

28 M. Yannakakis. The complexity of the partial order dimension problem. *SIAM J. Algebraic Discrete Methods*, 3:351–358, 1982.

# Smaller Parameters for Vertex Cover Kernelization*

**Eva-Maria C. Hols[1] and Stefan Kratsch[2]**

1 **Department of Computer Science, University of Bonn, Germany**
`hols@cs.uni-bonn.de`
2 **Department of Computer Science, University of Bonn, Germany**
`kratsch@cs.uni-bonn.de`

## Abstract

We revisit the topic of polynomial kernels for VERTEX COVER relative to structural parameters. Our starting point is a recent paper due to Fomin and Strømme [WG 2016] who gave a kernel with $\mathcal{O}(|X|^{12})$ vertices when $X$ is a vertex set such that each connected component of $G - X$ contains at most one cycle, i.e., $X$ is a modulator to a pseudoforest. We strongly generalize this result by using modulators to $d$-quasi-forests, i.e., graphs where each connected component has a feedback vertex set of size at most $d$, and obtain kernels with $\mathcal{O}(|X|^{3d+9})$ vertices. Our result relies on proving that minimal blocking sets in a $d$-quasi-forest have size at most $d + 2$. This bound is tight and there is a related lower bound of $\mathcal{O}(|X|^{d+2-\epsilon})$ on the bit size of kernels.

In fact, we also get bounds for minimal blocking sets of more general graph classes: For $d$-quasi-bipartite graphs, where each connected component can be made bipartite by deleting at most $d$ vertices, we get the same tight bound of $d + 2$ vertices. For graphs whose connected components each have a vertex cover of cost at most $d$ more than the best fractional vertex cover, which we call $d$-quasi-integral, we show that minimal blocking sets have size at most $2d + 2$, which is also tight. Combined with existing randomized polynomial kernelizations this leads to randomized polynomial kernelizations for modulators to $d$-quasi-bipartite and $d$-quasi-integral graphs. There are lower bounds of $\mathcal{O}(|X|^{d+2-\epsilon})$ and $\mathcal{O}(|X|^{2d+2-\epsilon})$ for the bit size of such kernels.

**1998 ACM Subject Classification** G.2.2 Graph Algorithms

**Keywords and phrases** Vertex Cover, Kernelization, Structural Parameterization

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.20

## 1 Introduction

The VERTEX COVER problem plays a central role in parameterized complexity. In particular, it has been very important for the development of new kernelization techniques and the study of structural parameters. As a result of this work, there is now a solid understanding of which parameterizations of VERTEX COVER lead to fixed-parameter tractability or existence of a polynomial kernelization. This is motivated by the fact that parameterization by solution size leads to large parameter values on many types of easy instances. Thus, while there is a well-known kernelization for instances of VERTEX COVER$(k)$ to at most $2k$ vertices, it may be more suitable to apply a kernelization with a size guarantee that is a larger function but depends on a smaller parameter.

Jansen and Bodlaender [13] were the first to study kernelization for VERTEX COVER under different, smaller parameters. Their main result is a polynomial kernelization to instances

* A full version of the paper is available at `http://arxiv.org/abs/1711.04604`.


with $\mathcal{O}(|X|^3)$ vertices when $X$ is a feedback vertex set of the input graph, also called a *modulator* to the class of forests. Clearly, the size of $X$ is a lower bound on the vertex cover size (as any vertex cover is a modulator to an independent set). Since then, their result has been generalized and complemented in several ways. The two main directions of follow-up work are to use modulators to other tractable cases instead of forests (see below) and parameterization above lower bounds (see related work).

For any graph class $\mathcal{C}$, we can define a parameterization of VERTEX COVER by distance to $\mathcal{C}$, i.e., by the minimum size of a modulator $X$ such that $G - X$ belongs to $\mathcal{C}$. For fixed-parameter tractability and kernelization of the arising parameterized problem it is necessary that VERTEX COVER is tractable on inputs from $\mathcal{C}$. For hereditary classes $\mathcal{C}$, this condition is also sufficient for fixed-parameter tractability but not necessarily for the existence of a polynomial kernelization. Interesting choices for $\mathcal{C}$ are various well-studied hereditary graph classes, like forests, bipartite, or chordal graphs, and graphs of bounded treewidth, bounded treedepth, or bounded degree.

Majumdar et al. [16] studied VERTEX COVER parameterized by (the size of) a modulator $X$ to a graph of maximum degree at most $d$. For $d \geq 3$ this problem is NP-hard but for $d = 2$ and $d = 1$ they obtained kernels with $\mathcal{O}(|X|^5)$ and $\mathcal{O}(|X|^2)$ vertices, respectively. Their result motivated Fomin and Strømme [9] to investigate a parameter that is smaller than both a modulator to degree at most two and the size of a feedback vertex set: They consider $X$ being a modulator to a *pseudoforest*, i.e., with each connected component of $G - X$ having at most one cycle. For this they obtain a kernelization to $\mathcal{O}(|X|^{12})$ vertices, generalizing (except for the size) the results of Majumdar et al. [16] and Jansen and Bodlaender [13]. They also prove that the parameterization by a modulator to so-called *mock forests*, where no cycles share a vertex, admits no polynomial kernelization unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

For their kernelization, Fomin and Strømme [9] prove that minimal blocking sets in a pseudoforest have size at most three, which requires a lengthy proof. (A minimal blocking set is a set of vertices whose deletion decreases the independence number by exactly one.)[1] This allows to reduce the number of components of the pseudoforest such that one can extend the modulator $X$ to a sufficiently small feedback vertex set by adding one (cycle) vertex per component to $X$. At this point, the kernelization of Jansen and Bodlaender [13] can be applied to get the result.

The results of Fomin and Strømme [9] suggest that the border for existence of polynomial kernels for feedback vertex set-like parameters may be much more interesting than expected previously. Arguably, there is still quite some room between allowing a single cycle per component and allowing an arbitrary number of cycles so long as they share no vertices. Do larger numbers of cycles per component still allow a polynomial kernelization? Similarly, cycles in the lower bound proof have odd length and it is known that absence of odd cycles is sufficient, i.e., a kernelization for modulators to bipartite graphs is known. Could this be extended to allowing bipartite graphs with one or more odd cycles per connected component?

**Our work.** We show that the answers to the above questions are largely positive and provide, essentially, a single elegant proof to cover them. To this end, it is convenient to take the perspective of feedback sets rather than the maximum size of a cycle packing. Say that a *$d$-quasi-forest* is a graph such that each connected component has a feedback vertex set of size at most $d$, whereas in a *$d$-quasi-bipartite graph* each connected component must have an odd cycle transversal (a feedback set for odd cycles) of size at most $d$.

[1] Like previous work [13, 9] we prefer to work with INDEPENDENT SET rather than VERTEX COVER, but this makes no important difference.

We show that VERTEX COVER admits a kernelization with $\mathcal{O}(|X|^{3d+9})$ vertices when $X$ is a modulator to a $d$-quasi-forest (Section 3). The case for $d = 1$ strengthens the result of Fomin and Strømme [9] (as one cycle per component is stricter than feedback vertex set size one). For every fixed larger value of $d$ we obtain a polynomial kernelization, though of increasing size. The result is obtained by proving that minimal blocking sets in a $d$-quasi-forest have size at most $d+2$ (and then applying [13]). Intuitively, having a large minimal blocking set implies getting a fairly small maximum independent set because there are optimal independent sets that avoid all but any chosen vertex of a minimal blocking set. In contrast, a $d$-quasi-forest always has a large independent set because each connected component is almost a tree.

The value $d + 2$ is tight already for cliques of size $d + 2$, which are permissible connected components in a $d$-quasi-forest. Such cliques also imply that our parameterization inherits a lower bound of $\mathcal{O}(|X|^{d+2-\epsilon})$ from the lower bound of $\mathcal{O}(|X'|^{r-\epsilon})$ (assuming $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$) for $X'$ being a modulator to a cluster graph with component size at most $r$ [16].

It turns out that our proof directly extends also to $d$-quasi-bipartite graphs, proving that their minimal blocking sets similarly have size at most $d + 2$ (Section 4). Thus, when given a modulator $X$ such that $G - X$ is $d$-quasi-bipartite, we can extend it to an odd cycle transversal $X'$ of size at most $d \cdot |X|^{d+3} + |X|$, which directly yields a randomized polynomial kernel by using a randomized polynomial kernelization for VERTEX COVER parameterized by an odd cycle transversal [15]. Motivated by this, we explore also modulators to graphs in which each connected component has vertex cover size at most $d$ plus the size of a minimum fractional vertex cover, which we call $d$-quasi-integral (Section 4). This is stronger than the previous parameter because it allows connected components that have an odd cycle transversal of size at most $d$. We show that minimal blocking sets in any $d$-quasi-integral graph have size at most $2d + 2$. This bound is tight, as witnessed by the cliques with $2d + 2$ vertices, and the problem inherits a lower bound of $\mathcal{O}(|X|^{2d+2-\epsilon})$ from the lower bound for modulators to cluster graphs with clique size at most $r = 2d + 2$ [16]. Using the upper bound of $2d + 2$ one can remove redundant connected components until the obtained instance has vertex cover size at most $d \cdot |X|^{2d+3} + |X|$ more than the best fractional vertex cover. In other words, one can reduce to an instance of VERTEX COVER parameterized above LP with parameter value $d \cdot |X|^{2d+3} + |X|$ and apply the randomized polynomial kernelization of Kratsch and Wahlström [15] to get a randomized polynomial kernel.

**Related work.** Recent work of Bougeret and Sau [5] shows that VERTEX COVER admits a kernel of size $\mathcal{O}(|X|^{f(c)})$ when $X$ is a modulator to a graph of treedepth at most $c$. Their result is incomparable to ours: Already the kernelization by feedback vertex set size [13], which we generalize, allows arbitrarily long paths in $G - X$; such paths are forbidden in a graph of bounded treedepth. Conversely, taking a star with $d$ leaves and appending a 3-cycle at each leaf yields a graph with feedback vertex set and odd cycle transversal size equal to $d$ but constant treedepth; $d$ can be chosen arbitrarily large.

The fact that deciding whether a graph $G$ has a vertex cover of size at most $k$ is trivial when $k$ is lower than the size $MM(G)$ of a largest matching in $G$ has motivated the study of above lower bound parameters like $\ell = k - MM(G)$. The strongest lower bound employed so far is $2LP(G) - MM(G)$, where $LP(G)$ denotes the minimum cost of a fractional vertex cover, and Garg and Philip [10] gave an $\mathcal{O}^*(3^{k-(2LP(G)-MM(G))})$ time algorithm. Randomized polynomial kernels are known for parameters $k - MM(G)$ and $k - LP(G)$ [15] and for parameter $k - (2LP(G) - MM(G))$ [14]. Our present kernelizations are not covered even by the strongest parameter $k - (2LP(G) - MM(G))$ because already $d$-quasi-forests for any $d \geq 2$ can have a vertex cover size that is arbitrarily larger than $k - (2LP(G) - MM(G))$: Consider, for example, a disjoint union of cliques $K_4$ with four vertices each, where $2LP(K_4) - MM(K_4) = 2$ but vertex cover size is three per component.

Regarding lower bounds for kernelization (all assuming $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$), it is of course well known that there are no polynomial kernels for VERTEX COVER when parameterized by width parameters like treewidth, pathwidth, or treedepth (cf. [2]). Lower bounds similar to the one for modulators to mock forests by Fomin and Strømme [9] were already obtained by Cygan et al. [7] (modulators to treewidth at most two) and Jansen [12] (modulators to outerplanar graphs). Bodlaender et al. [3] showed that there is no polynomial kernelization in terms of the vertex deletion distance to a single clique, which is stronger than distance to cluster or perfect graphs for example. Majumdar et al. [16] ruled out kernels of size $\mathcal{O}(|X|^{r-\epsilon})$ when $X$ is a modulator to a cluster graph with cliques of size bounded by $r$.

Due to space limitations several proofs are deferred to the full version.

## 2 Preliminaries and notation

**Graphs.** We use standard notation mostly following Diestel [8]. Let $G = (V, E)$ be a graph. For a set $X \subseteq V$, let $N_G(X)$ denote the neighborhood of $X$ in $G$, i.e., $N_G(X) = \{v \in V \setminus X \mid \exists u \in X \colon \{u, v\} \in E\}$ and let $N_G[X]$ denote the neighborhood of $X$ in $G$ including $X$, i.e., $N_G[X] = N_G(X) \cup X$. We omit the subscript whenever the underlying graph is clear from the context. Furthermore, we use $G - X$ as shorthand for $G[V \setminus X]$. For a graph $G$ we denote by $\mathrm{vc}(G)$ the vertex cover number of $G$ and by $\alpha(G)$ the independence number of $G$. Let $Y \subseteq V$, we call $Y$ a *blocking set* of $G$, if deleting the vertex set $Y$ from the graph $G$ decreases the size of a maximum independent set, hence if $\alpha(G) > \alpha(G - Y)$. A blocking set $Y$ is *minimal*, if no proper subset $Y' \subsetneq Y$ of $Y$ is a blocking set of $G$. We denote by $K_n$ the clique of size $n$.

**Linear Programming.** We denote the linear program relaxation for VERTEX COVER resp. INDEPENDENT SET for a graph $G = (V, E)$ by $\mathrm{LP}_{\mathrm{VC}}(G)$ resp. $\mathrm{LP}_{\mathrm{IS}}(G)$. Recall that $\mathrm{LP}_{\mathrm{VC}}(G) = \min\{\sum_{v \in V} x_v \mid \forall \{u, v\} \in E \colon x_u + x_v \geq 1 \wedge \forall v \in V \colon 0 \leq x_v \leq 1\}$ and $\mathrm{LP}_{\mathrm{IS}}(G) = \max\{\sum_{v \in V} x_v \mid \forall \{u, v\} \in E \colon x_u + x_v \leq 1 \wedge \forall v \in V \colon 0 \leq x_v \leq 1\}$. A *feasible solution* to one of the above linear program relaxations is an assignment to the variables $x_v$ for all vertices $v \in V$ which satisfies the conditions of the linear program. An optimum solution to $\mathrm{LP}_{\mathrm{VC}}(G)$ resp. $\mathrm{LP}_{\mathrm{IS}}(G)$ is a feasible solution $x$ which minimizes resp. maximizes the objective function value $w(x) := \sum_{v \in V} x_v$. It follows directly from the definition that $x$ is a feasible solution to $\mathrm{LP}_{\mathrm{VC}}(G)$ if and only if $x' = 1 - x$ is a feasible solution to $\mathrm{LP}_{\mathrm{IS}}(G)$; thus $w(x') = |V| - w(x)$. It is well known that there exists an optimum feasible solution $x$ to $\mathrm{LP}_{\mathrm{VC}}(G)$ with $x_v \in \{0, \frac{1}{2}, 1\}$; we call such a solution *half integral*. The same is, of course, true for $\mathrm{LP}_{\mathrm{IS}}(G)$. Given a half integral solution $x$ (to $\mathrm{LP}_{\mathrm{VC}}(G)$ or $\mathrm{LP}_{\mathrm{IS}}(G)$), we define $V_i^x = \{v \in V \mid x_v = i\}$ for each $i \in \{0, \frac{1}{2}, 1\}$. Note that if $x$ is an optimum half integral solution to $\mathrm{LP}_{\mathrm{VC}}(G)$, then it holds that $N(V_0^x) = V_1^x$, whereas, it holds that $N(V_1^x) = V_0^x$, when $x$ is an optimum half integral solution to $\mathrm{LP}_{\mathrm{IS}}(G)$. We omit the subscript $x$, when the solution $x$ is clear from the context.

## 3 Vertex Cover parameterized by a modulator to a $d$-quasi-forest

In this section we present a polynomial kernel for VERTEX COVER parameterized by a modulator to a $d$-quasi-forest. More precisely, we develop a polynomial kernel for INDEPENDENT SET parameterized by a modulator to a $d$-quasi-forest which, by the relation between these two problems, directly yields a polynomial kernel for VERTEX COVER parameterized by a modulator to a $d$-quasi-forest.

Consider an instance $(G, X, k)$ of the problem, which asks whether graph $G$, with $G - X$ is a $d$-quasi-forest, has an independent set of size $k$. Like Fomin and Strømme [9], we reduce the input instance $(G, X, k)$ until the $d$-quasi-forest $G - X$ has at most polynomially many connected components in terms of $|X|$; see Rule 1. By adding for each component of the $d$-quasi-forest a feedback vertex set of size $d$ to the modulator $X$, we polynomially increase the size of the modulator $X$. The resulting modulator is a feedback vertex set, hence we can apply the polynomial kernelization for INDEPENDENT SET parameterized by a modulator to a feedback vertex set from Jansen and Bodlaender [13].

Let $(G, X, k)$ be an instance of INDEPENDENT SET parameterized by a modulator to a $d$-quasi-forest. Since $d$ is a constant we can compute in polynomial time a maximum independent set in $G - X$. Choosing some vertices from the set $X$ to be in an independent set will prevent some vertices in $G - X$ to be part of the same independent set; thus it may be that we can add less than $\alpha(G - X)$ vertices from $G - X$ to an independent set that contains some vertices of $X$. To measure this difference, we use the term of conflicts introduced by Jansen and Bodlaender [13]. Our definition is more general in order to use it also for modulators to $d$-quasi-bipartite resp. $d$-quasi-integral graphs.

▶ **Definition 1** (Conflicts). Let $G = (V, E)$ be a graph and $X \subseteq V$ be a subset of $V$, such that we can compute a maximum independent set in $G - X$ in polynomial time. Let $F$ be a subgraph of $G - X$ and let $X' \subseteq X$. We define the number of conflicts on $F$ which are induced by $X'$ as $\textsc{Conf}_F(X') := \alpha(F) - \alpha(F - N(X'))$.

Now we can state our reduction rule, which deletes some components of the $d$-quasi-forest $G-X$. More precisely, we delete components $H$ of which we know that there exists a maximum independent set in $G$ that contains a maximum independent set of the component $H$.

**Rule 1:** If there exists a connected component $H$ of $G - X$ such that for all independent sets $X_I \subseteq X$ of size at most $d+2$ with $\textsc{Conf}_H(X_I) > 0$ it holds that $\textsc{Conf}_{G-H-X}(X_I) \geq |X|$, then delete $H$ from $G$ and reduce $k$ by $\alpha(H)$.

The proof of safeness will be given in the sequel. In particular, we delete connected components that have no conflicts. The goal of Rule 1 is to delete connected components of the $d$-quasi-forest $G - X$ such that we can bound the number of connected components by a polynomial in the size of $X$. Thus, if we cannot apply this reduction rule any more we should be able to find a good bound for the number of connected components in the $d$-quasi-forest $G - X$. The following lemma yields such a bound.

▶ **Lemma 2.** *Let $(G, X, k)$ be an instance of* INDEPENDENT SET *parameterized by a modulator to a $d$-quasi-forest where Rule 1 is not applicable. Then the number of connected components in $G - X$ is at most $|X|^{d+3}$.*

**Proof.** Let $H$ be a connected component of the $d$-quasi-forest $G - X$. Since Rule 1 is not applicable, there exists an independent set $X_I \subseteq X$ of size at most $d + 2$ such that $\textsc{Conf}_H(X_I) > 0$ and $\textsc{Conf}_{G-H-X}(X_I) < |X|$; otherwise Rule 1 would delete $H$ (or another connected component with the same properties).

Observe, that there are at most $|X|$ connected components of the $d$-quasi-forest $G - X$ that have a conflict with an independent set $X_I \subseteq X$, when $X_I$ is the reason that we cannot apply Rule 1 to one of these connected components: Assume for contradiction that there are $p > |X|$ connected components $H_1, H_2, \ldots, H_p$ of the $d$-quasi-forest $G - X$ that have a conflict with the same independent set $X_I \subseteq X$ of size at most $d + 2$; therefore it holds that

$\textsc{Conf}_{H_i}(X_I) > 0$ for all $i \in \{1, 2, \ldots, p\}$. But now, for all $i \in \{1, 2, \ldots, p\}$

$$\textsc{Conf}_{G-H_i-X}(X_I) \geq \sum_{\substack{j=1 \\ j \neq i}}^{p} \textsc{Conf}_{H_j}(X_I) \geq p - 1 \geq |X|,$$

where the first inequality corresponds to summing over some connected components of $G - H_i - X$. Thus, $X_I$ could not be the reason why the connected components $H_1, H_2, \ldots, H_p$ are not reduced during Rule 1.

This leads to the claimed bound of at most $\binom{|X|}{\leq d+2} \cdot |X| \leq |X|^{d+3}$ connected components in $G - X$, because for every independent set $X_I \subseteq X$ of size at most $d + 2$ there are at most $|X|$ connected components for which $X_I$ is the reason that we cannot apply Rule 1. ◀

It remains to show that Rule 1 is safe; i.e. that there exists a solution for $(G, X, k)$ if and only if there exists a solution for $(G', X, k')$, where $G' = G - H$, $k' = k - \alpha(H)$ and $H$ is the connected component of $G - X$ we delete during Rule 1. The main ingredient for this is to prove that any minimal blocking set has size at most $d + 2$ (Lemma 6). To bound the size of minimal blocking sets we need the existence of a half integral solution $x$ to $\text{LP}_{\text{IS}}(G - Y)$ for which *every* maximum independent set $I$ in $G - Y$ fulfills $V_1 \subseteq I \subseteq V_{\frac{1}{2}} \cup V_1$. This is similar to the result of Nemhauser and Trotter [17] and other results about the connection between maximum independent sets (resp. minimum vertex covers) and their fractional LP solutions [1, 4, 6, 11].

▶ **Lemma 3.** *Let $G = (V, E)$ be an undirected graph. There exists an optimum half integral solution $x \in \{0, \frac{1}{2}, 1\}^{|V|}$ to $\text{LP}_{\text{IS}}(G)$ such that for all maximum independent sets $I$ in $G$ it holds that $V_1^x \subseteq I \subseteq V \setminus V_0^x$.*

**Proof.** Let $x \in \{0, \frac{1}{2}, 1\}^{|V|}$ be an optimum half integral solution to $\text{LP}_{\text{IS}}(G)$, such that $V_{\frac{1}{2}}^x$ is maximal; this means, that there exists no optimum half integral solution $x' \neq x$ to $\text{LP}_{\text{IS}}(G)$ such that $V_{\frac{1}{2}}^x \subsetneq V_{\frac{1}{2}}^{x'}$. We will show that every independent set $I$ in $G$ with $V_1^x \not\subseteq I$ or $V_0^x \cap I \neq \emptyset$ is not a maximum independent set in $G$.

First, we observe that for all subsets $V_0' \subseteq V_0^x$ it must hold that the size of the neighborhood of $V_0'$ in $V_1^x$ is larger than the size of $V_0'$, i.e. $|V_1^x \cap N(V_0')| > |V_0'|$; if this is not the case, then we can construct an optimum half integral solution $x'$ to $\text{LP}_{\text{IS}}(G)$ with $V_{\frac{1}{2}}^x \subsetneq V_{\frac{1}{2}}^{x'}$ (which contradicts the fact that $V_{\frac{1}{2}}^x$ is maximal), by assigning a value of $\frac{1}{2}$ to all vertices in $(V_1^x \cap N(V_0')) \cup V_0'$. Obviously, it holds that $V_{\frac{1}{2}}^x \subsetneq V_{\frac{1}{2}}^{x'}$ and that

$$w(x') = w(x) - |V_1^x \cap N(V_0')| + \frac{1}{2}(|V_1^x \cap N(V_0')| + |V_0'|) \geq w(x).$$

In order to show that $x'$ is indeed a feasible solution to $\text{LP}_{\text{IS}}(G)$, it suffices to consider edges $\{u, v\}$ of $G$ that have at least one endpoint in $V_0'$, say $v \in V_0'$, because these are the only vertices for which we increase the value of the half integral solution $x$ to obtain $x'$. Since $x'_v = \frac{1}{2}$, the constraint $x'_u + x'_v \leq 1$ can only be violated if $x'_u = 1$. But then $x_u = 1$ must hold since the only changed values are $\frac{1}{2}$ in $x'$. This of course means that $u \in V_1^x \cap N(V_0')$ and $x'_u = \frac{1}{2}$; a contradiction.

Now, we assume that there exists a maximum independent set $I$ that contains a vertex of the set $V_0^x$. Let $V_0' = V_0^x \cap I \neq \emptyset$. We will show that deleting the set $V_0'$ from the independent set $I$ and adding the set $N(V_0') \cap V_1^x$ to the independent set $I$ leads to a larger independent $I'$ of $G$, i.e. $I' = I \setminus V_0' \cup (N(V_0') \cap V_1^x)$. First we show that $I'$ has larger cardinality than $I$. Since $I$ is an independent set, we know that $(N(V_0') \cap V_1^x) \cap I = \emptyset$ and hence that the

cardinality of $I'$ is $|I| - |V_0'| + |N(V_0') \cap V_1^x|$. From the above observation, we know that $|N(V_0') \cap V_1^x| > |V_0'|$ and it follows that $I'$ has larger cardinality than $I$. To prove that $I'$ is an independent set in $G$, it is enough to show that any vertex $v \in N(V_0') \cap V_1^x$ has no neighbor in $I'$; this holds because $V_1^x$ is an independent set, $N(V_1^x) \subseteq V_0^x$ and $V_0^x \cap I' = \emptyset$. Thus, $I'$ is an independent set which has larger cardinality than $I$; this contradicts the assumption that $I$ is a maximum independent set.

It remains to show that there exists no maximum independent set $I$ in $G$ with $V_1^x \nsubseteq I \subseteq V_1^x \cup V_{\frac{1}{2}}^x$. Let $v \in V_1^x \setminus I$. Since $I$ is a maximum independent set, there exists a vertex $w \in N(V_1^x) \cap I$ (otherwise $I \cup \{v\}$ would be a larger independent set in $G$). But $N(V_1^x) \subseteq V_0^x$ and hence $w \in V_0^x \cap I$, which contradicts the assumption that $I \subseteq V_1^x \cup V_{\frac{1}{2}}^x$. ◀

Using the above lemma, we can show that every minimal blocking set in a $d$-quasi-forest has size at most $d + 2$. This generalizes the result of Fomin and Stromme [9], who showed that a minimal blocking set in a pseudoforest has size at most three. Furthermore, we can show that this bound is tight.

▶ **Theorem 4.** *Minimal blocking sets have a tight upper bound of $d + 2$ in $d$-quasi-forests.*

The crucial part of Theorem 4 is to prove the upper bound.

▶ **Lemma 5.** *Let $H = (V, E)$ be a $d$-quasi-forest and let $Z$ be a feedback vertex set in $H$ of size at most $d$. Then it holds that a minimal blocking set $Y$ in the $d$-quasi-forest $H$ has size at most $|Z| + 2 \leq d + 2$.*

**Proof.** We consider an optimum half integral solution $x$ to $\mathrm{LP}_{\mathrm{IS}}(H - Y)$ which fulfills the properties of Lemma 3; let $V_i = \{v \in V(H - Y) \mid x_v = i\}$ for $i \in \{0, \frac{1}{2}, 1\}$. We know that every maximum independent set $I$ of $H - Y$ contains the set $V_1$ and no vertex of the set $V_0$ (because $x$ fulfills the properties of Lemma 3).

Observe that for all vertices $y \in Y$ it holds that $\alpha(H - (Y \setminus \{y\})) = \alpha(H)$; otherwise, the set $Y$ would not be a minimal blocking set. Furthermore, from the above observation it follows that $\alpha(H) = \alpha(H - Y) + 1$, because

$$\alpha(H - Y) < \alpha(H) = \alpha(H - (Y \setminus \{y\})) \leq \alpha(H - Y) + 1 \text{ for all } y \in Y.$$

The key observation of our proof is that $N_H(Y) \subseteq V_0 \cup V_{\frac{1}{2}}$; this follows from the fact that $Y$ is minimal: As observed above, we know that $\alpha(H - (Y \setminus \{y\})) = \alpha(H)$. Thus, for all vertices $y \in Y$ there exists a maximum independent set $I_y$ in $H$ that contains the vertex $y$ and no other vertex from the set $Y$. Consider the sets $I_y' = I_y \setminus \{y\}$ for all vertices $y \in Y$. Obviously, the sets $I_y'$ are independent sets in $H - Y$ for all vertices $y \in Y$, because $y \in Y$ is the only vertex of the set $Y$ that is contained in $I_y$. Furthermore, we know that the sets $I_y'$ are maximum independent sets in $H - Y$ because

$$|I_y'| + 1 = |I_y| = \alpha(H) = \alpha(H - Y) + 1.$$

The fact that $I_y'$ is a maximum independent set for all vertices $y \in Y$ implies that $V_1 \subseteq I_y' = I_y \setminus \{y\} \subseteq I_y$ (by the choice of the solution $x$ to $\mathrm{LP}_{\mathrm{IS}}(H - Y)$). Thus, for all vertices $y \in Y$ it holds that $V_1 \subseteq I_y$ and therefore that $N_H(I_y) \cap V_1 = \emptyset$ which implies that $N_H(\{y\}) \cap V_1 = \emptyset$ (because $V_1 \cup \{y\} \subseteq I_y$). Since this holds for all vertices $y \in Y$ it follows that $N_H(Y) \cap V_1 = \emptyset$, hence $N_H(Y) \subseteq V_0 \cup V_{\frac{1}{2}}$.

To bound the size of $Y$ we try to find an upper bound for the size of a maximum independent set in $H - Y$ and a lower bound for the size of a maximum independent set in $H$. An obvious upper bound for the size of a maximum independent set in $H - Y$ is the

optimum value of $\mathrm{LP}_{\mathrm{IS}}(H-Y)$ which is equal to $|V_1| + \frac{1}{2}|V_{\frac{1}{2}}|$. This leads to an upper bound for $\alpha(H-Y)$:

$$\begin{aligned}\alpha(H-Y) \leq w(x) &= |V_1| + \frac{1}{2}|V_{\frac{1}{2}}| = |V_1| + \frac{1}{2}|H - V_0 - V_1 - Y| \\ &= |V_1| + \frac{|H-V_0-V_1|}{2} - \frac{|Y|}{2}, \end{aligned} \tag{1}$$

because $V_0 \cup V_1 \subseteq H - Y$.

Next, we try to find a lower bound for the size of a maximum independent set in $H$. We will construct an independent set $I_H$ in $H$ and the size of this independent set is a lower bound for the size of a maximum independent set in $H$. First of all, we add all vertices from the independent set $V_1$ to $I_H$; this will prevent every vertex from $N_H(V_1)$ to be part of the independent set $I_H$. Now, we can extend the independent $V_1$ by an independent set in $H - N_H[V_1]$. First, observe that $N_H[V_1] \cap Y = \emptyset$, because $V_1 \subseteq (H-Y)$ and $N_H(Y) \cap V_1 = \emptyset$. From this follows that $H - N_H[V_1] = H - V_0 - V_1$, because $N(V_1) = V_0$. Instead of adding an independent set of $H - V_0 - V_1$ to $I_H$, we add a maximum independent set $I_F$ of the forest $H - V_0 - V_1 - Z$ to $I_H$; such an independent set $I_F$ has size at least $\frac{1}{2}|H - V_0 - V_1 - Z|$. This leads to the following lower bound for $\alpha(H)$:

$$\begin{aligned}\alpha(H) \geq |I_H| &= |V_1| + |I_F| \geq |V_1| + \frac{|H-V_0-V_1-Z|}{2} \\ &= |V_1| + \frac{|H-V_0-V_1|}{2} - \frac{|Z \setminus (V_0 \cup V_1)|}{2} \geq |V_1| + \frac{|H-V_0-V_1|}{2} - \frac{|Z|}{2} \end{aligned} \tag{2}$$

Using the equation $\alpha(H) = \alpha(H-Y) + 1$ together with the upper bound for $\alpha(H-Y)$ and the lower bound for $\alpha(H)$ leads to the requested upper bound for the size of $Y$:

$$\begin{aligned}|V_1| + \frac{|H-V_0-V_1|}{2} - \frac{|Z|}{2} \overset{(2)}{\leq} \alpha(H) = \alpha(H-Y) + 1 &\overset{(1)}{\leq} |V_1| + \frac{|H-V_0-V_1|}{2} - \frac{|Y|}{2} + 1 \\ \implies |Y| &\leq |Z| + 2. \end{aligned}$$

◀

We showed that every minimal blocking set in a $d$-quasi-forest has size at most $d+2$. To proof Theorem 4 it remains to show that the bound is tight:

**Proof of Theorem 4.** We show the remaining part of Theorem 4, namely that the bound is tight. Consider the connected graph $H = K_{d+2}$. It holds that $H$ is a $d$-quasi-forest, because any $d$ vertices from $H$ are a feedback vertex set. It holds that the size of a maximum independent set in a clique is 1, hence $\alpha(H - Y') = 1$ for all subsets $Y' \subsetneq V(H)$. Therefore, $Y = V(H)$ is the only, and hence a minimal, blocking set in $H$. ◀

Recall that Rule 1 considers the conflicts that a connected component $H$ of the $d$-quasi-forest $G - X$ has with subsets of $X$. So far, we only talked about the size of minimal blocking sets instead of the size of minimal subset of $X$ that leads to a conflict. Since every independent set $X_I \subseteq X$ that has a conflict with $H$, has some neighbors in this component, we know that these vertices are a blocking set of $H$. Using Lemma 5 we can argue that only a subset of at most $d+2$ vertices (of the neighborhood of $X_I$ in $H$) is important. Like Jansen and Bodlaender [13] resp. Fomin and Strømme [9] we show how a smaller subset of $V(H) \cap N(X_I)$ leads to a smaller subset of $X_I$ that has a conflict with the connected component $H$.

▶ **Lemma 6.** *Let $(G, X, k)$ be an instance of* INDEPENDENT SET *parameterized by a modulator to a $d$-quasi-forest. Let $H$ be a connected component of $G - X$ and let $X_I \subseteq X$ be an independent set in $G$. If $\textsc{Conf}_H(X_I) > 0$, then there exists a set $X' \subseteq X_I$ of size at most $d+2$ such that $\textsc{Conf}_H(X') > 0$.*

We showed that if a component $H$ of $G - X$ has a conflict with a subset $X' \subseteq X$ of the modulator, then there always exists a set $X'' \subseteq X'$ of size at most $d + 2$ that has a conflict with the component $H$. Knowing this, we can show that Rule 1 is safe using Lemma 6 as well as some observations that where already used in earlier work [9, 13].

▶ **Lemma 7.** *Rule 1 is safe; let $(G, X, k)$ be the instance before applying Rule 1 and let $(G', X, k')$ be the reduced instance. Then there exists a solution for $(G, X, k)$ if and only if there exists a solution for $(G', X, k')$.*

Recall that if we have an instance $(G, X, k)$ of INDEPENDENT SET parameterized by a modulator to a $d$-quasi-forest where Rule 1 is not applicable then $G - X$ has at most $|X|^{d+3}$ connected components. To apply the kernelization for INDEPENDENT SET parameterized by a modulator to a forest from Jansen and Bodlaender [13], we have to add vertices from each connected component of the $d$-quasi-forest $G - X$ to the modulator $X$, getting a set $X' \supseteq X$, such that the connected components of $G - X'$ are trees.

We know that every connected component of the $d$-quasi-forest $G - X$ has a feedback vertex set of size at most $d$, which we can find in polynomial time, since $d$ is a constant. Let $Z \subseteq V(G - X)$ be the union of these feedback vertex sets; it holds that $|Z| \leq d \cdot |X|^{d+3}$. Now, the instance $(G', X', k')$ with $G' = G$, $X' = X \cup Z$ and $k' = k$ is an instance of INDEPENDENT SET parameterized by a modulator to feedback vertex set. Obviously, it holds that $(G, X, k)$ has a solution if and only if $(G', X', k')$ has a solution. Applying the following result of Jansen and Bodlaender [13] will finish our kernelization.

▶ **Proposition 8** ([13, Theorem 2])**.** INDEPENDENT SET *parameterized by a modulator to a* FEEDBACK VERTEX SET *has a kernel with a cubic number of vertices: there is a polynomial-time algorithm that transforms an instance $(G, X, k)$ into an equivalent instance $(G', X', k')$ such that $|X'| \leq 2|X|$ and $|V(G')| \leq 2|X| + 28|X|^2 + 56|X|^3$.*

▶ **Theorem 9.** INDEPENDENT SET *parameterized by a modulator to a $d$-quasi-forest admits a kernel with $\mathcal{O}(d^3|X|^{3d+9})$ vertices.*

▶ **Corollary 10.** VERTEX COVER *parameterized by a modulator to a $d$-quasi-forest admits a kernel with $\mathcal{O}(d^3|X|^{3d+9})$ vertices.*

## 4 Two other graph classes with small blocking sets

In this section we consider VERTEX COVER parameterized by a modulator to a $d$-quasi-bipartite graph and by a modulator to a $d$-quasi-integral graph. As in the case of VERTEX COVER parameterized by a modulator to a $d$-quasi-forest, we prove that the size of a minimal blocking set is bounded linearly in $d$ to reduce the number of connected components in the $d$-quasi-bipartite graph resp. the $d$-quasi-integral graph. Having only polynomial in the modulator many connected components we show that we can apply the randomized polynomial kernelizations for VERTEX COVER parameterized by a modulator to a bipartite graph, resp. VERTEX COVER above $\mathrm{LP}_{\mathrm{VC}}$.

The proof that there exists a kernelization for VERTEX COVER parameterized by a modulator to a $d$-quasi-bipartite graph works just the same as the kernelization for VERTEX COVER parameterized by a modulator to a $d$-quasi-forest, except for the last step. Here we apply the kernelization of VERTEX COVER parameterized by a modulator to a bipartite graph.

▶ **Corollary 11.** *In a $d$-quasi-bipartite graph the size of a minimal blocking set has a tight upper bound of $d + 2$.*

▶ **Corollary 12.** VERTEX COVER *parameterized by a modulator to a $d$-quasi-bipartite graph admits a randomized polynomial kernel.*

In contrast to $d$-quasi-forests and $d$-quasi-bipartite graphs, where every minimal blocking set is of size at most $d+2$, $d$-quasi-integral graphs have minimal blocking sets of size up to $2d+2$. Nevertheless, all proofs, to show that there exists a polynomial kernel, still work, because we only need the existence of a small blocking set.

▶ **Lemma 13.** *Let $H=(V,E)$ be a $d$-quasi-integral graph. Then it holds that a minimal blocking set $Y$ in the $d$-quasi-integral graph $H$ has size at most $2d+2$.*

**Proof.** Like in the proof of Lemma 5 we consider an optimum half integral solution $x$ to $\mathrm{LP}_{\mathrm{IS}}(H-Y)$ which fulfills the properties of Lemma 3. Let $V_i=\{v\in V(H-Y)\mid x_v=i\}$ for $i\in\{0,\frac{1}{2},1\}$.

Note that the upper bound $\alpha(H-Y)\overset{(1)}{\leq}|V_1|+\frac{1}{2}|H-V_0-V_1|-\frac{1}{2}|Y|$ also holds in this case, because the value of an optimum half integral solution is always a valid upper bound.

In this case the lower bound for $\alpha(H)$ works slightly differently. Instead of constructing an independent set in $H$ we construct a feasible solution to $\mathrm{LP}_{\mathrm{IS}}(H)$. We first use the fact that $H$ is a $d$-quasi-integral graph, hence $\mathrm{vc}(H)\leq\mathrm{LP}_{\mathrm{VC}}(H)+d$, which is equivalent to $\alpha(H)\geq\mathrm{LP}_{\mathrm{IS}}(H)-d$, because $\alpha(H)=|H|-\mathrm{vc}(H)$ and $|H|-\mathrm{LP}_{\mathrm{VC}}(H)=\mathrm{LP}_{\mathrm{IS}}(H)$. Now, we construct a feasible solution $x'$ to $\mathrm{LP}_{\mathrm{IS}}(H)$. First, we assign every vertex $v$ in the independent set $V_1$ the value 1 and every vertex $w$ in $N_H(V_0)$ the value 0. Like in the proof of Lemma 5, it holds that $N_H[V_1]=H-V_0-V_1$, because $N_H[V_1]\cap Y=\emptyset$. Finally, we assign the value $\frac{1}{2}$ to every vertex in $H-V_0-V_1$. Obviously, $x'$ is a feasible solution to $\mathrm{LP}_{\mathrm{IS}}(H)$. This leads to the following lower bound for $\alpha(H)$:

$$\alpha(H)\geq\mathrm{LP}_{\mathrm{IS}}(H)-d\geq|V_1|+\mathrm{LP}_{\mathrm{IS}}(H-V_0-V_1)-d\geq|V_1|+\frac{|H-V_0-V_1|}{2}-d \tag{3}$$

Again, using the equation $\alpha(H)=\alpha(H-Y)+1$ together with the upper bound for $\alpha(H-Y)$ and the lower bound for $\alpha(H)$ leads to the requested bound for the size of $Y$:

$$|V_1|+\frac{|H-V_0-V_1|}{2}-d\overset{(3)}{\leq}\alpha(H)=\alpha(H-Y)+1\overset{(1)}{\leq}|V_1|+\frac{|H-V_0-V_1|}{2}-\frac{|Y|}{2}+1$$
$$\implies |Y|\leq 2d+2.$$ ◀

▶ **Theorem 14.** *In a $d$-quasi-integral graph the size of a minimal blocking set has a tight upper bound of $2d+2$.*

▶ **Theorem 15.** VERTEX COVER *parameterized by a modulator to a $d$-quasi-integral graph admits a randomized polynomial kernel.*

**Proof sketch.** Let $(G,X,k)$ be an instance of VERTEX COVER parameterized by a modulator to a $d$-quasi-integral graph. We can obtain in polynomial time an equivalent instance $(\widetilde{G},X,\widetilde{k})$ of VERTEX COVER parameterized by a modulator to a $d$-quasi-integral graphs by applying Rule 1 exhaustively, but instead of decreasing $k$ by $\alpha(H)$ we decrease $k$ by $\mathrm{vc}(H)$. Now, $\widetilde{G}-X$ has at most $|X|^{2d+3}$ connected components, because Lemma 2 uses only the fact that a minimal blocking set in $G-X$ has size at most $d+2$ (we only have to replace $d+2$ by $2d+2$). These instances are equivalent, since we only delete connected components $H$ (during Rule 1) of which we know that there exists a minimum vertex cover in $G$ that contains a minimum vertex cover of $H$. We can assume that $\mathrm{vc}(\widetilde{G}-X)+|X|>\widetilde{k}$. If this is not the case, then we can compute in polynomial time a vertex cover in $\widetilde{G}-X$ of size $\mathrm{vc}(\widetilde{G}-X)$ which together with the set $X$ is a vertex cover in $\widetilde{G}$ of size at most $\widetilde{k}$.

Finally, we apply the kernelization algorithm for VERTEX COVER above $\mathrm{LP}_{\mathrm{VC}}$ to the instance $(\widetilde{G}, \widetilde{k})$ and obtain an instance $(G', k')$ in polynomial time. Note that we can bound the parameter $\widetilde{k} - \mathrm{LP}_{\mathrm{VC}}(\widetilde{G})$ by a polynomial in the size of $X$ as follows:

$$\begin{aligned}
\widetilde{k} - \mathrm{LP}_{\mathrm{VC}}(\widetilde{G}) &\leq \widetilde{k} - \mathrm{LP}_{\mathrm{VC}}(\widetilde{G} - X) \\
&= \widetilde{k} - \sum_{H \text{ c.c of } \widetilde{G}-X} \mathrm{LP}_{\mathrm{VC}}(H) \\
&\leq \widetilde{k} - \sum_{H \text{ c.c of } \widetilde{G}-X} (\mathrm{vc}(H) - d) \\
&= \widetilde{k} + |X|^{2d+3}d - \mathrm{vc}(\widetilde{G} - X) \\
&\leq |X| + |X|^{2d+3}d
\end{aligned}$$

Since $(G', k')$ is polynomially bounded in the size of $\widetilde{k} - \mathrm{LP}_{\mathrm{VC}}(\widetilde{G})$, which is bounded by a polynomial in the size of $|X|$, we know that the instance $(G', X' = V(G'), k')$ is an equivalent instance of VERTEX COVER parameterized by a modulator to a $d$-quasi-integral graph. ◀

## 5 Conclusion

Starting from the work of Fomin and Strømme [9] we have presented new results for polynomial kernels for VERTEX COVER subject to structural parameters. Our results for modulators to $d$-quasi-forests show that bounds on the feedback vertex set size are more meaningful for kernelization than the treewidth of $G - X$ (recalling that there is a lower bound for treewidth of $G - X$ being at most two). By extending our kernelization to work for modulators to ($d$-quasi-bipartite and) $d$-quasi-integral graphs, we have encompassed existing kernelizations for parameterization by distance to forests [13], distance to max degree two [16] (both previously subsumed by), distance to pseudoforests [9], and parameterization above fractional optimum [15]. It would be interesting whether there is a single positive result that encompasses all parameterizations with polynomial kernels.

To obtain our results we have established tight bounds for the size of minimal blocking sets in $d$-quasi-forests, $d$-quasi-bipartite graphs, and $d$-quasi-integral graphs. Tightness comes from the fact that cliques of size $d + 2$ respectively $2d + 2$ are contained in these classes. The presence of these cliques also implies lower bounds ruling out kernels of size $\mathcal{O}(|X|^{r-\epsilon})$, assuming $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$, when $r = r(d)$ is the maximum size of minimal blocking sets as a consequence of a lower bound by Majumdar et al. [16]. It would be interesting whether there are matching upper bounds for kernelization, e.g., whether the kernelization of Jansen and Bodlaender [13] for modulators to forests can be improved to size $\mathcal{O}(|X|^2)$.

## References


1. Faisal N. Abu-Khzam, Michael R. Fellows, Michael A. Langston, and W. Henry Suters. Crown structures for vertex cover kernelization. *Theory Comput. Syst.*, 41(3):411–430, 2007. doi:10.1007/s00224-007-1328-0.
2. Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.
3. Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014. doi:10.1137/120880240.

4 Endre Boros, Martin Charles Golumbic, and Vadim E. Levit. On the number of vertices belonging to all maximum stable sets of a graph. *Discrete Applied Mathematics*, 124(1-3):17–25, 2002. doi:10.1016/S0166-218X(01)00327-4.

5 Marin Bougeret and Ignasi Sau. How much does a treedepth modulator help to obtain polynomial kernels beyond sparse graphs? *CoRR*, abs/1609.08095, 2016. arXiv:1609.08095.

6 Miroslav Chlebík and Janka Chlebíková. Crown reductions for the minimum weighted vertex cover problem. *Discrete Applied Mathematics*, 156(3):292–312, 2008. doi:10.1016/j.dam.2007.03.026.

7 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. On the hardness of losing width. *Theory Comput. Syst.*, 54(1):73–82, 2014. doi:10.1007/s00224-013-9480-1.

8 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

9 Fedor V. Fomin and Torstein J. F. Strømme. Vertex cover structural parameterization revisited. In Pinar Heggernes, editor, *Graph-Theoretic Concepts in Computer Science - 42nd International Workshop, WG 2016, Istanbul, Turkey, June 22-24, 2016, Revised Selected Papers*, volume 9941 of *Lecture Notes in Computer Science*, pages 171–182, 2016. doi:10.1007/978-3-662-53536-3_15.

10 Shivam Garg and Geevarghese Philip. Raising the bar for vertex cover: Fixed-parameter tractability above A higher guarantee. 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 1152–1166. SIAM, 2016. doi:10.1137/1.9781611974331.ch80.

11 Peter L. Hammer, Pierre Hansen, and Bruno Simeone. Vertices belonging to all or to no maximum stable sets of a graph. *SIAM Journal on Algebraic Discrete Methods*, 3(4):511–522, 1982.

12 Bart M. P. Jansen. *The power of data reduction: Kernels for fundamental graph problems*. PhD thesis, Utrecht University, 2013.

13 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.

14 Stefan Kratsch. A randomized polynomial kernelization for vertex cover with a smaller parameter. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 59:1–59:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.59.

15 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 450–459. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.46.

16 Diptapriyo Majumdar, Venkatesh Raman, and Saket Saurabh. Kernels for structural parameterizations of vertex cover - case of small degree modulators. In Thore Husfeldt and Iyad A. Kanj, editors, *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, volume 43 of *LIPIcs*, pages 331–342. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.IPEC.2015.331.

17 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.

# Polynomial-Time Algorithms for the Longest Induced Path and Induced Disjoint Paths Problems on Graphs of Bounded Mim-Width*†

**Lars Jaffke‡[1], O-joung Kwon§[2], and Jan Arne Telle[3]**

1 **Department of Informatics, University of Bergen, Norway**
`lars.jaffke@uib.no`
2 **Logic and Semantics, Technische Universität Berlin, Berlin, Germany**
`ojoungkwon@gmail.com`
3 **Department of Informatics, University of Bergen, Norway**
`jan.arne.telle@uib.no`

## Abstract

We give the first polynomial-time algorithms on graphs of bounded *maximum induced matching width* (mim-width) for problems that are not locally checkable. In particular, we give $n^{\mathcal{O}(w)}$-time algorithms on graphs of mim-width at most $w$, when given a decomposition, for the following problems: Longest Induced Path, Induced Disjoint Paths and $H$-Induced Topological Minor for fixed $H$. Our results imply that the following graph classes have polynomial-time algorithms for these three problems: Interval and Bi-Interval graphs, Circular Arc, Permutation and Circular Permutation graphs, Convex graphs, $k$-Trapezoid, Circular $k$-Trapezoid, $k$-Polygon, Dilworth-$k$ and Co-$k$-Degenerate graphs for fixed $k$.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, Computations on discrete structures, G.2.2 Graph Theory, Graph algorithms

**Keywords and phrases** graph width parameters, dynamic programming, graph classes, induced paths, induced topological minors

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.21

## 1 Introduction

Ever since the definition of the tree-width of graphs emerged from the Graph Minors project of Robertson and Seymour, bounded-width structural graph decompositions have been a successful tool in designing fast algorithms for graph classes on which the corresponding width-measure is small. Over the past few decades, many more width-measures have been introduced, see e.g. [8] for an excellent survey and motivation for width-parameters of graphs. In 2012, Vatshelle [18] defined the *maximum induced matching width* (mim-width for short) which measures how easy it is to decompose a graph along vertex cuts with bounded maximum induced matching size on the bipartite graph induced by edges crossing the cut. One interesting aspect of this width-measure is that its modeling power is much stronger than

* The work was done while the authors were at Polytechnic University of Valencia, Spain.
† A full version of the paper is available at `https://arxiv.org/abs/1708.04536`.
‡ Lars Jaffke is supported by the Bergen Research Foundation (BFS).
§ O-joung Kwon is Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant DISTRUCT, agreement No. 648527).


tree-width and clique-width and many well-known and deeply studied graph classes such as Interval graphs and Permutation graphs have (linear) mim-width 1, with decompositions that can be found in polynomial time [2, 18], while their clique-width can be proportional to the square root of the number of vertices. Hence, designing an algorithm for a problem $\Pi$ that runs in XP time parameterized by mim-width yields polynomial-time algorithms for $\Pi$ on several interesting graph classes at once.

For Locally Checkable Vertex Subset and Vertex Partitioning (LC-VSVP) problems, a class introduced [17] to capture many well-studied algorithmic problems in a unified framework, Belmonte and Vatshelle [2] and Bui-Xuan et al. [3] provided XP-algorithms on graphs of bounded mim-width. LC-VSVP problems include many NP-hard problems such as Maximum Independent Set, Minimum Dominating Set, and $q$-Coloring. A common feature of these problems is that they (as the name suggests) can be checked locally: Take $q$-Coloring for example. Here, we want to determine whether there is a $q$-partition of the vertex set of an input graph such that each part induces an independent set. The latter property can be checked individually for each vertex by inspecting only its direct neighborhood.

Until now, the only problems known to be XP-time solvable on graphs of bounded mim-width were of the type LC-VSVP. It is therefore natural to ask whether similar results can be shown for problems concerning graph properties that are *not* locally checkable. In this paper, we study problems related to finding *induced* paths in graphs, namely Longest Induced Path, Induced Disjoint Paths and $H$-Induced Topological Minor. Although their 'non-induced' counterparts are more deeply studied in the literature, also these induced variants have received considerable attention. Below, we briefly survey results for exact algorithms to these three problems on graph classes studied so far.

For the first problem, Gavril [5] showed that Longest Induced Path can be solved in polynomial time for graphs without induced cycles of length at least $q$ for fixed $q$ (the running time was improved by Ishizeki et al. [9]), while Kratsch et al. [13] solved the problem on AT-free graphs in polynomial time. Kang et al. [11] recently showed that those classes have unbounded mim-width. However, graphs of bounded mim-width are not necessarily graphs without cycles of length at least $k$ or AT-free graphs. The second problem derives from the well-known Disjoint Paths problem which is solvable in $O(n^3)$ time if the number of paths $k$ is a fixed constant, as shown by Robertson and Seymour [15], while if $k$ is part of the input it is NP-complete on graphs of linear mim-width 1 (interval graphs) [14]. In contrast, Induced Disjoint Paths is NP-complete already for $k = 2$ paths [12]. In this paper we consider the number of paths $k$ as part of the input. Under this restriction Induced Disjoint Paths is NP-complete on claw-free graphs, as shown by Fiala et al. [4], while Golovach et al. [7] gave a linear-time algorithm for circular-arc graphs. For the third problem, $H$-Induced Topological Minor, we consider $H$ to be a fixed graph. This problem, and also Induced Disjoint Paths, were both shown solvable in polynomial time on chordal graphs by Belmonte et al. [1], and on AT-free graphs by Golovach et al. [6].

We show that Longest Induced Path, Induced Disjoint Paths and $H$-Induced Topological Minor for fixed $H$ can be solved in time $n^{\mathcal{O}(w)}$ given a branch decomposition of mim-width $w$. Since bounded mim-width decompositions, usually mim-width 1 or 2, can be computed in polynomial-time for all well-known graph classes having bounded mim-width [2], our results thus provide unified polynomial-time algorithms for these problems on the following classes of graphs: Interval and Bi-Interval graphs, Circular Arc, Permutation and Circular Permutation graphs, Convex graphs, $k$-Trapezoid, Circular $k$-Trapezoid, $k$-Polygon, Dilworth-$k$ and Co-$k$-Degenerate graphs for fixed $k$, all graph classes of bounded mim-width [2].

The problem of computing the mim-width of general graphs was shown to be W[1]-hard [16] and no algorithm for computing the mim-width of a graph in XP time is known. Furthermore, there is no polynomial-time constant-factor approximation for mim-width unless NP = ZPP [16].

What makes our algorithms work is an analysis of the structure induced by a solution to the problem on a cut in the branch decomposition. There are two ingredients. First, in all the problems we investigate, we are able to show that for each cut induced by an edge of the given branch decomposition of an input graph, it is sufficient to consider induced subgraphs of size at most $\mathcal{O}(w)$ as intersections of solutions and the set of edges crossing the cut, where $w$ is the mim-width of the branch decomposition. For instance, in the Longest Induced Paths problem, an induced path is a target solution. We argue that an induced path cannot cross a cut many times if there is no large induced matching between vertex sets $A$ and $B$ of the cut $(A, B)$. Such an intersection is always a disjoint union of paths. Thus, we enumerate all subgraphs of size at most $\mathcal{O}(w)$, which are disjoint unions of paths, and these will be used as indices of our table.

However, a difficulty arises if we recursively ask for a given cut and such an intersection of size at most $\mathcal{O}(w)$, whether there is an induced disjoint union of paths of certain size in the union of one part and edges crossing the cut, whose intersection on the crossing edges is the given subgraph. The reason is that there are unbounded number of vertices in each part of the cut that are not contained in the given subgraph of size $\mathcal{O}(w)$ but still have neighbors in the other part. We need to control these vertices in such a way that they do not further create an edge in the solution. We control these vertices using vertex covers of the bipartite graph induced by edges crossing the cut. Roughly speaking, if there is a valid partial solution, then there is a vertex cover of such a bipartite graph, which meets all other edges not contained in the given subgraph. The point is that there are only $n^{\mathcal{O}(w)}$ many minimal vertex covers of such a bipartite graph with maximum matching size $w$. We discuss this property in Section 2. Based on these two results, each table will consist of a subgraph of size $\mathcal{O}(w)$ and a vertex cover of the remaining part of the bipartite graph, and we check whether there is a (valid) partial solution to the problem with respect to given information. We can argue that we need to store at most $n^{\mathcal{O}(w)}$ table entries in the resulting dynamic programming scheme and that each of them can be computed in time $n^{\mathcal{O}(w)}$ as well.

The strategy for Induced Disjoint Paths is very similar to the one for Longest Induced Path. The only thing to additionally consider is that in the disjoint union of paths, which is guessed as the intersection of a partial solution and edges crossing a cut, we need to remember which path is a subpath of the path connecting a given pair. We lastly provide a one-to-many reduction from $H$-Induced Topological Minor to Induced Disjoint Paths, that runs in polynomial time, and show that it can be solved in time $n^{\mathcal{O}(w)}$. Similar reductions have been shown earlier (see e.g. [1, 6]) but we include it here for completeness.

Throughout the paper, proofs of statements marked with '★' are deferred to the full version [10].

## 2 Preliminaries

For integers $a$ and $b$ with $a \leq b$, we let $[a..b] := \{a, a+1, \ldots, b\}$ and if $a$ is positive, we define $[a] := [1..a]$. Throughout the paper, a graph $G$ on vertices $V(G)$ and edges $E(G) \subseteq \binom{V(G)}{2}$ is assumed to be finite, undirected and simple. For graphs $G$ and $H$ we say that $G$ is a *subgraph* of $H$, if $V(G) \subseteq V(H)$ and $E(G) \subseteq E(H)$ and we write $G \subseteq H$. For a vertex set $X \subseteq V(G)$, we denote by $G[X]$ the subgraph *induced* by $X$, i.e. $G[X] := (X, E(G) \cap \binom{X}{2})$. If

$H \subseteq G$ and $X \subseteq V(G)$ then we let $H[X] := H[X \cap V(H)]$. We use the shorthand $G - X$ for $G[V(G) \setminus X]$. For two (disjoint) vertex sets $X, Y \subseteq V(G)$, we denote by $G[X, Y]$ the bipartite subgraph of $G$ with bipartition $(X, Y)$ such that for $x \in X, y \in Y$, $x$ and $y$ are adjacent in $G$ if and only if they are adjacent in $G[X, Y]$. A *cut* of $G$ is a bipartition $(A, B)$ of its vertex set. For a vertex $v \in V(G)$, we denote by $N[v]$ the set of *neighbors* of $v$ in $G$, i.e. $N[v] := \{w \in V(G) \mid \{v, w\} \in E(G)\}$. A set $M$ of edges is a *matching* if no two edges in $M$ share an end vertex, and a matching $\{a_1b_1, \ldots, a_kb_k\}$ is *induced* if there are no other edges in the subgraph induced by $\{a_1, b_1, \ldots, a_k, b_k\}$. For an edge $e = \{v, w\} \in E(G)$, the operation of *contracting* $e$ is to remove the edge $e$ from $G$ and merging its endpoints $v$ and $w$.

**Branch Decompositions and Mim-Width.** A pair $(T, \mathcal{L})$ of a subcubic tree $T$ and a bijection $\mathcal{L}$ from $V(G)$ to the set of leaves of $T$ is called a *branch decomposition*. For each edge $e$ of $T$, let $T_1^e$ and $T_2^e$ be the two connected components of $T - e$, and let $(A_1^e, A_2^e)$ be the vertex bipartition of $G$ such that for each $i \in \{1, 2\}$, $A_i^e$ is the set of all vertices in $G$ mapped to leaves contained in $T_i^e$ by $\mathcal{L}$. The *mim-width of* $(T, \mathcal{L})$, denoted by $\mathrm{mimw}(T, \mathcal{L})$, is defined as $\max_{e \in E(T)} \mathrm{mim}(A_1^e)$, where for a vertex set $A \subseteq V(G)$, $\mathrm{mim}(A)$ denotes the maximum size of an induced matching in $G[A, V(G) \setminus A]$. The minimum mim-width over all branch decompositions of $G$ is called the *mim-width of* $G$. If $|V(G)| \leq 1$, then $G$ does not admit a branch decomposition, and the mim-width of $G$ is defined to be 0.

To avoid confusion, we refer to elements in $V(T)$ as *nodes* and elements in $V(G)$ as *vertices* throughout the rest of the paper. Given a branch decomposition, one can subdivide an arbitrary edge and let the newly created vertex be the root of $T$, in the following denoted by $r$. Throughout the following we assume that each branch decomposition has a root node of degree two. For two nodes $t, t' \in V(T)$, we say that $t'$ is a *descendant* of $t$ if $t$ lies on the path from $r$ to $t'$ in $T$. For $t \in V(T)$, we denote by $G_t$ the subgraph induced by all vertices that are mapped to a leaf that is a descendant of $t$, i.e. $G_t = G[X_t]$, where $X_t = \{v \in V(G) \mid \mathcal{L}^{-1}(t') = v \text{ where } t' \text{ is a descendant of } t \text{ in } T\}$. We use the shorthand '$V_t$' for '$V(G_t)$' and let $\bar{V}_t := V(G) \setminus V_t$.

The following definitions which we relate to branch decompositions of graphs will play a central role in the design of the algorithms in Section 3.

▶ **Definition 1** (Boundary). Let $G$ be a graph and $A, B \subseteq V(G)$ such that $A \cap B = \emptyset$. We let $\mathrm{bd}_B(A)$ be the set of vertices in $A$ that have a neighbor in $B$, i.e. $\mathrm{bd}_B(A) := \{v \in V(A) \mid N(v) \cap B \neq \emptyset\}$. We define $\mathrm{bd}(A) := \mathrm{bd}_{V(G) \setminus A}(A)$ and call $\mathrm{bd}(A)$ the *boundary* of $A$ in $G$.

▶ **Definition 2** (Crossing Graph). Let $G$ be a graph and $A, B \subseteq V(G)$. If $A \cap B = \emptyset$, we define the graph $G_{A,B} := G[\mathrm{bd}_B(A), \mathrm{bd}_A(B)]$ to be the *crossing graph from $A$ to $B$*.

If $(T, \mathcal{L})$ is a branch decomposition of $G$ and $t_1, t_2 \in V(T)$ such that the crossing graph $G_{V_{t_1}, V_{t_2}}$ is defined, we use the shorthand $G_{t_1,t_2} := G_{V_{t_1}, V_{t_2}}$. We use the analogous shorthand notations $G_{t_1,\bar{t_2}} := G_{V_{t_1}, \bar{V_{t_2}}}$ and $G_{\bar{t_1},t_2} := G_{\bar{V_{t_1}}, V_{t_2}}$ (whenever these graphs are defined). For the frequently arising case when we consider $G_{t,\bar{t}}$ for some $t \in V(T)$, we refer to this graph as the *crossing graph w.r.t. $t$*.

We furthermore use the following notation. Let $G$ be a graph, $v \in V(G)$ and $A \subseteq V(G)$. We denote by $N_A[v]$ the set of neighbors of $v$ in $A$, i.e. $N_A[v] := N[v] \cap A$. For $X \subseteq V(G)$, we let $N_A[X] := \bigcup_{v \in X} N_A[v]$. If $(T, \mathcal{L})$ is a branch decomposition of $G$ and $t \in V(T)$, we use the shorthand notations $N_t[X] := N_{V_t}[X]$ and $N_{\bar{t}}[X] := N_{\bar{V_t}}[X]$.

**The Minimal Vertex Covers Lemma.** Let $G$ be a graph. We now prove that given a set $A \subseteq V(G)$, the number of minimal vertex covers in $G[A, V(G) \setminus A]$ is bounded by $n^{\mathrm{mim}(A)}$.

This observation is crucial to argue that we only need to store $n^{\mathcal{O}(w)}$ entries at each node in the branch decomposition in all algorithms we design, where $w$ is the mim-width of the given branch decomposition.

▶ **Corollary 3** (Minimal Vertex Covers Lemma, ★). *Let $H$ be a bipartite graph on $n$ vertices with a bipartition $(A, B)$. The number of minimal vertex covers of $H$ is at most $n^{\mathrm{mim}(A)}$, and the set of all minimal vertex covers of $H$ can be enumerated in time $n^{\mathcal{O}(\mathrm{mim}(A))}$.*

## 3 Algorithms

In all algorithms presented in this section, we assume that we are given as input an undirected graph $G$ together with a branch decomposition $(T, \mathcal{L})$ of $G$ of mim-width $w$, rooted at a degree two vertex obtained from subdividing an arbitrary edge in $T$. We do bottom-up dynamic programming over $(T, \mathcal{L})$. To obtain our algorithms, we study the structure a solution induces across a cut in the branch decomposition and argue that the size of this structure is bounded by a function only depending on the mim-width. The table entries at each node $t \in V(T)$ are then indexed by all possible such structures and contain the value 1 if and only if the structure used as the index of this entry constitutes a solution for the respective problem. After applying the dynamic programming scheme, the solution to the problem can be obtained by inspecting the table values associated with the root of $T$.

The rest of this section is organized as follows. In Section 3.1 we present an $n^{\mathcal{O}(w)}$-time algorithm for Longest Induced Path, and in Section 3.2 we give an algorithm for Induced Disjoint Paths with the same asymptotic runtime bound. We give a polynomial-time one-to-many reduction from $H$-Induced Topological Minor (for fixed $H$) to Induced Disjoint Paths in Section 3.3, yielding an $n^{\mathcal{O}(w)}$ for the former problem as well.

### 3.1 Longest Induced Path

For a disjoint union of paths $P$, we refer to its *size* as the number of its vertices, i.e. $|P| := |V(P)|$. If $P$ has only one component, we use the terms 'size' and 'length' interchangeably. We now give an $n^{\mathcal{O}(w)}$ time algorithm for the following parameterized problem.

Longest Induced Path (LIP)/Mim-Width
**Input:** A graph $G$ with branch decomposition $(T, \mathcal{L})$ and an integer $k$
**Parameter:** $w := \mathrm{mimw}(T, \mathcal{L})$
**Question:** Does $G$ contain an induced path of length at least $k$?

Before we describe the algorithm, we observe the following. Let $G$ be a graph and $A \subseteq V(G)$ with $\mathrm{mim}(A) = w$ and let $P$ be an induced path in $G$. Then the subgraph induced by edges of $P$ in $G_{A,\bar{A}}$ and vertices incident with these edges has size linearly bounded by $w$. The following lemma provides a bound of this size.

▶ **Lemma 4.** *Let $p$ be a positive integer and let $F$ be a disjoint union of paths such that each component of $F$ contains an edge. If $|V(F)| \geq 4p$, then $F$ contains an induced matching of size at least $p$.*

**Proof.** We prove the lemma by induction on $p$. If $p = 1$, then it is clear. We may assume $p \geq 2$. Suppose $F$ contains a connected component $C$ with at most 4 vertices. Then $F - V(C)$ contains at least $4(p-1)$ vertices, and thus it contains an induced matching of size at least $p-1$ by the induction hypothesis. As $C$ contains an edge, $F$ contains an induced matching of size at least $p$. Thus, we may assume that each component of $F$ contains at least 5 vertices.

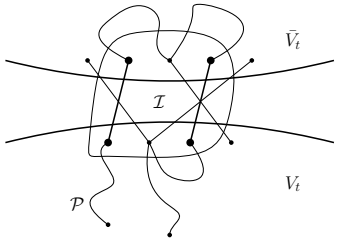



**Figure 1** The intersection of an induced path $\mathcal{P}$ with $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$, which is an induced disjoint union of paths $\mathcal{I}$. The subgraph $S$ to be used as an index for the corresponding table entry consists of the boldface vertices and edges in $\mathcal{I}$.

Let us choose a leaf $v$ of $F$, and let $v_1$ be the neighbor of $v$, and $v_2$ be the neighbor of $v_1$ other than $v$. Since each component of $F - \{v, v_1, v_2\}$ contains at least one edge, we can apply induction to conclude that $F - \{v, v_1, v_2\}$ contains an induced matching of size at least $p - 1$. Together with $vv_1$, $F$ contains an induced matching of size at least $p$. ◀

Before we give the description of the dynamic programming algorithm, we first observe how a solution $\mathcal{P}$, i.e. an induced path in $G$, interacts with the graph $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$, for some $t \in V(T)$. The intersection of $\mathcal{P}$ with $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$ is an induced disjoint union of paths which we will denote by $\mathcal{I}$ in the following. To keep the number of possible table entries bounded by $n^{\mathcal{O}(w)}$, we have to focus on the interaction of $\mathcal{I}$ with the crossing graph $G_{t,\bar{t}}$ w.r.t. $t$, in particular the intersection of $\mathcal{I}$ with its edges. Note that after removing isolated vertices, $\mathcal{I}$ induces a disjoint union of paths on $G_{t,\bar{t}}$ which throughout the following we will denote by $S$. For an illustration see Figure 1. There cannot be any additional edges crossing the cut $(V_t, \bar{V}_t)$ between vertices in $\mathcal{I}$ on opposite sides of the boundary that are not contained in $V(S)$. This property of $\mathcal{I}$ can be captured by considering a minimal vertex cover $M$ of the bipartite graph $G_{t,\bar{t}} - V(S)$. We remark that the vertices in $M$ play different roles, depending on whether they lie in $M \cap V_t$ or $M \cap \bar{V}_t$. We therefore define the following two sets.

- $M_t^{\mathrm{in}} := M \cap V_t$ is the set of vertices that must be avoided by $\mathcal{I}$.
- $M_t^{\mathrm{out}} := M \cap \bar{V}_t$ is the set of vertices that must be avoided by a partial solution (e.g. the intersection of $\mathcal{P}$ with $G[\bar{V}_t]$) to be combined with $\mathcal{I}$ to ensure that their combination does not use any edges in $G_{t,\bar{t}} - V(S)$.

Furthermore, $\mathcal{I}$ also indicates how the vertices in $S[V_t]$ that have degree one in $S$ are joined together in the graph $G_t$ (possibly outside $\mathrm{bd}(V_t)$). This gives rise to a collection of vertex pairs $Q$, which we will refer to as *pairings*, with the interpretation that $(s, t) \in Q$ if and only if there is a path from $s$ to $t$ in $\mathcal{I}[V_t]$.

The description given above immediately tells us how to index the table entries in the dynamic programming table $\mathcal{T}$ to keep track of all possible partial solutions in the graph $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$: We set the table entry $\mathcal{T}[t, (S, M, Q), i, j] = 1$, where $i \in [0..n]$ and $j \in [0..2]$, if and only if the following conditions are satisfied. For an illustration of the table indices, see Figure 2.

- **(i)** There is a set of induced paths $\mathcal{I}$ of total size $i$ in $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$ such that $\mathcal{I}$ has $j$ degree one endpoints in $G_t$.
- **(ii)** $E(\mathcal{I}) \cap E(G_{t,\bar{t}}) = E(S)$.
- **(iii)** $M$ is a minimal vertex cover of $G_{t,\bar{t}} - V(S)$ such that $V(\mathcal{I}) \cap M = \emptyset$.
- **(iv)** Let $D$ denote the vertices in $S[V_t]$ that have degree one in $S$. Let $Q = (s_1, t_1), \ldots, (s_\ell, t_\ell)$ be a partition of all but $j$ vertices of $D$ into pairs, throughout the following called a

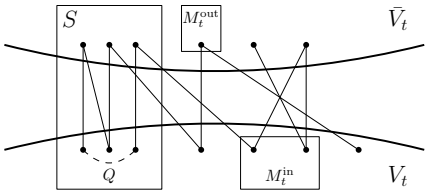



**Figure 2** A crossing graph $G_{t,\bar{t}}$ and the structures associated with the table indices of the algorithm for Longest Induced Path. Note that by (1) and (4) it follows that if the table entry corresponding to the above structures is 1, then $j = 0$: Since both degree one endpoints in $S[V_t]$ are paired, the corresponding set of induced paths $\mathcal{I}$ has zero degree one endpoints in $G[V_t]$.

*pairing*, such that if we contract all edges in $\mathcal{I} - E(G_{t,\bar{t}})$ from $\mathcal{I}$ incident with at least one vertex not in $S$ (we denote the resulting graph as $S \odot Q$) we obtain the same graph as when adding $\{s_k, t_k\}$ to $S$, for each $k \in [\ell]$.

Regarding (4), observe that $|D| = 2\ell + j$ and that there are $j$ unpaired vertices in $Q$, each of which is connected to a degree one endpoint of $\mathcal{I}$ in $G_t$. For notational convenience, we will denote by $\mathcal{T}_t$ all table entries that have the node $t \in V(T)$ as the first index.

We now show that the solution to Longest Induced Path can be obtained from a table entry corresponding to the root $r$ of $T$ and hence ensure that the information stored in $\mathcal{T}$ is sufficient.

▶ **Proposition 5 (★).** *$G$ contains an induced path of length $i$ if and only if $\mathcal{T}[r, (\emptyset, \emptyset, \emptyset), i, 2] = 1$.*

Throughout the following, we denote by $\mathcal{S}_t$ the set of all sets of induced disjoint paths in $G_{t,\bar{t}}$ on at most $4w$ vertices (which includes all possible intersections of partial solutions with $G_{t,\bar{t}}$ by Lemma 4), for $S \in \mathcal{S}_t$ by $\mathcal{M}_{t,S}$ the set of all minimal vertex covers of $G_{t,\bar{t}} - V(S)$ and by $\mathcal{Q}_{t,S}$ the set of all pairings of degree one vertices in $S[\text{bd}(V_t)]$. We now argue that the number of such entries is bounded by a polynomial in $n$ whose degree is $\mathcal{O}(w)$.

▶ **Proposition 6.** *For each $t \in V(T)$, there are at most $n^{\mathcal{O}(w)}$ table entries in $\mathcal{T}_t$ and they can be enumerated in time $n^{\mathcal{O}(w)}$.*

**Proof.** Note that each index is an element of $\mathcal{S}_t \times \mathcal{M}_{t,S} \times \mathcal{Q}_{t,S} \times [0..n] \times [0..2]$. Since the size of each maximum induced matching in $G_{t,\bar{t}}$ is at most $w$, we know by Lemma 4 that the size of each index $S$ is bounded by $4w$, so $|\mathcal{S}_t| \leq \mathcal{O}(n^{4w})$. By the Minimal Vertex Covers Lemma (Corollary 3), $|\mathcal{M}_{t,S}| \leq n^{\mathcal{O}(w)}$. Since the number of vertices in $S$ is bounded by $4w$, we know that $|\mathcal{Q}_{t,S}| \leq w^{\mathcal{O}(w)}$ and since $i \in [0..n]$ and $j \in [0..2]$, we can conclude that the number of table entries for each $t \in V(T)$ is at most $\mathcal{O}(n^{4w}) \cdot n^{\mathcal{O}(w)} \cdot w^{\mathcal{O}(w)} \cdot (n+1) \cdot 3 = n^{\mathcal{O}(w)}$. Clearly, all elements in $\mathcal{S}_t$ and $\mathcal{Q}_{t,S}$ can be enumerated in time $n^{\mathcal{O}(w)}$ and by the Minimal Vertex Covers Lemma, we know that all elements in $\mathcal{M}_{t,S}$ can be enumerated in time $n^{\mathcal{O}(w)}$ as well. The claimed time bound on the enumeration of the table indices follows. ◀

In the remainder of the proof we will describe how to fill the table entries from the leaves of $T$ to its root, asserting the correctness of the updates in the table. Together with Proposition 5, this will yield the correctness of the algorithm. The description of how the tables are filled at a leaf node are deferred to the full version [10].

**Internal nodes of $T$.** Let $t \in V(T)$ be an internal node of $T$, let $(S, M, Q) \in \mathcal{S}_t \times \mathcal{M}_{t,S} \times \mathcal{Q}_{t,S}$, let $i \in [0..n]$ and $j \in [0..2]$. We show how to compute the table entry $\mathcal{T}[t, (S, M, Q), i, j]$ from

table entries corresponding to the children $a$ and $b$ of $t$ in $T$. To do so, we have to take into account the ways in which partial solutions for $G[V_a \cup \mathrm{bd}(\bar{V_a})]$ and $G[V_b \cup \mathrm{bd}(\bar{V_b})]$ interact. We therefore try all pairs of indices $\mathfrak{I}_a = ((S_a, M_a, Q_a), i_a, j_a)$, $\mathfrak{I}_b = ((S_b, M_b, Q_b), i_b, j_b)$ and for each such pair, first check whether it is 'compatible' with $\mathfrak{I}_t$: We say that $\mathfrak{I}_a$ and $\mathfrak{I}_b$ are *compatible with* $\mathfrak{I}_t$ if and only if any partial solution $\mathcal{I}_a$ represented by $\mathfrak{I}_a$ for $G[V_a \cup \mathrm{bd}(\bar{V_a})]$ and $\mathcal{I}_b$ represented by $\mathfrak{I}_b$ for $G[V_b \cup \mathrm{bd}(\bar{V_b})]$ can be combined to a partial solution $\mathcal{I}_t$ for $G[V_t \cup \mathrm{bd}(\bar{V_t})]$ that is represented by the index $\mathfrak{I}_t$. We then set $\mathcal{T}_t[\mathfrak{I}_t] := 1$ if and only if we can find a compatible pair of indices $\mathfrak{I}_a$, $\mathfrak{I}_b$ as above such that $\mathcal{T}_a[\mathfrak{I}_a] = 1$ and $\mathcal{T}_b[\mathfrak{I}_b] = 1$.

**Step 0 (Valid Index).** We first check whether the index $\mathfrak{I}_t$ can represent a valid partial solution of $G[V_t \cup \mathrm{bd}(\bar{V_t})]$. The definition of the table entries requires that $S \odot Q$ is a disjoint union of paths, so if $S \odot Q$ is not a disjoint union of paths, we set $\mathcal{T}_t[\mathfrak{I}_t] := 0$ and skip the remaining steps. In general, the number of degree one vertices in $V(S \odot Q) \cap V_t$ has to be equal to $j$ and we can proceed as described in Steps 1-4, except for the following special cases, the details of which can be found in the full version [10].

**Special Case 1 ($j = 2$, $V(S \odot Q) \cap V_t$ has 0 deg. 1 vertices).**
**Special Case 2 ($j = 2$, $V(S \odot Q) \cap V_t$ has 2 deg. 1 vertices in same component).**
**Special Case 3 ($j = 0$ and $S = \emptyset$).**

**Step 1 (Induced disjoint unions of paths).** We now check whether $S_a$ and $S_b$ are compatible with $S$. We have to ensure that $S \cap G_{a,\bar{t}} = S_a \cap G_{a,\bar{t}}$, $S \cap G_{b,\bar{t}} = S_b \cap G_{b,\bar{t}}$ and $S_a \cap G_{a,b} = S_b \cap G_{a,b}$. If these conditions are not satisfied, we skip the current pair of indices $\mathfrak{I}_a$, $\mathfrak{I}_b$. In the following, we use the notation $R = S_a \cap G_{a,b}$ $(= S_b \cap G_{a,b})$.

**Step 2 (Pairings of degree one vertices and $j$).** First, we deal with Special Case 1, i.e. $j = 2$ and $S = \emptyset$. We then check whether the graph obtained from taking $R$ and adding an edge (and, if not already present, the corresponding vertices) for each pair in $Q_a$ and $Q_b$ is a single induced path. Note that we require the values of the integers $j_a$ and $j_b$ to be the number of endpoints of the resulting path in $V_a$ and $V_b$, respectively.

Since the case $j = 0$ and $S = \emptyset$ is dealt with in Special Case 3 and $S$ cannot be empty whenever $j = 1$, we may from now on assume that $S \neq \emptyset$ and hence $S \odot Q \neq \emptyset$.[1]

Consider the graph on vertex set $V(S) \cup V(R)$ whose edges consist of the edges in $S$ and $R$ together with the pairs in $Q_a$ and $Q_b$. We then contract all edges in $R$ and all edges that were added due to the pairings $Q_a$ and $Q_b$ and incident with a vertex not in $S$, and denote the resulting graph by $\mathcal{H}$. Then, $Q_a$ and $Q_b$ are compatible if and only if $\mathcal{H} = S \odot Q$. By the definition of the table entries (and since by Step 0, $S \odot Q$ is a disjoint union of paths) we can then see that $Q_a, Q_b$ together with the edges of $R$ connect the paired degree one vertices of $Q$ as required. We furthermore need to ensure that the values of the integers $j_a$ and $j_b$ are the number of degree one endpoints in $\mathcal{H}[V_a]$ and $\mathcal{H}[V_b]$, respectively. For an illustration see Figure 3.

**Step 3 (Minimal vertex covers).** We now describe the checks we have to perform to ensure that $M_a$ and $M_b$ are compatible with $M$, which from now on we will denote by $M_t$ to avoid confusion. For ease of exposition, we denote by $\mathcal{I}_t$, $\mathcal{I}_a$ and $\mathcal{I}_b$ (potential) partial solutions corresponding to $\mathfrak{I}_t$, $\mathfrak{I}_a$ and $\mathfrak{I}_b$, respectively.

Recall that the purpose of the minimal vertex cover $M_t$ is to ensure that no unwanted edges appear between vertices used by the partial solution $\mathcal{I}_t$ and any partial solution of $G[\bar{V_t} \setminus \mathrm{bd}(\bar{V_t})]$ that can be combined with $\mathcal{I}_t$. Hence, when checking whether $\mathcal{I}_a$ and $\mathcal{I}_b$ can

[1] Note that it could still happen that $Q = \emptyset$ but since this does not essentially influence the following argument, we assume that $Q \neq \emptyset$.

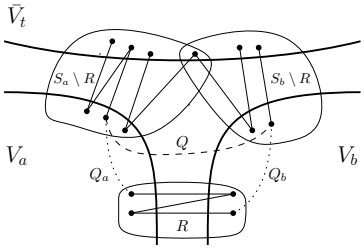



**Figure 3** Step 3 of the join operation. Recall that $S_a \cap G_{a,b} = S_b \cap G_{a,b} = R$ by Step 1.

be combined to $\mathcal{I}_t$ without explicitly having access to these sets of induced disjoint paths, we have to make sure that the indices $\mathfrak{I}_a$ and $\mathfrak{I}_b$ assert the absence of unwanted edges — for any intersection of a partial solution with $G_{a,\bar{a}}$ and $G_{b,\bar{b}}$, as well as with $G_{a,b}$. Recall that $G_{a,\bar{a}} = G_{a,\bar{t}} \cup G_{a,b}$ and $G_{b,\bar{b}} = G_{b,\bar{t}} \cup G_{a,b}$.

We distinguish several cases, depending on where the unwanted edge might appear: First, between two intermediate vertices of partial solutions and second, between a vertex in $S_a$ or $S_b$ and an intermediate vertex. Step 3.1 handles the former and Step 3.2 the latter. In Step 3.1, we additionally have to distinguish whether the edge might appear in $G_{a,b}$ or in $G_{a,\bar{t}}$ (respectively, in $G_{b,\bar{t}}$). In the following, we let $M_a^{\text{out}(b)} := M_a^{\text{out}} \cap V_b$ and $M_b^{\text{out}(a)} := M_b^{\text{out}} \cap V_a$.

**Step 3.1.1 (intermediate-intermediate, $G_{a,b}$).** $M_a^{\text{out}(b)} \subseteq M_b^{\text{in}}$ and $M_b^{\text{out}(a)} \subseteq M_a^{\text{in}}$: A vertex $v \in M_a^{\text{out}(b)}$ can have a neighbor $w \in V_a$ which is used as an intermediate vertex in $\mathcal{I}_a$. Hence, to avoid that the unwanted edge $\{v, w\}$ appears in the combined solution $\mathcal{I}_a \cup \mathcal{I}_b$, we have to make sure that $v$ is not used by $\mathcal{I}_b$, which is asserted if $v \in M_b^{\text{in}}$. By a symmetric argument we justify that $M_b^{\text{out}(a)} \subseteq M_a^{\text{in}}$.

**Step 3.1.2 (intermediate-intermediate, $G_{a,\bar{t}}$ or $G_{b,\bar{t}}$).** We have to check the following two conditions, the first one regarding $M_t^{\text{in}}$ and the second one regarding $M_t^{\text{out}}$.

- **(a)** $M_t^{\text{in}} \subseteq M_a^{\text{in}} \cup M_b^{\text{in}}$: By the definition of $M_t^{\text{in}}$, $\mathcal{I}_t$ has to avoid the vertices in $M_t^{\text{in}}$. Hence, $\mathcal{I}_a$ and $\mathcal{I}_b$ have to avoid the vertices in $M_t^{\text{in}}$ as well, which is ensured if for $v \in M_t^{\text{in}} \cap V_a$, we have that $v \in M_a^{\text{in}}$ and for $w \in M_t^{\text{in}} \cap V_b$, we have that $w \in M_b^{\text{in}}$.
- **(b)** For each vertex $v \in M_t^{\text{out}}$ having a neighbor $x$ in $V_a$ such that $x$ is also contained in $V_a \setminus (V(S_a) \cup M_a^{\text{in}})$, we have that $v \in M_a^{\text{out}}$: Recall that by the definition of $M_t^{\text{out}}$, $\mathcal{I}_t$ could use the vertex $x$ as an intermediate vertex. If $x \notin V(S_a) \cup M_a^{\text{in}}$, this means that $x$ might be used by $\mathcal{I}_a$ as an intermediate vertex as well. Now, in a table entry representing a partial solution $\mathcal{I}_a$ using $x$, this is signalized by having $v \in M_a^{\text{out}}$. We check the analogous condition for $M_b^{\text{out}}$.

**Step 3.2 (intermediate-($S_a$ or $S_b$)).** $N_a[V(S_b) \setminus V(S_a)] \subseteq M_a^{\text{in}}$ and $N_b[V(S_a) \setminus V(S_b)] \subseteq M_b^{\text{in}}$: We justify the first condition and note that the second one can be argued for symmetrically. Clearly, $\mathcal{I}_a$ cannot have a neighbor $x$ of any vertex $v \in V(S_b)$ as an intermediate vertex, if $\mathcal{I}_a$ is to be combined with $\mathcal{I}_b$. However, if $v \in V(S_a)$, then $\mathcal{I}_a$ does not use $x$ by Part (2) of the definition of the table entries. Note that this includes all vertices in $V(R) \subseteq V(S_a)$. If on the other hand, $v \in V(S_b) \setminus V(S_a)$ then the neighbors of $v$ have not been accounted for earlier, since $v$ is not a vertex in the partial solution $\mathcal{I}_a$. Hence, we now have to assert that $\mathcal{I}_a$ does not use $x$, the neighbor of $v$, and so we require that $x \in M_a^{\text{in}}$.

**Step 4 ($i$).** We consider all pairs of integers $i_a$, $i_b$ such that $i = i_a + i_b - |V(R)|$. By Step 2, all vertices in $R$ are used in the partial solution $\mathcal{I}_t$. They are counted twice, since they are both accounted for in $\mathcal{I}_a$ and in $\mathcal{I}_b$.

Now, we let $\mathcal{T}_t[\mathfrak{I}_t] = 1$ if and only if there is a pair of indices $\mathfrak{I}_a = ((S_a, M_a, Q_a), i_a, j_a)$ and $\mathfrak{I}_b = ((S_b, M_b, Q_b), i_b, j_b)$ passing all checks performed in Steps 1-4 above, such that $\mathcal{T}_a[\mathfrak{I}_a] = 1$ and $\mathcal{T}_b[\mathfrak{I}_b] = 1$. This finishes the description of the algorithm.

▶ **Proposition 7 (★).** *Let $t \in V(T)$. The table entries $\mathcal{T}_t[\mathfrak{I}_t]$ computed according to Steps 0-4 above are correct.*

By Propositions 5 and 7 and the fact that in the leaf nodes of $T$, we enumerate all possible partial solutions, we know that the algorithm we described is correct. Since by Proposition 6, there are at most $n^{\mathcal{O}(w)}$ table entries at each node of $T$ (and they can be enumerated in time $n^{\mathcal{O}(w)}$), the value of each table entry in $\mathcal{T}_t$ as above can be computed in time $n^{\mathcal{O}(w)} \cdot n^{\mathcal{O}(w)} \cdot n^{\mathcal{O}(1)} = n^{\mathcal{O}(w)}$, since each check described in Steps 0-4 can be done in time polynomial in $n$. Since additionally, $|V(T)| = \mathcal{O}(n)$, the total runtime of the algorithm is $n^{\mathcal{O}(w)} \cdot n^{\mathcal{O}(w)} \cdot \mathcal{O}(n) = n^{\mathcal{O}(w)}$ and we have the following theorem.

▶ **Theorem 8.** *There is an algorithm that given a graph $G$ on $n$ vertices and a branch decomposition $(T, \mathcal{L})$ of $G$, solves* Longest Induced Path *in time $n^{\mathcal{O}(w)}$, where $w$ denotes the mim-width of $(T, \mathcal{L})$.*

## 3.2 Induced Disjoint Paths

In this section, we build upon the ideas of the algorithm for Longest Induced Path presented above to obtain an $n^{\mathcal{O}(w)}$-time algorithm for the following parameterized problem.

Induced Disjoint Paths (IDP)/Mim-Width
**Input:** A graph $G$ with branch decomposition $(T, \mathcal{L})$ and pairs of vertices $(x_1, y_1)$, …, $(x_k, y_k)$ of $G$.
**Parameter:** $w := \mathrm{mimw}(T, \mathcal{L})$
**Question:** Does $G$ contain a set of vertex-disjoint induced paths $P_1, \ldots, P_k$, such that for $i \in [k]$, $P_i$ is a path from $x_i$ to $y_i$ and for $i \neq j$, $P_i$ does not contain a vertex adjacent to a vertex in $P_j$?

Throughout the remainder of this section, we refer to the vertices $\{x_i, y_i\}$, where $i \in [k]$ as the *terminals* and we denote the set of all terminals by $X := \bigcup_{i \in [k]} \{x_i, y_i\}$. We furthermore use the following notation: We denote by $\mathcal{C}(G)$ the set of all connected components of $G$ and for a vertex $v \in V(G)$, $C_G(v)$ refers to the connected component containing $v$.

We observe how a solution $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_k)$ interacts with the graph $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$, for some $t \in V(T)$. In this case, for each $i \in [k]$, $\mathcal{P}_i$ is an $(x_i, y_i)$-path and additionally for $j \neq i$, there is no vertex in $\mathcal{P}_i$ adjacent to a vertex of $\mathcal{P}_j$. The intersection of $\mathcal{P}$ with $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$ is a subgraph $\mathcal{I} = (\mathcal{I}_1, \ldots, \mathcal{I}_k)$, where each $\mathcal{I}_i$ is a (possibly empty) induced disjoint union of paths which is the intersection of the $(x_i, y_i)$-path $\mathcal{P}_i$ with $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$. Note that each terminal $v_i \in \{x_i, y_i\}$ that is contained in $V_t \cup \mathrm{bd}(\bar{V}_t)$ is also contained in $V(\mathcal{I}_i)$.

Again our goal is to bound the number of table entries at each node $t \in V(T)$ by $n^{\mathcal{O}(w)}$, so we focus on the intersection of $\mathcal{I}$ with the crossing graph $G_{t,\bar{t}}$. There are several reasons why $\mathcal{I}_i$ can have a nonempty intersection with the crossing graph $G_{t,\bar{t}}$: If precisely one of $x_i$ and $y_i$ is contained in $V_t$, then the path $\mathcal{P}_i$ must cross the boundary of $G_t$. If both $x_i$ and $y_i$ are contained in $V_t$ ($\bar{V}_t$), yet $\mathcal{P}_i$ uses a vertex of $\bar{V}_t$ ($V_t$), then it crosses the boundary of $G_t$.

We now turn to the definition of the table indices. Let us first point out what table indices in the resulting algorithm for Induced Disjoint Paths have in common with the indices in the algorithm for Longest Induced Path and we refer to Section 3.1 for the motivation and details. These similarities arise since in both problems, the intersection of a solution with a crossing graph $G_{t,\bar{t}}$ is an induced disjoint union of paths.

- The intersection of $\mathcal{I}$ with the edges of $G_{t,\bar{t}}$ is $S$, an induced disjoint union of paths where each component contains at least one edge.
- $M$ is a minimal vertex cover of $G_{t,\bar{t}} - V(S)$ such that $M \cap V(S) = \emptyset$.

The first important observation to be made is that by Lemma 4, the number of components of $S$ is linearly bounded in $w$ and hence at most $\mathcal{O}(w)$ paths of $\mathcal{P}$ can have a nonempty intersection with $G_{t,\bar{t}}$. We need to store information about which path $\mathcal{P}_i$ (resp., to which $\mathcal{I}_i$) the components of $S$ correspond to. To do so, another part of the index will be a labeling function $\lambda\colon \mathcal{C}(S) \to [k]$, whose purpose is to indicate that each component $C \in \mathcal{C}(S)$ is contained in $\mathcal{I}_{\lambda(C)}$. We just observed that each such $\lambda$ contains at most $\mathcal{O}(w)$ entries.

Let $i \in [k]$. Again, we need to indicate how (some of) the components of $S$ are connected via $\mathcal{I}_i$ in $G[V_t]$. As before, we do so by considering a pairing of the vertices in $S[V_t]$ that have degree one in $S$, however in this case we also have to take into account the labeling function $\lambda$. That is, two such vertices $s$ and $t$ can only be paired if they belong to the same induced disjoint union of paths $\mathcal{I}_i$.

In accordance with the above discussion, we define the table entries as follows. We let $\mathcal{T}[t, (S, M, \lambda, Q_\lambda)] = 1$ if and only if the following conditions are satisfied.

- **(i)** There is an induced disjoint union of paths $\mathcal{I} = (\mathcal{I}_1, \ldots, \mathcal{I}_k)$ in $G[V_t \cup \mathrm{bd}(\bar{V_t})]$, such that for $i \neq j$, $\mathcal{I}_i$ does not contain a vertex adjacent (in $G$) to a vertex in $\mathcal{I}_j$. For each $i \in [k]$, we have that if $v_i \in \{x_i, y_i\} \cap V_t$, then $v_i \in \mathcal{I}_i$. Furthermore, $v_i$ has degree one in $\mathcal{I}_i$.
- **(ii)** **(a)** $E(\mathcal{I}) \cap E(G_{t,\bar{t}}) = E(S)$.
  - **(b)** $\lambda\colon \mathcal{C}(S) \to [k]$ is a labeling function of the connected components of $S$, such that for each component $C \in \mathcal{C}(S)$, $\lambda(C) = i$ if and only if $C \subseteq \mathcal{I}_i$.
- **(iii)** $M$ is a minimal vertex cover of $G_{t,\bar{t}} - V(S)$ such that $V(\mathcal{I}) \cap M = \emptyset$.
- **(iv)** Let $D$ denote the set of vertices in $S[V_t]$ that have degree one in $S$ and let $X_t = X \cap V_t$. Then, $Q_\lambda$ is a pairing (i.e. a partition into pairs) of the vertices in $D \triangle X_t$ with the following properties.[2]
  1. $(s, t) \in Q_\lambda$ if and only if there is a path from $s$ to $t$ in $\mathcal{I}[V_t]$. Note that this implies that $(s, t) \in Q_\lambda$ only if both $s$ and $t$ belong to the same $\mathcal{I}_i$ for some $i \in [k]$ and in particular only if $s, t \in V(\lambda^{-1}(i)) \cup \{x_i, y_i\}$.
  2. For each $i \in [k]$, $(x_i, y_i) \in Q_\lambda$ only if $\lambda^{-1}(i) = \emptyset$, i.e. no component of $S$ has label $i$.

Using this definition of the table indices, one can design an algorithm solving Induced Disjoint Paths in time $n^{\mathcal{O}(w)}$ analogously to the algorithm for Longest Induced Path. We give further details in the full version [10] and we have the following theorem.

▶ **Theorem 9.** *There is an algorithm that given a graph $G$ on $n$ vertices, pairs of terminal vertices $(x_1, y_1), \ldots, (x_k, y_k)$ and a branch decomposition $(T, \mathcal{L})$ of $G$, solves* Induced Disjoint Paths *in time $n^{\mathcal{O}(w)}$, where $w$ denotes the mim-width of $(T, \mathcal{L})$.*

## 3.3 $H$-Induced Topological Minor

Let $G$ be a graph and $uv \in E(G)$. We call the operation of replacing the edge $uv$ by a new vertex $x$ and edges $ux$ and $xv$ the *edge subdivision* of $uv$. We call a graph $H$ a *subdivision* of $G$ if it can be obtained from $G$ by a series of edge subdivisions. We call $H$ an *induced topological minor* of $G$ if a subdivision of $H$ is isomorphic to an induced subgraph of $G$.

[2] We denote by '$\triangle$' the symmetric difference, i.e. $D \triangle X_t = (D \cup X_t) \setminus (D \cap X_t)$. $Q_\lambda$ is a pairing on $D \triangle X_t$, since if a terminal $v_i$ is contained in $D$, it is supposed to be paired 'with itself': Since $v_i$ has degree one in $\mathcal{I}_i$ by (1) and is incident to an edge in $S$, $v_i$ cannot be paired with another vertex.

$H$-Induced Topological Minor/Mim-Width
**Input:** A graph $G$ with branch decomposition $(T, \mathcal{L})$
**Parameter:** $w := \text{mimw}(T, \mathcal{L})$
**Question:** Does $G$ contain $H$ as an induced topological minor?

▶ **Theorem 10 (★).** *There is an algorithm that given a graph $G$ on $n$ vertices and a branch decomposition $(T, \mathcal{L})$ of $G$, solves $H$-Induced Topological Minor in time $n^{\mathcal{O}(w)}$, where $H$ is a fixed graph and $w$ the mim-width of $(T, \mathcal{L})$.*

## References


1 Rémy Belmonte, Petr A. Golovach, Pinar Heggernes, Pim van 't Hof, Marcin Kaminski, and Daniël Paulusma. Detecting fixed patterns in chordal graphs in polynomial time. *Algorithmica*, 69(3):501–521, 2014. doi:10.1007/s00453-013-9748-5.

2 Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theor. Comput. Sci.*, 511:54–65, 2013. doi:10.1016/j.tcs.2013.01.011.

3 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoret. Comput. Sci.*, 511:66–76, 2013.

4 Jirí Fiala, Marcin Kaminski, Bernard Lidický, and Daniël Paulusma. The k-in-a-path problem for claw-free graphs. *Algorithmica*, 62(1-2):499–519, 2012. doi:10.1007/s00453-010-9468-z.

5 Fanica Gavril. Algorithms for maximum weight induced paths. *Inf. Process. Lett.*, 81(4):203–208, 2002. doi:10.1016/S0020-0190(01)00222-8.

6 Petr A. Golovach, Daniël Paulusma, and Erik Jan van Leeuwen. Induced disjoint paths in at-free graphs. In Fedor V. Fomin and Petteri Kaski, editors, *Algorithm Theory - SWAT 2012 - 13th Scandinavian Symposium and Workshops, Helsinki, Finland, July 4-6, 2012. Proceedings*, volume 7357 of *Lecture Notes in Computer Science*, pages 153–164. Springer, 2012. doi:10.1007/978-3-642-31155-0_14.

7 Petr A. Golovach, Daniël Paulusma, and Erik Jan van Leeuwen. Induced disjoint paths in circular-arc graphs in linear time. *Theor. Comput. Sci.*, 640:70–83, 2016. doi:10.1016/j.tcs.2016.06.003.

8 Petr Hlinený, Sang-il Oum, Detlef Seese, and Georg Gottlob. Width parameters beyond tree-width and their applications. *Comput. J.*, 51(3):326–362, 2008. doi:10.1093/comjnl/bxm052.

9 Tetsuya Ishizeki, Yota Otachi, and Koichi Yamazaki. An improved algorithm for the longest induced path problem on k-chordal graphs. *Discrete Applied Mathematics*, 156(15):3057–3059, 2008. doi:10.1016/j.dam.2008.01.019.

10 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Polynomial-time algorithms for the longest induced path and induced disjoint paths problems on graphs of bounded mim-width. *ArXiv e-prints*, 2017. arXiv:1708.04536.

11 Dong Yeap Kang, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. A width parameter useful for chordal and co-comparability graphs. In Sheung-Hung Poon, Md. Saidur Rahman, and Hsu-Chun Yen, editors, *WALCOM: Algorithms and Computation, 11th International Conference and Workshops, WALCOM 2017, Hsinchu, Taiwan, March 29-31, 2017, Proceedings.*, volume 10167 of *Lecture Notes in Computer Science*, pages 93–105. Springer, 2017. doi:10.1007/978-3-319-53925-6_8.

12 Ken-ichi Kawarabayashi and Yusuke Kobayashi. The induced disjoint paths problem. In Andrea Lodi, Alessandro Panconesi, and Giovanni Rinaldi, editors, *Integer Programming and Combinatorial Optimization, 13th International Conference, IPCO 2008, Bertinoro,*

*Italy, May 26-28, 2008, Proceedings*, volume 5035 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2008. doi:10.1007/978-3-540-68891-4_4.

13 Dieter Kratsch, Haiko Müller, and Ioan Todinca. Feedback vertex set and longest induced path on at-free graphs. In Hans L. Bodlaender, editor, *Graph-Theoretic Concepts in Computer Science, 29th International Workshop, WG 2003, Elspeet, The Netherlands, June 19-21, 2003, Revised Papers*, volume 2880 of *Lecture Notes in Computer Science*, pages 309–321. Springer, 2003. doi:10.1007/978-3-540-39890-5_27.

14 Sridhar Natarajan and Alan P. Sprague. Disjoint paths in circular arc graphs. *Nordic J. of Computing*, 3(3):256–270, 1996. URL: http://dl.acm.org/citation.cfm?id=642150.642154.

15 Neil Robertson and Paul D. Seymour. Graph minors .xiii. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

16 Sigve Hortemo Sæther and Martin Vatshelle. Hardness of computing width parameters based on branch decompositions over the vertex set. *Theor. Comput. Sci.*, 615:120–125, 2016.

17 Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial $k$-trees. *SIAM J. Discrete Math.*, 10(4):529–550, 1997. doi:10.1137/S0895480194275825.

18 Martin Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, 2012.

# Optimal Data Reduction for Graph Coloring Using Low-Degree Polynomials*

**Bart M. P. Jansen[1] and Astrid Pieterse[2]**

1 **Eindhoven University of Technology, Eindhoven, The Netherlands**
b.m.p.jansen@tue.nl
2 **Eindhoven University of Technology, Eindhoven, The Netherlands**
a.pieterse@tue.nl

## Abstract

The theory of kernelization can be used to rigorously analyze data reduction for graph coloring problems. Here, the aim is to reduce a $q$-Coloring input to an equivalent but smaller input whose size is provably bounded in terms of structural properties, such as the size of a minimum vertex cover. In this paper we settle two open problems about data reduction for $q$-Coloring. First, we use a recent technique of finding redundant constraints by representing them as low-degree polynomials, to obtain a kernel of bitsize $\mathcal{O}(k^{q-1} \log k)$ for $q$-Coloring parameterized by Vertex Cover for any $q \geq 3$. This size bound is optimal up to $k^{o(1)}$ factors assuming NP $\not\subseteq$ coNP/poly, and improves on the previous-best kernel of size $\mathcal{O}(k^q)$. Our second result shows that 3-Coloring does not admit non-trivial sparsification: assuming NP $\not\subseteq$ coNP/poly, the parameterization by the number of vertices $n$ admits no (generalized) kernel of size $\mathcal{O}(n^{2-\varepsilon})$ for any $\varepsilon > 0$. Previously, such a lower bound was only known for coloring with $q \geq 4$ colors.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

**Keywords and phrases** graph coloring, kernelization, sparsification

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.22

## 1 Introduction

The $q$-Coloring problem asks whether the vertices of a graph can be properly colored using $q$ colors. It is one of many colorability problems on graphs that have been widely studied. Since these are often NP-hard, they are good candidates to study from a parameterized perspective [2, 5]. Here we use additional parameters, other than the size of the input, to describe the complexity of the problem. In this paper we study preprocessing algorithms (called kernelizations or kernels) that aim to reduce the size of an input graph in polynomial time, without changing its colorability status.

The natural choice for a parameter for $q$-Coloring is the number of colors $q$. However, since even 3-Coloring is NP-hard, this parameter does not give interesting results. Therefore the problem is studied using different parameters, that often try to capture the complexity of the input graph. For example, Fiala et. al. [6] compared the parameterized complexity of several coloring problems when parameterized by vertex cover, to the complexity when parameterized by treewidth. Jansen and Kratsch [8] studied graph coloring when parameterized by a hierarchy of different parameters.

* This work was supported by NWO Veni grant "Frontiers in Parameterized Preprocessing" and NWO Gravitation grant "Networks".





In this earlier work [8], Jansen and Kratsch provided a kernel for $q$-Coloring parameterized by Vertex Cover with $\mathcal{O}(k^q)$ vertices that can be encoded in $\mathcal{O}(k^q)$ bits. Furthermore they showed that for $q \geq 4$, a kernel of bitsize $\mathcal{O}(k^{q-1-\varepsilon})$ is unlikely to exist. Unfortunately, these bounds left a gap of a factor $k$ and it remained unclear whether the upper or the lower bound had to be strengthened. As our first main result, we close this gap. We show in Theorem 7 that the kernel for $q$-Coloring parameterized by Vertex Cover can be further improved to have $\mathcal{O}(k^{q-1})$ vertices and a bitsize of $\mathcal{O}(k^{q-1} \log k)$. This matches the previously known lower bound up to $k^{o(1)}$ factors.

To obtain this improvement, we use a recent result by the current authors [9] about the kernelization of constraint satisfaction problems when parameterized by the number of variables. A non-trivial data reduction can be achieved when the constraints are given by equalities of low-degree polynomials on boolean variables. The size of the resulting instance then depends on the maximum degree of the given polynomials. Suppose now we are given a 3-Coloring instance $G$ with vertex cover $S$ and let $I = V(G) \setminus S$ be the corresponding independent set. One can think of each vertex $v \in I$ as a constraint of the form "my neighbors use at most 2 different colors", such that a remaining color can be used to color $v$. We write these constraints as polynomial equalities and apply our previous result to find out which ones are redundant. Since vertices of the independent set can be colored independently, a vertex that corresponds to a redundant constraint can be removed from $G$, without changing the 3-colorability of $G$. To apply this idea to obtain a kernel for $q$-Coloring parameterized by Vertex Cover, the key technical step is to build a polynomial of degree $q-1$ that captures the desired constraint.

Our second main result concerns the parameterization by the number of vertices $n$. The current authors showed in earlier work [10] that for a number of graph problems it is impossible to give a kernel of size $\mathcal{O}(n^{2-\varepsilon})$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. This implies that the number of edges cannot efficiently be reduced to a subquadratic amount without changing the answer, a task that is also known as sparsification. For example, $q$-Coloring was shown to have no non-trivial sparsification for any $q \geq 4$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. The case for $q = 3$ remained open. One might think that 3-Coloring is so restrictive, that a 3-colorable instance is likely to either be sparse, or have a very specific structure. Exploiting this structure could then allow for a non-trivial sparsification. In Theorem 12 we show that this is not the case: 3-Coloring allows no kernel of size $\mathcal{O}(n^{2-\varepsilon})$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

From this bound it follows that the $\Omega(k^{q-1-\varepsilon})$ lower bound for the parameterization by vertex cover also holds for $q = 3$, since the size of a vertex cover is at most the total number of vertices in the graph. This completely settles the kernelization complexity of $q$-Coloring parameterized by Vertex Cover, up to $k^{o(1)}$ factors.

### Related work

Dell and Van Melkebeek showed that $d$-CNF-Satisfiability with $n$ variables has no kernel of size $\mathcal{O}(n^{d-\varepsilon})$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ [4]. Continuing this line of research, precise kernel lower bounds were shown for a variety of problems. For example, it was shown that Vertex Cover is unlikely to have a kernel of size $\mathcal{O}(k^{2-\varepsilon})$ [4], while a kernel with $\mathcal{O}(k^2)$ edges and $\mathcal{O}(k)$ vertices is known. Furthermore, the Point-Line cover problem, which asks to cover a set of $n$ points in the plane with at most $k$ lines, was proven to have a tight kernel lower bound of size $\mathcal{O}(k^{2-\varepsilon})$ [11], assuming $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$. Dell and Marx [3] proved polynomial kernelization lower bounds for several packing problems. They showed how a table structure can help realize the reduction that is needed for such a lower bound. We will also use this table structure in this paper.

## 2 Preliminaries

To denote the set of numbers 1 to $n$, we use the following notation: $[n] := \{i \in \mathbb{N} \mid 1 \leq i \leq n\}$. For $x, y \in \mathbb{Z}$ we write $x \equiv_2 y$ to denote that $x$ and $y$ are congruent modulo 2. For a finite set $X$ and non-negative integer $k$, let $\binom{X}{k}$ be the collection of all subsets of $X$ of size exactly $k$.

A graph $G$ has vertex set $V(G)$ and edge set $E(G)$. All graphs considered in this paper are simple and undirected. For a vertex $u \in V(G)$, let $N_G(u) := \{v \in V(G) \mid \{u, v\} \in E(G)\}$ denote its open *neighborhood*. Let $G[S]$ for $S \subseteq V(G)$ denote the subgraph of $G$ induced by $S$. A *vertex cover* of a graph $G$ is a set $S \subseteq V(G)$ such that each edge has at least one endpoint in $S$ (equivalently, $V(G) \setminus S$ is an *independent set* in $G$). A *proper $q$-coloring* of $G$ is a function $c\colon V(G) \to [q]$ such that for all $\{u, v\} \in E(G)\colon c(u) \neq c(v)$.

A *parameterized problem* $\mathcal{Q}$ is a subset of $\Sigma^* \times \mathbb{N}$, where $\Sigma$ is a finite alphabet. Let $\mathcal{Q}, \mathcal{Q}' \subseteq \Sigma^* \times \mathbb{N}$ be parameterized problems and let $h\colon \mathbb{N} \to \mathbb{N}$ be a computable function. A *generalized kernel for $\mathcal{Q}$ into $\mathcal{Q}'$ of size $h(k)$* is an algorithm that, on input $(x, k) \in \Sigma^* \times \mathbb{N}$, takes time polynomial in $|x| + k$ and outputs an instance $(x', k')$ such that:

1. $|x'|$ and $k'$ are bounded by $h(k)$, and
2. $(x', k') \in \mathcal{Q}'$ if and only if $(x, k) \in \mathcal{Q}$.

The algorithm is a *kernel* for $\mathcal{Q}$ if $\mathcal{Q} = \mathcal{Q}'$. It is a *polynomial (generalized) kernel* if $h(k)$ is a polynomial. Since a polynomial-time reduction to an equivalent sparse instance yields a generalized kernel, a lower bound for the size of a generalized kernel can be used to prove the non-existence of sparsification algorithms.

We use the framework of cross-composition [1] to establish kernelization lower bounds, requiring the definitions of polynomial equivalence relations and OR-cross-compositions. We repeat them here for completeness:

▶ **Definition 1** (Polynomial equivalence relation, [1, Def. 3.1]). An equivalence relation $\mathcal{R}$ on $\Sigma^*$ is called a *polynomial equivalence relation* if the following conditions hold.

- There is an algorithm that, given two strings $x, y \in \Sigma^*$, decides whether $x$ and $y$ belong to the same equivalence class in time polynomial in $|x| + |y|$.
- For any finite set $S \subseteq \Sigma^*$ the equivalence relation $\mathcal{R}$ partitions the elements of $S$ into a number of classes that is polynomially bounded in the size of the largest element of $S$.

▶ **Definition 2** (Cross-composition, [1, Def. 3.3]). Let $L \subseteq \Sigma^*$ be a language, let $\mathcal{R}$ be a polynomial equivalence relation on $\Sigma^*$, let $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem, and let $f\colon \mathbb{N} \to \mathbb{N}$ be a function. An OR-*cross-composition of $L$ into $\mathcal{Q}$* (with respect to $\mathcal{R}$) *of cost $f(t)$* is an algorithm that, given $t$ instances $x_1, x_2, \ldots, x_t \in \Sigma^*$ of $L$ belonging to the same equivalence class of $\mathcal{R}$, takes time polynomial in $\sum_{i=1}^{t} |x_i|$ and outputs an instance $(y, k) \in \Sigma^* \times \mathbb{N}$ such that:

- The parameter $k$ is bounded by $\mathcal{O}(f(t) \cdot (\max_i |x_i|)^c)$, where $c$ is some constant independent of $t$, and
- instance $(y, k) \in \mathcal{Q}$ if and only if there is an $i \in [t]$ such that $x_i \in L$.

▶ **Theorem 3** ([1, Theorem 6]). *Let $L \subseteq \Sigma^*$ be a language, let $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem, and let $d, \varepsilon$ be positive reals. If $L$ is NP-hard under Karp reductions, has an* OR*-cross-composition into $\mathcal{Q}$ with cost $f(t) = t^{1/d+o(1)}$, where $t$ denotes the number of instances, and $\mathcal{Q}$ has a polynomial (generalized) kernelization with size bound $\mathcal{O}(k^{d-\varepsilon})$, then* NP $\subseteq$ coNP/poly.

We will refer to an OR-cross-composition of cost $f(t) = \sqrt{t}\log(t)$ as a *degree-2 cross-composition*. By Theorem 3, a degree-2 cross-composition can be used to rule out generalized kernels of size $\mathcal{O}(k^{2-\varepsilon})$.

## 3 Kernel for $q$-Coloring parameterized by Vertex Cover

In this section we develop a kernel for $q$-Coloring parameterized by Vertex Cover. The main tool is an earlier result [9] on constraint satisfaction problems (CSPs). In the right conditions, it can be used to reduce the number of constraints without changing the answer. We recall the required terminology. Define $d$-Polynomial Root CSP over the integers modulo 2 as the problem whose input consists of a set $L$ of polynomial equalities over a set of boolean variables $V = \{x_1, \ldots, x_n\}$. Each equality is of the form $p(x_1, \ldots, x_n) \equiv_2 0$, where each polynomial has degree at most $d$. The question is whether all equalities can be satisfied by setting the input variables to 0 or 1. The following theorem follows directly from Theorem 2 together with Claim 3 in [9], where $n$ is the total number of used variables.

▶ **Theorem 4.** *There is a polynomial-time algorithm that, given an instance $(L, V)$ of $d$-Polynomial root CSP over an efficient field $F$, outputs $L' \subseteq L$ with at most $n^d + 1$ constraints such that any 0/1-assignment to $V$ satisfies $L'$ if and only if it satisfies $L$.*

A field $F$ is efficient if the field operations and Gaussian elimination can be done in polynomial time in the size of a reasonable input encoding. For our purposes it is only relevant that the integers modulo 2 form an efficient field.

To apply this machinery, we need to show how the coloring constraints expressed by an independent set of vertices can be encoded as polynomial equalities. To encode the color of a vertex $v_i$ in this context we will use $q$ boolean variables $y_{i,1}, \ldots, y_{i,q}$, one per possible color. The variable $y_{i,k}$ is set to *true* if vertex $v_i$ has color $k$. We now define a choice assignment to the variables, to express that each vertex gets exactly one color.

▶ **Definition 5.** Let $\{y_{i,k} \mid i \in [n], k \in [q]\}$ be a set of boolean variables and let $\mathbf{y}$ be the vector containing all these variables. We say $\mathbf{y}$ is given a *choice assignment* if for all $i \in [n]$:

$$\sum_{k=1}^{q} y_{i,k} = 1.$$

Note that a choice assignment always sets exactly $n$ variables to *true*. The following lemma gives a polynomial that can be used to express the constraint that out of exactly $q$ neighbors of a given vertex $u$, there are at least two that have the same color. This constraint has to be satisfied to allow $u$ to be properly $q$-colored. We will later apply such constraints to all possible subsets of $q$ neighbors of $u$ to obtain a safe reduction.

▶ **Lemma 6.** *Let $q > 0$ be an integer and let $y_{i,k}$ for $i \in [q], k \in [q]$ be boolean variables. Then there exists a polynomial $p$ of degree $q - 1$ such that for any choice assignment to $\mathbf{y}$, we have $p(\mathbf{y}) \equiv_2 0$ if and only if there are $i, j, k \in [q]$ such that $y_{i,k} = y_{j,k} = 1$.*

Before proving Lemma 6, we give the polynomial $p$ corresponding to $q = 3$ as an example.

$$p(\mathbf{y}) := \sum_{i_1 \neq i_2 \in [3]} \prod_{k=1}^{2} y_{i_k,k} = y_{1,1} \cdot y_{2,2} + y_{1,1} \cdot y_{3,2} + y_{2,1} \cdot y_{1,2} + y_{2,1} \cdot y_{3,2} + y_{3,1} \cdot y_{1,2} + y_{3,1} \cdot y_{2,2}.$$

Verify for this example that letting $y_{1,1} = y_{2,1} = y_{3,1} = 1$ and all other variables be zero, gives $p(\mathbf{y}) = 0 \equiv_2 0$. Setting $y_{1,1} = y_{2,2} = y_{3,2} = 1$ and all other variables to zero, gives $p(\mathbf{y}) = 2 \equiv_2 0$. Choosing $y_{1,1} = y_{2,2} = y_{3,3} = 1$ and all other variables zero, gives $p(\mathbf{y}) = 1 \equiv_2 1$, as desired. We now proceed with the general construction.

**Proof of Lemma 6.** Define the multivariate polynomial $p$ as

$$p(\mathbf{y}) := \sum_{\substack{i_1,\ldots,i_{q-1}\in[q]\\ \text{distinct}}} \prod_{k=1}^{q-1} y_{i_k,k}.$$

To understand this polynomial and facilitate the remainder of the proof, it is useful to think of an associated set of variables $x_1, \ldots, x_q$ that take values from $[q]$ and represent a color. For each color variable $x_i$, the corresponding boolean variables $y_{i,1}, \ldots, y_{i,q}$ encode the value taken by $x_i$. In this notation, each monomial of $p$ corresponds to a permutation of all but one of the color variables $x_1, \ldots, x_q$. The monomial evaluates to 1 if the $i$'th variable in this permutation has value $i$ for all $i \in [q-1]$, and to 0 otherwise.

We proceed to show that $p$ has the desired properties. It is easy to see the degree of $p$ is $q-1$. It remains to prove the claim on the values of $p(\mathbf{y})$ for choice assignments. So consider a choice assignment to $\mathbf{y}$, and for each $i \in [q]$ let $x_i := k$ exactly when $y_{i,k} = 1$. This is well-defined as there is exactly one $k \in [q]$ such that $y_{i,k} = 1$. In these terms, we have to show that $p(\mathbf{y}) \equiv_2 0$ if and only if there are distinct color variables $x_i, x_j$ such that $x_i = x_j$.

Suppose there do not exist $i, j \in [q]$ such that $x_i = x_j$, implying that $x_1, \ldots, x_q$ take $q$ distinct values. For $k \in [q-1]$, let $j_k$ be the unique index such that $x_{j_k} = k$, implying that $y_{j_k,k} = 1$. Then, $\prod_{k=1}^{q-1} y_{j_k,k} = 1$. For any other choice of distinct indices $i_1, \ldots, i_{q-1} \in [q]$, there exists $m \in [q-1]$ such that $i_m \neq j_m$. This implies that $y_{i_m,m} = 0$ and thereby $\prod_{k=1}^{q-1} y_{i_k,k} = 0$. Thus, $p(\mathbf{y}) = 1 \equiv_2 1$.

For the other direction, suppose there exist $i, j \in [q]$, such that $x_i = x_j$. We do a case distinction, where we consider the following cases: One color is used at least thrice, or there exist two colors that are both used more than once, or one color is used more than once and color $q$ is used, or all colors except color $q$ are used. More formally:

- There exist distinct $i, j, \ell \in [q]$ such that $x_i = x_j = x_\ell$. Then $p(\mathbf{y}) = 0$, because there do not exist distinct $i_1, \ldots, i_{q-1} \in [q]$ such that $x_{i_k} = k$ (and thus $y_{i_k,k} = 1$) for all $k \in [q-1]$. Hence all monomials of $p$ evaluate to 0 and $p(\mathbf{y}) = 0$.
- There exist distinct $i, j, i', j' \in [q]$ such that $x_i = x_j$ and $x_{i'} = x_{j'}$. Then $p(\mathbf{y}) = 0$, because there do not exist distinct $i_1, \ldots, i_{q-1} \in [q]$ such that $x_{i_k} = k$ for all $k \in [q-1]$.
- There exist distinct $i, j \in [q]$ and there exists $\ell \in [q]$ such that $x_i = x_j$ and $x_\ell = q$. If $x_i = x_j = q$, it is not possible to find distinct $i_1, \ldots, i_{q-1} \in [q] \setminus \{i, j\}$ such that $x_{i_k} = k$ for all $k \in [q-1]$, thereby $p(\mathbf{y}) = 0$. If $x_i \neq q$, it is again not possible to find distinct $i_1, \ldots, i_{q-1} \in [q] \setminus \{\ell\}$ such that $x_{i_k} = k$ for all $k \in [q-1]$ since $x_i$ and $x_j$ are equal.
- Otherwise, there are distinct $i, j \in [q]$ and $k \in [q-1]$ such that $x_i = x_j = k$ and there is no $\ell \in [q]$ such that $x_\ell = q$. Furthermore, there are no distinct $i', j' \in [q] \setminus \{i, j\}$ such that $x_{i'} = x_{j'}$. In other words, each value from $[q-1]$ is assigned to exactly one color variable, except for the value $k$ which occurs twice. For all $c \in [q-1]$ with $c \neq k$, let $i_c$ be the unique index such that $x_{i_c} = c$ and thus $y_{i_c,c} = 1$. Then

$$y_{i,k} \cdot \prod_{\substack{c=1\\ c\neq k}}^{q-1} y_{i_c,c} = y_{j,k} \cdot \prod_{\substack{c=1\\ c\neq k}}^{q-1} y_{i_c,c} = 1.$$

  However, $\prod_{c=1}^{q-1} y_{i_c,c} = 0$ for any other choice of $i_1, \ldots, i_{q-1}$. Thereby, $p(\mathbf{y}) = 2 \equiv_2 0$. ◀

We now give a kernel for the $q$-Coloring problem parameterized by the size of a vertex cover. The problem is defined as follows:

| $q$-Coloring parameterized by Vertex Cover | **Parameter:** $\|S\|$ |
|---|---|
| **Input:** A graph $G$ with a vertex cover $S \subseteq V(G)$. | |
| **Question:** Does $G$ have a proper $q$-coloring? | |

We remark that in settings where no vertex cover of $G$ is known, one can simply apply the kernelization using a 2-approximate vertex cover for $S$.

▶ **Theorem 7.** *For any constant $q \geq 3$, $q$-Coloring parameterized by the size of a vertex cover has a kernel with $\mathcal{O}(k^{q-1})$ vertices, which can be encoded in $\mathcal{O}(k^{q-1} \log k)$ bits. Furthermore, the resulting instance is a subgraph of the original input graph.*

**Proof.** Let input graph $G$ with vertex cover $S$ be given, where $|S| = k$. For each vertex $v \in S$, create boolean variables $C_{v,i}$ for $i \in [q]$. These variables can describe the color of $v$, by choosing $C_{v,i} = 1$ if $v$ has color $i$ and zero otherwise, which will give them a proper choice assignment. Let $\mathbf{C}$ contain all $q \cdot k$ constructed variables.

For each vertex $u \in V(G) \setminus S$, for each $X \in \binom{N_G(u)}{q}$, let $\mathbf{C_{u,X}}$ contain the variables constructed for set $X$ in $N_G(u) \subseteq S$. Use Lemma 6 to obtain a polynomial $p_{u,X}$ of degree $q-1$, such that for any choice assignment to the variables we have $p_{u,X}(\mathbf{C_{u,X}}) \equiv_2 0$ if and only if there exist $i \in [q]$ and $v, w \in X$ such that $C_{v,i} = C_{w,i} = 1$.

Let $L$ be the set of created polynomial equalities, thus $L := \{p_{u,X}(\mathbf{C_{u,X}}) \equiv_2 0 \mid u \in V(G) \setminus S \wedge X \in \binom{N_G(u)}{q}\}$. It is easy to see that $L$ is an instance of $(q-1)$-Polynomial root CSP over the integers modulo 2. Use Theorem 4 in order to find $L' \subseteq L$ with $|L'| \leq (qk)^{q-1} + 1$, such that a boolean assignment to the variables in $\mathbf{C}$ satisfies $L'$ if and only if it satisfies $L$. To obtain the kernel $G'$, start with graph $G[S]$. For every equality $p_{u,X}(\mathbf{C_{u,X}}) \equiv_2 0 \in L'$, add $u$ to $G'$ if $u$ is not yet present in $G'$. Furthermore, connect $u$ to all vertices in $X$ that $u$ is not already adjacent to. It is easy to see that by this procedure, $G'$ is a subgraph of $G$.

▶ **Claim 8.** *$G'$ is $q$-colorable if and only if $G$ is $q$-colorable.*

**Proof.** Since $G'$ is a subgraph of $G$, graph $G'$ is $q$-colorable if $G$ is $q$-colorable.

For the opposite direction, let $c'$ be a proper $q$-coloring of $G'$. For vertex $v \in S$ and color $i \in [q]$, define $C_{v,i} = 1$ if $c'(v) = i$ and $C_{v,i} = 0$ otherwise. By this definition, $\sum_{i=1}^{q} C_{v,i} = 1$ for all $v$, so all variable sets $\mathbf{C_{u,X}}$ are given a choice assignment. We will first show that this assignment satisfies all equalities in $L'$. Let $p_{u,X}(\mathbf{C_{u,X}}) \equiv_2 0 \in L'$. Then $u \in V(G') \setminus S$ and $u$ is connected to all vertices in $X$ in $G'$. Since $u$ is colored by $c'$, its neighbors do no have color $c'(u)$, thus $c'(u)$ is unused in the coloring of $X$. Since $|X| = q$ and we have exactly $q$ colors, this implies that there exist $v, w \in X$ and color $i \in [q]$ such that $C_{v,i} = C_{w,i} = 1$. By Lemma 6, this implies $p_{u,X}(\mathbf{C_{u,X}}) \equiv_2 0$ as required.

From the choice of $L'$ and Theorem 4 it now follows that all equalities in $L$ are satisfied by this assignment. Let $c$ denote the coloring $c'$ restricted to the vertices in $G[S] = G'[S]$. We prove that $c$ can be extended to a proper coloring of $G$. Since $V(G) \setminus S$ is an independent set, such an extension is possible if for each vertex $v \in V(G) \setminus S$ there exists a color that is not used on any vertex of $N_G(v)$.

Now assume for a contradiction that $c$ cannot be extended to properly color some vertex $u$ in $V(G) \setminus S$. Then for each color $i \in [q]$, there exists a vertex $v \in N_G(u)$ with $c(v) = i$ (or else we could use color $i$ for $u$). Since $V(G) \setminus S$ is an independent set, $N_G(u) \subseteq S$. Pick a set $X \subseteq N_G(u)$ containing exactly one vertex of each color, thus $|X| = q$. By Lemma 6, $p_{u,X}(\mathbf{C_{u,X}}) \equiv_2 1$ since there do not exist $v, w \in X$ and color $i \in [q]$ such that $C_{v,i} = C_{w,i} = 1$.

But this contradicts the fact that all polynomial equalities in $L$ are satisfied by the given assignment, since $p_{u,X}(\mathbf{C}_{\mathbf{u},\mathbf{X}}) \equiv_2 0 \in L$. Hence $c$ can be extended to properly color $G$. ◀

▶ **Claim 9.** *$G'$ has at most $\mathcal{O}(k^{q-1})$ vertices and can be encoded in $\mathcal{O}(k^{q-1} \log k)$ bits.*

**Proof.** Theorem 4 guarantees that $|L'| \leq (qk)^{q-1} + 1$ since there are $qk$ boolean variables in total, and the polynomials have degree $q-1$. Thereby, $|V(G')| \leq k + (qk)^{q-1} + 1 = \mathcal{O}(k^{q-1})$, since $q$ is a constant. Furthermore, $|E(G')| \leq |E(G'[S])| + q \cdot |L'| \leq k^2 + q \cdot ((qk)^{q-1} + 1) = \mathcal{O}(k^{q-1})$. An adjacency list encoding of the graph has size $\mathcal{O}(|E| \log |V| + |V|)$, which is $\mathcal{O}(k^{q-1} \cdot \log k^{q-1}) = \mathcal{O}(k^{q-1} \log k)$ for constant $q$. ◀

It is easy to see that the kernel can be computed in polynomial time. Thereby, it follows from Claims 8 and 9 that we have given a kernel for $q$-Coloring of bitsize $\mathcal{O}(k^{q-1} \log k)$. ◀

## 4 Sparsification lower bound for 3-Coloring

In this section we provide a sparsification lower bound for 3-Coloring. We show that 3-Coloring does not have a (generalized) kernel of size $\mathcal{O}(n^{2-\varepsilon})$, unless NP $\subseteq$ coNP/poly. This will also provide a kernel lower bound for 3-Coloring parameterized by vertex cover size, that matches the upper bound given in the previous section up to $k^{o(1)}$ factors.

For ease of presentation, we will prove the lower bound by giving a degree-2 cross-composition from a tailor-made problem to 3-List Coloring. The input to 3-List Coloring is a graph $G$ together with a function $L$ that assigns to each vertex $v$ a list $L(v) \subseteq \{1, 2, 3\}$. The problem asks whether there exists a proper coloring of $G$, such that each vertex is assigned a color from its list. Before presenting the cross-composition, we introduce an important gadget that will be used. It was constructed by Jaffke and Jansen [7]. The gadget, which we will call a blocking-gadget, will be used to forbid one specific coloring of a given vertex set. The following Lemma is a rephrased version of Lemma 15 in [7].

▶ **Lemma 10.** *There is a polynomial-time algorithm that, given $\mathbf{c} = (c_1, \ldots, c_m) \in [3]^m$, outputs a* 3-List-Coloring *instance with $\mathcal{O}(m)$ vertices called blocking-gadget($\mathbf{c}$) that contains distinguished vertices $(\pi_1, \ldots, \pi_m)$. A coloring $f \colon \{\pi_i \mid i \in [m]\} \to [3]$ can be extended to a proper list coloring of blocking-gadget($\mathbf{c}$) if and only if $f(\pi_i) = c_i$ for some $i \in [m]$.*

The blocking-gadget can be used to forbid one specific coloring given by the tuple $\mathbf{c}$ of a set of vertices $v_1, \ldots, v_m$, by adding a blocking-gadget($\mathbf{c}$) and connecting $\pi_i$ to $v_i$ for all $i \in [m]$. If the color of $v_i$ is $c_i$ for all $i$, then the inserted edges prevent all $\pi_i$ to receive the corresponding color $c_i$, and by Lemma 10 the coloring cannot be extended to the gadget. If however the color of $v_i$ differs from $c_i$ for some $i$, the gadget can be properly colored.

Having presented the gadget we use in our construction, we define the source problem for the cross-composition. This problem was also used as the starting problem for a cross-composition in our earlier sparsification lower bound for 4-Coloring [10].

2-3-Coloring with Triangle Split Decomposition [10]

**Input:** A graph $G$ with a partition of its vertex set into $U \cup V$ such that $G[U]$ is an edgeless graph and $G[V]$ is a disjoint union of triangles.

**Question:** Is there a proper 3-coloring $c : V(G) \to \{1, 2, 3\}$ of $G$, such that $c(u) \in \{1, 2\}$ for all $u \in U$? We will refer to such a coloring as a *2-3-coloring* of the graph $G$, since two colors are used to color $U$, and three to color $V$.

▶ **Lemma 11** ([9, Lemma 3]). 2-3-Coloring with Triangle Split Decomposition *is NP-complete.*

To establish a quadratic lower bound on the size of generalized kernels, it suffices to give a degree-2 cross-composition from this special coloring problem into 3-Coloring. Effectively, we have to show that for any $t$, one can efficiently embed a series of $t$ size-$n$ instances indexed as $X_{i,j}$ for $i, j \in [\sqrt{t}]$, into a single 3-Coloring instance with $\mathcal{O}(\sqrt{t} \cdot n^{\mathcal{O}(1)})$ vertices that acts as the logical or of the inputs. To achieve this composition, a common strategy is to construct vertex sets $S_i$ and $T_i$ of size $n^{\mathcal{O}(1)}$ for $i \in [\sqrt{t}]$, such that the graph induced by $S_i \cup T_j$ encodes input $X_{i,j}$. The fact that the inputs can be partitioned into an independent set and a collection of triangles facilitates this embedding; we represent the independent set within sets $S_i$ and the triangles in sets $T_i$. To embed $t$ inputs into a graph on $\mathcal{O}(\sqrt{t} \cdot n^{\mathcal{O}(1)})$ vertices, each vertex will have incident edges corresponding to many different input instances. The main issue when trying to find a cross-composition into 3-Coloring, is to ensure that when there is one 2-3-colorable input graph, the entire graph becomes 3-colorable. This is difficult, since the neighbors that a vertex in $S_i$ has among the many different sets $T_j$ should not invalidate the coloring. For vertices in some set $T_j$, we have a similar issue. Our choice of starting problem ensures that if some combination $S_{i^*}, T_{j^*}$ corresponding to input $X_{i^*,j^*}$ has a 2-3-coloring, then the remaining sets $T_j$ can be safely colored 3, since vertices in $S_{i^*}$ will use only two of the available colors. The key insight to ensure that vertices in the remaining $S_i$ can also be colored, is to split them into multiple copies that each have at most one neighbor in any $T_j$. There will be at most one vertex in the neighborhood of a copy that is colored using color 1 or 2, thereby we can always color it using the other available color. Finally, additional gadgets will ensure that in some $S_i$ all these copies get equal colors, and in some $T_j$ the vertices that correspond to a triangle in the inputs are properly colored as such. With this intuition, we give the construction.

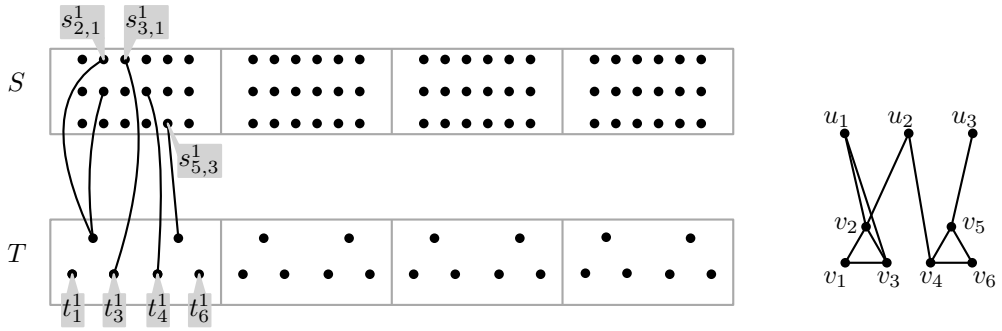
▶ **Theorem 12.** 3-Coloring *parameterized by the number of vertices $n$ does not have a generalized kernel of size $\mathcal{O}(n^{2-\varepsilon})$ for any $\varepsilon > 0$, unless* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

**Proof.** To prove this statement, we give a degree-2 cross-composition from 2-3-Coloring with triangle split decomposition to 3-List Coloring and then show how to change this instance into a 3-Coloring instance. We start by defining a polynomial equivalence relation $\mathcal{R}$ on instances of 2-3-Coloring with triangle split decomposition. Let two instances be equivalent under $\mathcal{R}$, when the sets $U$ have the same size and sets $V$ consist of the same number of triangles. It is easy to verify that $\mathcal{R}$ is a polynomial equivalence relation.

By duplicating one of the inputs several times if needed, we ensure that the number of inputs to the cross-composition is a square. This increases the number of inputs by at most a factor four and does not change the value of the or. Therefore, assume we are given $t$ instances of 2-3-Coloring with Triangle Split Decomposition such that $t' := \sqrt{t}$ is integer. Enumerate these instances as $X_{i,j}$ for $i, j \in [t']$ and let instance $X_{i,j}$ have graph $G_{i,j}$. For input instance $X_{i,j}$, let $U$ and $V$ be such that $U$ is an independent set with $|U| = m$ and $V$ consists of $n$ vertex-disjoint triangles. Enumerate the vertices in $U$ as $u_1, \ldots, u_m$ and in $V$ as $v_1, \ldots, v_{3n}$ such that $v_{3k-2}, v_{3k-1}, v_{3k}$ form a triangle for $k \in [n]$. We now create an instance of the 3-List Coloring problem, consisting of a graph $G'$ together with a list function $L$ that assigns a subset of the color palette $\{1, 2, 3\}$ to each vertex.

Refer to Figure 1 for a sketch of $G'$.

1. Initialize $G'$ as the graph containing $t'$ sets of $m \cdot 3n$ vertices each, called $S_i$ for $i \in [t']$. Label the vertices in each of these sets as $s^i_{k,\ell}$ for $i \in [t']$, $k \in [3n]$ and $\ell \in [m]$. Define $L(s^i_{k,\ell}) := \{1, 2\}$. The vertices $s^i_{1,\ell}, s^i_{2,\ell}, \ldots, s^i_{3n,\ell}$ together represent a single vertex of




**(a)** Constructed graph $G'$ **(b)** Instance $X_{1,1}$

**Figure 1** Construction of graph $G'$ for $t' = 4$, $m = 3$, and $n = 2$. Edges between vertices in $S$ and $T$ are shown for instance $X_{1,1}$. All blocking-gadgets and the vertex sets $A$ and $B$ are left out.

the independent set of an input instance, which is split into copies to ensure that every copy has at most one neighbor in each cell of $T$ (the bottom row in Figure 1a).

2. Add $t'$ sets of $3n$ vertices each, labeled $T_j$ for $j \in [t']$. Label the vertices in $T_j$ as $t^j_k$ for $k \in [3n]$ and let $L(t^j_k) := \{1, 2, 3\}$. Vertices $t^j_{3k-2}, t^j_{3k-1}, t^j_{3k}$ correspond to a triangle in an input graph. They are not connected, so that we can safely color all vertices that do not correspond to a 3-colorable input with color 3.
3. Connect vertex $s^i_{k,\ell}$ to vertex $t^j_k$ if in graph $G_{i,j}$ vertex $u_\ell$ is connected to $v_k$, for $k \in [3n]$ and $\ell \in [m]$. By this construction, the graph $G_{i,j}$ is isomorphic to the graph obtained from $G'[S_i \cup T_j]$ by replacing each triple $t^j_{3k-2}, t^j_{3k-1}, t^j_{3k}$ by a triangle for $k \in [n]$ and merging all $3n$ vertices $s^i_{k,\ell}$ in $S_i$ that have the same value for $\ell \in [m]$.
4. Add vertex sets $A = \{a_1, \ldots, a_{t'}\}$ and $B := \{b_1, \ldots, b_{t'}\}$. These are used to choose indices $i$ and $j$ such that $G_{i,j}$ is 3-colorable. Let $L(a_i) := L(b_i) := \{1, 2\}$ for all $i \in [t']$.
5. Let $\mathbf{c}$ be defined by $c_i := 2$ for all $i \in [t']$. Add a blocking-gadget($\mathbf{c}$) to $G'$. Connect vertex $a_i$ to the distinguished vertex $\pi_i$ of this blocking-gadget for all $i \in [t']$.
6. Let $\mathbf{c}$ again be defined by $c_i := 2$ for all $i \in [t']$. Add a blocking-gadget($\mathbf{c}$) to $G'$. Connect vertex $b_j$ to $\pi_j$ for all $j \in [t']$. Together with the previous step, this ensures that in any proper list coloring at least one vertex in $A$ and at least one vertex in $B$ has color 1.
7. For every $i \in [t']$, $\ell \in [m]$, and $k \in [3n - 1]$, for every $c_1, c_2 \in [2]$ with $c_1 \neq c_2$, add a blocking-gadget($(c_1, c_2, 1)$) to $G'$. Connect $s^i_{k,\ell}$ to $\pi_1$, $s^i_{k+1,\ell}$ to $\pi_2$, and $a_i$ to $\pi_3$. This ensures that when $a_i$ has color 1, vertices $s^i_{k,\ell}$ and $s^i_{k',\ell}$ have the same color for all $k, k' \in [3n]$.
8. For every $j \in [t']$, $k \in [n]$, for every $c_1, c_2, c_3 \in [3]$ that are not all pairwise distinct, add a blocking-gadget($(c_1, c_2, c_3, 1)$) to $G'$. Connect $t^j_{3k-2}$ to $\pi_1$, $t^j_{3k-1}$ to $\pi_2$, $t^j_{3k}$ to $\pi_3$, and $b_j$ to $\pi_4$. This construction ensures that if $b_j$ is colored 1, all "triangles" in $T_j$ are properly colored. If $b_j$ is colored 2 however, the gadgets add no additional restrictions to the coloring of vertices in $T_j$.

This concludes the construction of $G'$; we proceed with the analysis.

▶ **Claim 13.** *Let $c$ be a proper 3-list coloring of $G'$. Then there exists $i \in [t']$ such that for all $\ell \in [m]$ and for all $k, k' \in [3n]$ we have $c(s^i_{k,\ell}) = c(s^i_{k',\ell})$.*

**Proof.** By the blocking-gadget added in Step 5, there exists $i \in [t']$ such that $c(a_i) \neq 2$. Since $L(a_i) = \{1, 2\}$, this implies that $c(a_i) = 1$. We show that $i$ has the required property.

Suppose there exist $k, k' \in [3n]$ and $\ell \in [m]$ such that $c(s^i_{k,\ell}) \neq c(s^i_{k',\ell})$. Then there must also exist $k \in [3n - 1]$ such that $c(s^i_{k,\ell}) \neq c(s^i_{k+1,\ell})$, or else they would all be

equal. Let $(c_1, c_2, c_3)$ correspond to the coloring of $s^i_{k,\ell}$,$s^i_{k+1,\ell}$, and $a_i$ as given by $c$. Then blocking-gadget$((c_1, c_2, c_3))$ was added in Step 7 and connected to these three vertices. But by Lemma 10, it follows that any list-coloring of this blocking-gadget must assign color $c_i$ to some $\pi_i$ for $i \in [3]$. By the way they are connected to $s^i_{k,\ell}$,$s^i_{k+1,\ell}$ and $a_i$, one edge has two endpoints of equal color, which is a contradiction. ◀

We will say a triple of vertices $v_1, v_2, v_3$ is *colorful* (under coloring $c$), if they receive distinct colors, meaning $c(v_1) \neq c(v_2) \neq c(v_3) \neq c(v_1)$.

▶ **Claim 14.** *Let $c$ be a proper 3-list coloring of $G'$. Then there exists $j \in [t']$ such that for all $k \in [n]$ the triple $t^j_{3k}$, $t^j_{3k-1}$, $t^j_{3k-2}$ is colorful.*

**Proof.** By the blocking-gadget added in Step 6, there exists $j \in [t']$ such that $c(b_j) \neq 2$. Since $L(b_j) = \{1, 2\}$, this implies that $c(b_j) = 1$. We show that $j$ has the desired property.

Suppose there exists $k \in [n]$, such that $t^j_{3k}$, $t^j_{3k-1}$, and $t^j_{3k-2}$ are not a colorful triple. Let $(c_1, c_2, c_3, c_4) \in [3]^4$ correspond to the coloring given to $t^j_{3k}$, $t^j_{3k-1}$, $t^j_{3k-1}$, and $b_j$. In Step 8, blocking-gadget$((c_1, c_2, c_3, c_4))$ was added, together with connections to these four vertices. But by Lemma 10, any list-coloring of this blocking-gadget must assign color $c_i$ to some $\pi_i$ for $i \in [4]$. By the way they are connected to $t^j_{3k}$, $t^j_{3k-1}$, $t^j_{3k-2}$, and $b_j$, one edge has two endpoints of equal color, which is a contradiction. ◀

▶ **Claim 15.** *The graph $G'$ is 3-list colorable $\Leftrightarrow$ some input instance $X_{i^*j^*}$ is 2-3-colorable.*

**Proof.** ($\Rightarrow$) Suppose we are given a 3-list coloring $c$ of $G'$. By Lemmas 13 and 14 there exist integers $i^*$ and $j^* \in [t']$ such that for all $\ell \in [m]$ and for all $k, k' \in [3n]$ we have $c(s^{i^*}_{k,\ell}) = c(s^{i^*}_{k',\ell})$ and furthermore for all $k \in [n]$ the triple $t^{j^*}_{3k}$, $t^{j^*}_{3k-1}$, $t^{j^*}_{3k-2}$ is colorful. We show that this implies that $G_{i^*,j^*}$ has a valid 2-3-coloring $c'$, which we define as follows. Let $c'(u_\ell) := c(s^{i^*}_{1,\ell})$ for $\ell \in [m]$ and let $c'(v_k) := c(t^{j^*}_k)$ for $k \in [3n]$. It remains to verify that $c'$ is a valid coloring of $G_{i^*,j^*}$. For any edge $\{u_\ell, v_k\} \in E(G_{i^*,j^*})$ with $\ell \in [m], k \in [3n]$, the endpoints receive different colors since

$$c'(v_k) = c(t^{j^*}_k) \neq c(s^{i^*}_{k,\ell}) = c(s^{i^*}_{1,\ell}) = c'(u_\ell).$$

For an edge $\{v_k, v'_k\} \in G_{i^*,j^*}$, its coloring corresponds to the coloring of $t^{j^*}_k$ and $t^{j^*}_{k'}$, which are colored differently by choice of $j^*$ in Lemma 14. Furthermore, $u_\ell$ is always colored with color 1 or 2 as $L(s^{i^*}_{1,\ell}) = \{1, 2\}$. Thereby, $c'$ is a proper 2-3-coloring of $G_{i^*,j^*}$.

($\Leftarrow$) Suppose $c$ is a 2-3-coloring of $G_{i^*,j^*}$, such that the $U$-partite set of $G_{i^*,j^*}$ is colored using only the colors 1 and 2. We will construct a 3-list coloring $c'$ for graph $G'$. For $\ell \in [m]$ let $c'(s^{i^*}_{k,\ell}) := c(u_\ell)$ for all $k \in [3n]$. For $k \in [3n]$ let $c'(t^{j^*}_k) := c(v_k)$. For $j \neq j^*$ and $k \in [3n]$ let $c'(t^j_k) := 3$. For $i \neq i^* \in [t']$, $k \in [3n]$ and $\ell \in [m]$, pick $c'(s^i_{k,\ell}) \in \{1, 2\} \setminus \{c'(t^{j^*}_k)\}$. Let $c'(a_{i^*}) := 1$ and let $c'(b_{j^*}) := 1$. For $i \neq i^*$ let $c'(a_i) := 2$, similarly for $j \neq j^*$ let $c'(b_j) := 2$. Before coloring the vertices in blocking-gadgets, we will show that $c'$ is proper on $G'[S \cup T]$. This will imply that the coloring defined so far is proper, as vertices in $A$ and $B$ only connect to blocking-gadgets.

Note that all edges in $G'[S \cup T]$ go from $S$ to $T$. Consider an edge $\{s, t\}$ for $s \in S, t \in T$. Since $c'(s) \neq 3$, if $t \in T_j$ for $j \neq j^* \in [t']$, it follows immediately that $c'(s) \neq c'(t)$. Furthermore, if $s \in S_i$ for $i \neq i^* \in [t']$, $c'(s) \neq c'(t)$ by the definition of $c'(s)$. Otherwise, $s \in S_{i^*}$ and $t \in T_{j^*}$ and there exist $\{u, v\} \in E(G_{i^*,j^*})$ such that $c'(s) = c(u)$ and $c'(t) = c(v)$. Since $c$ is a proper coloring, it follows that $c'(s) \neq c'(t)$.

To complete the proof, extend $c'$ to also properly color all blocking-gadgets. This is possible for the blocking-gadgets added in Steps 5 and 6, since $c'(a_{i^*}) = 1$ and $c'(b_{j^*}) = 1$.

Furthermore we show that this is possible for all blocking-gadgets introduced in Step 7. A blocking-gadget$((c_1, c_2, c_3))$ introduced in Step 7 either has $\pi_3$ connected to $a_i$ for $i \neq i^*$ with $c'(a_i) = 2 \neq c_3$, or it is connected to $a_{i^*}$ and in this case the vertices $s^{i^*}_{k,\ell}$ and $s^{i^*}_{k+1,\ell}$ are assigned equal colors and thus at least one of them has a coloring different from the coloring given by $c_1$ and $c_2$ as these colors are distinct. Thus, the colors that are forbidden on vertices $\pi_i$ by the connections to the rest of the graph, do not correspond to $(c_1, c_2, c_3)$ and $c'$ can be extended to color the entire blocking-gadget by Lemma 10.

Similarly, coloring $c'$ can be extended to blocking-gadgets($\mathbf{c}$) added in Step 8, as either $\pi_4$ in the gadget is connected to $b_j$ for $j \neq j^*$ and $c(b_j) = 2 \neq c_4$, or the three vertices from $T$ connected to this gadget are colored with three different colors. ◀

The claim above shows that we have given a cross-composition into 3-List Coloring. To obtain an instance of 3-Coloring, we add a triangle consisting of vertices $\{C_1, C_2, C_3\}$ to the graph. We connect a vertex $v$ in $G'$ to $C_i$ if $i \notin L(v)$ for $i \in [3]$. This graph now has a proper 3-coloring if and only if the original graph had a proper 3-list coloring. Thus, by Claim 15, the resulting 3-Coloring instance acts as the logical or of the inputs.

It remains to bound the number of vertices of $G'$. In Step 1 we add $|S| = m \cdot 3n \cdot t'$ vertices and in Step 2 we add another $|T| = 3n \cdot t'$ vertices. Then in Step 4 we add $|A| + |B| = 2t'$ additional vertices. The two blocking-gadgets added in Steps 5 and 6 each have size $\mathcal{O}(t')$. The blocking-gadgets added in Step 7 have constant size, and we add six of them for each $i \in [t'], \ell \in [m], k \in [3n-1]$, thus adding $\mathcal{O}(t' \cdot m \cdot n)$ vertices. Similarly, the blocking-gadgets added in Step 8 have constant size, and we add a constant number of them for each $j \in [t'], \ell \in [n]$, thus adding $\mathcal{O}(t' \cdot n)$ vertices. This gives a total of $\mathcal{O}(t' \cdot n \cdot m) = \mathcal{O}(\sqrt{t} \cdot (\max_{i,j} |X_{i,j}|)^{\mathcal{O}(1)})$ vertices. Theorem 12 now follows from Theorem 3 and Lemma 11. ◀

The set of all vertices of a graph is always a valid vertex cover for that graph. Thereby, it follows from Theorem 12 that the lower bound also holds when parameterized by vertex cover. In [8, Theorem 3], it was shown that for any $q \geq 4$, $q$-Coloring parameterized by vertex cover does not have a generalized kernel of size $\mathcal{O}(k^{q-1-\varepsilon})$, unless NP $\subseteq$ coNP/poly. Combining these results gives a lower bound for $q$-Coloring that matches the kernel size presented in the first section.

▶ **Corollary 16.** *For any $q \geq 3$, $q$-Coloring parameterized by vertex cover does not have a generalized kernel of bitsize $\mathcal{O}(k^{q-1-\varepsilon})$ for any $\varepsilon > 0$, unless* NP $\subseteq$ coNP/poly.

## 5 Conclusion

We have given a kernel for $q$-Coloring parameterized by Vertex Cover with $\mathcal{O}(k^{q-1})$ vertices and bitsize $\mathcal{O}(k^{q-1} \log k)$, improving on the previously known kernel by almost a factor $k$. Furthermore, 3-Coloring when parameterized by the number of vertices has no kernel of size $\mathcal{O}(n^{2-\varepsilon})$, unless NP $\subseteq$ coNP/poly. It was already known that for $q \geq 4$, $q$-Coloring parameterized by Vertex Cover was unlikely to yield a kernel of size $\mathcal{O}(k^{q-1-\varepsilon})$. Combining these results allows us to give the same lower bound for $q = 3$, under the assumption that NP $\not\subseteq$ coNP/poly. Thereby we have provided an upper and lower bound on the kernel size of $q$-Coloring parameterized by Vertex Cover for any $q \geq 3$, that match up to $k^{o(1)}$ factors.

It is easy to see that the kernel lower bounds also hold for $q$-List Coloring, where every vertex $v$ in the graph has a list $L(v) \subseteq [q]$ of allowed colors. Furthermore, we can also apply our kernel, by first reducing an instance of $q$-List Coloring to an instance of $q$-Coloring

using $q$ additional vertices, and adding these $q$ vertices to the vertex cover of the graph. This only changes the size of the obtained kernel by a constant factor.

In this paper we gave a first example where applying the known results for sparsification of CSPs gives an improved kernel for a graph problem. It would be interesting to see if this technique can be applied to obtain smaller kernels for other graph problems as well. To apply this idea, one needs to first identify which constraints should be modeled. When the constraints are found, they need to be written as equalities of low-degree polynomials over a suitably chosen field. This requires the clever construction of polynomials that have a sufficiently low degree, in order to obtain a good bound on the kernel size.

Another direction for future research consists of obtaining optimal kernel bounds for $q$-Coloring with different structural parameters. For example, one could look at $q$-Coloring parameterized by a modulator to a cograph. This parameterization admits a polynomial kernel [8, Corollary 3], but tight bounds are not known.

## References


1. Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014. doi:10.1137/120880240.
2. 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.
3. Holger Dell and Dániel Marx. Kernelization of packing problems. In *Proc. 23th SODA*, SODA '12, pages 68–81, 2012.
4. Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.
5. 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.
6. Jirí Fiala, Petr A. Golovach, and Jan Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theor. Comput. Sci.*, 412(23):2513–2523, 2011. doi:10.1016/j.tcs.2010.10.043.
7. 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.
8. Bart M. P. Jansen and Stefan Kratsch. Data reduction for graph coloring problems. *Inf. Comput.*, 231:70–88, 2013. doi:10.1016/j.ic.2013.08.005.
9. Bart M. P. Jansen and Astrid Pieterse. Optimal sparsification for some binary csps using low-degree polynomials. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, volume 58 of *LIPIcs*, pages 71:1–71:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.71.
10. Bart M. P. Jansen and Astrid Pieterse. Sparsification upper and lower bounds for graph problems and not-all-equal SAT. *Algorithmica*, 79(1):3–28, 2017. doi:10.1007/s00453-016-0189-9.
11. Stefan Kratsch, Geevarghese Philip, and Saurabh Ray. Point line cover: The easy kernel is essentially tight. *ACM Trans. Algorithms*, 12(3):40:1–40:16, 2016. doi:10.1145/2832912.

# Turing Kernelization for Finding Long Paths in Graph Classes Excluding a Topological Minor*†

**Bart M. P. Jansen[1], Marcin Pilipczuk[‡2], and Marcin Wrochna[§3]**

1 **Eindhoven University of Technology, The Netherlands**
2 **University of Warsaw, Poland**
3 **University of Warsaw, Poland**

## Abstract

The notion of Turing kernelization investigates whether a polynomial-time algorithm can solve an NP-hard problem, when it is aided by an oracle that can be queried for the answers to bounded-size subproblems. One of the main open problems in this direction is whether $k$-Path admits a polynomial Turing kernel: can a polynomial-time algorithm determine whether an undirected graph has a simple path of length $k$, using an oracle that answers queries of size $k^{\mathcal{O}(1)}$?

We show this can be done when the input graph avoids a fixed graph $H$ as a topological minor, thereby significantly generalizing an earlier result for bounded-degree and $K_{3,t}$-minor-free graphs. Moreover, we show that $k$-Path even admits a polynomial Turing kernel when the input graph is not $H$-topological-minor-free itself, but contains a known vertex modulator of size bounded polynomially in the parameter, whose deletion makes it so. To obtain our results, we build on the graph minors decomposition to show that any $H$-topological-minor-free graph that does not contain a $k$-path has a separation that can safely be reduced after communication with the oracle.

**1998 ACM Subject Classification** G.2.2 Graph Theory, F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** Turing kernel, long path, $k$-path, excluded topological minor, modulator

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.23

## 1 Introduction

Suppose that Alice is a polynomial-time agent faced with an input to an NP-hard problem that she wishes to solve exactly. To facilitate her in this process, she can ask questions to an all-knowing oracle. These will be answered truthfully and instantly, but the oracle is memory-less and will not take previous questions into account when answering the next one. How large do these questions have to be, to allow Alice to find the answer to her problem? Clearly, the answer can be established by sending the entire input to the oracle, who determines the answer and sends it to Alice. Could there be a more clever strategy? Alice can attempt to isolate a small but meaningful question about the behavior of her input,

* This work was supported by the Netherlands Organization for Scientific Research (NWO) Veni grant 639.021.437 "Frontiers in Parameterized Preprocessing" and Gravitation grant 024.002.003 "Networks".
† A full version of the paper is available at [18], https://arxiv.org/abs/1707.01797.
‡ Marcin Pilipczuk is supported by the "Recent trends in kernelization: theory and experimental evaluation" project, carried out within the Homing programme of the Foundation for Polish Science co-financed by the European Union under the European Regional Development Fund.
§ Marcin Wrochna is supported by the National Science Centre of Poland grant number 2013/11/D/ST6/03073 and by the Foundation for Polish Science (FNP) via the START stipend programme.




such that after learning its answer, she can reduce to a smaller input without changing the outcome. Iterating this process solves her problem: when it has become sufficiently small, it can be posed to the oracle in its entirety.

Such problem-solving strategies can be rigorously analyzed using the notion of *Turing kernelization* that originated in parameterized algorithmics. The parameter makes it possible to express how the size of the questions that Alice asks, depends on properties of the input that she is given. (See Section 3 for a formal definition.)

Understanding the power of Turing kernelization is one of the main open research horizons in parameterized algorithmics. There is a handful of problems for which a nontrivial Turing kernelization is known [1, 2, 3, 5, 12, 15, 17, 19, 22, 24]. On the other hand, there is a hierarchy of parameterized complexity classes which are conjectured not to admit polynomial Turing kernels [14]. Arguably, the main open problem (cf. [4, 3, 14]) in this direction is to determine whether the $k$-Path problem (determine whether an undirected graph has a simple path of length $k$) has a polynomial Turing kernel. In earlier work [16], the first author showed that $k$-Path indeed admits polynomial Turing kernels on several graph classes. In this work, we develop Turing kernels for $k$-Path in a much more general setting.

**Our results.** Our algorithmic contributions are twofold. First of all, we extend the Turing kernelization for $k$-Path to much broader families of sparse graphs. Whereas the earlier work could only deal with $K_{3,t}$-minor-free graphs, claw-free graphs, and bounded-degree graphs, we show that a Turing kernelization exists on $H$-minor-free graphs for all fixed graphs $H$. We even lift the kernelization to $H$-topological-minor-free graphs, thereby capturing a common generalization of the bounded-degree and $K_{3,t}$-minor-free cases.

▶ **Theorem 1.** *For every fixed graph $H$, the $k$-Path problem, restricted to graphs excluding $H$ as a topological minor, admits a polynomial Turing kernel. Furthermore, the kernel runs in time $k^{\mathcal{O}_H(1)}n^2m$ and invokes $k^{\mathcal{O}_H(1)} \cdot n$ calls to the oracle.*

Our second contribution is the following theorem. By a novel algorithmic approach, we obtain a Turing kernelization even when the input graph does not belong to the desired restricted graph class itself, but contains a small known vertex modulator whose deletion places the graph in such a graph class.

▶ **Theorem 2.** *For every fixed graph $H$, the $k$-Path problem, on instances consisting of a graph $G$, integer $k$, and a modulator $M \subseteq V(G)$ such that $G - M$ is $H$-topological-minor-free, admits a polynomial Turing kernel, when parameterized by $k$ and $|M|$.*

**Techniques.** To explain our approach, we briefly recall the idea behind the Turing kernelization for $k$-Path on planar graphs. At the core lies a win/win: there is a polynomial-time algorithm that either (i) establishes that a planar graph $G$ has a $k$-path (a simple path on $k$ vertices), or (ii) finds a separation $(A, B)$ in $G$ with the following property: the size of $A$ is polynomially bounded in $k$, but large enough that after marking a witness structure for each reasonable way in which a $k$-path might intersect $A$, some vertex remains unmarked. Using bounded-size oracle queries to mark the witness structures, this allows the problem to be simplified by removing an unmarked vertex from $A$ without changing the answer.

Theorem 1 is established by lifting this win/win approach to $H$-(topological)-minor-free graphs. This requires an adaptation of the decomposition theorems of Robertson and Seymour [21] (for minors) and of Grohe and Marx [13] (for topological minors), to obtain the following. Every $H$-free graph that does not have a $k$-path, has a tree decomposition of constant adhesion and width $\text{poly}(k)$. A reducible separation can be found by inspecting

this tree decomposition. To establish this result, we exploit known theorems stating that triconnected $n$-vertex graphs that exclude $K_{3,t}$ as a minor for some $t$ [7], contain paths of length $\Omega(n^{\varepsilon})$ for some $\varepsilon > 0$. Roughly speaking, this allows us to infer the existence of a $k$-path if there is a large embedded part in the nearly-embeddable graph corresponding to a bag of the graph minors decomposition, since graphs embeddable in a fixed surface are $K_{3,t}$-minor-free for some $t$. We use lower bounds on the circumference of graphs of bounded degree [6, 23] to achieve a similar conclusion from the existence of a large bounded-degree bag in the topological-minor-free decomposition. Several technical steps are needed to translate this into the desired win/win, due to the existence of vortices, virtual edges, and the lack of a direct polynomial-time algorithm to compute the decomposition.

To prove Theorem 2, we introduce a new algorithmic tool for finding irrelevant vertices for the $k$-Path problem in the presence of a modulator $M$ in the input graph $G$. Since Theorem 1 can be applied to find a $k$-path in $G - M$ if one exists, the challenge is to detect a $k$-path in $G$ that jumps between $M$ and $G - M$ several times. The absence of a $k$-path in $G - M$ implies it has a tree decomposition of width $\mathrm{poly}(k)$ and constant adhesion. Using Theorem 1 as a subroutine, along with a packing argument, we can compute a vertex set $X$ of size polynomial in $k + |M|$ with the following guarantee. If there is a $k$-path, then there is a *guarded* $k$-path $P$ in which each successive pair of vertices in $M \cap P$ are connected by a subpath through $G - M$ that intersects $X$. Using the tree decomposition of $G - M$, the standard ancestor-marking technique allows us to identify a vertex subset $C$ of $G - (M \cup X)$ that is adjacent to constantly many vertices from $X$. Unless $G$ is already small, we can find such a set $C$ that is sufficiently large to be reducible but small enough that we may invoke the oracle for questions about it. We can then reduce the graph without losing the existence of a guarded $k$-path, by marking a witness for each sensible way in which a constant-size subset from $M$ can connect to prescribed vertices in $X$ through $C$. The fact that $C$ only has constantly many neighbors in $X$ implies that there are only polynomially many relevant choices. We may then safely remove the unmarked vertices.

**Organization.** After preliminaries in Section 2, we give a generic Turing-style reduction rule for $k$-Path in Section 3. In Section 4 we show that an $H$-minor-free graph either has a $k$-path or a separation that is suitable for reduction. In Section 5 we extend this to topological minors. Finally, in Section 6 we present a Turing kernel applicable when the input graph has a small modulator to a suitable graph class. Proofs of statements marked with ♠ can be found in the full version [18].

## 2 Preliminaries

All graphs we consider are finite, simple, and undirected. A *separation* of a graph $G$ is a pair $(A, B)$, $A, B \subseteq V(G)$ such that $A \cup B = V(G)$ and there are no edges between $A \setminus B$ and $B \setminus A$. The *order* of the separation $(A, B)$ is $|A \cap B|$. A graph is *triconnected* if it is connected and cannot be disconnected by deleting fewer than three vertices. When referring to the *size* of a graph in our statements, we mean the number of vertices.

A *tree decomposition* of a graph $G$ is a pair $(T, \mathcal{X})$ where $T$ is a rooted tree and $\mathcal{X}$ is a function that assigns to every node $t \in V(T)$ a subset $\mathcal{X}(t)$ of $V(G)$ called a *bag* such that:

- $\bigcup_{t \in V(T)} \mathcal{X}(t) = V(G)$;
- for each edge $uv \in E(G)$, there is a node $t \in V(T)$ with $u, v \in \mathcal{X}(t)$;
- for each $v \in V(G)$, the nodes $\{t \mid v \in \mathcal{X}(t)\}$ induce a (connected) subtree of $T$.

The *width* of $(T, \mathcal{X})$ is $\max_{t \in V(T)} |\mathcal{X}(t)| - 1$. Its *adhesion* is $\max_{tt' \in E(T)} |\mathcal{X}(t) \cap \mathcal{X}(t')|$. We also call the set $\mathcal{X}(t) \cap \mathcal{X}(t')$ the *adhesion of* $tt'$, for every edge $tt'$ of $T$. For a decomposition $(T, \mathcal{X})$ of $G$ and a node $t \in V(T)$, the *torso*, denoted $\textsc{torso}(G, \mathcal{X}(t))$, is the graph obtained from $G[\mathcal{X}(t)]$ by adding an edge between each pair of vertices in $\mathcal{X}(t) \cap \mathcal{X}(t')$, for every neighbor $t'$ of $t$ in $T$ (so each adhesion induces a clique in the torso). Added edges not present in $G$ are called *virtual edges*. For a subtree $T' \subseteq T$ we write $\mathcal{X}(T')$ for the union $\bigcup_{t \in V(T')} \mathcal{X}(t)$ of bags in $T'$.

For an edge $t_1t_2 \in E(T)$, let $T_i$ be the connected component of $T - \{t_1t_2\}$ that contains $t_i$. Let $V_i = \mathcal{X}(T_i)$. Observe that the properties of a tree decomposition imply that $(V_1, V_2)$ is a separation with $V_1 \cap V_2 = \mathcal{X}(t_1) \cap X(t_2)$.

A decomposition $(T, \mathcal{X})$ is *connected* if for every $t \in V(T)$ and its child $t'$, if $T_{t'}$ is the subtree of $T$ rooted at $t'$, we have (i) that $G[\mathcal{X}(T_{t'}) \setminus \mathcal{X}(t)]$ is connected, and (ii) that $\mathcal{X}(T_{t'}) \setminus \mathcal{X}(t)$ has edges to every vertex of the adhesion $\mathcal{X}(t) \cap \mathcal{X}(t')$. It is straightforward to turn any decomposition into a connected one without increasing its width nor adhesion.

We will also need the following non-standard complexity measure of a tree decomposition $(T, \mathcal{X})$. For every $t \in V(T)$, the number of distinct adhesions $\mathcal{X}(t) \cap \mathcal{X}(t')$ for $t' \in N_T(t)$ is called the *adhesion degree* of $t$. The maximum adhesion degree over all nodes $t$ is the *adhesion degree* of the decomposition $(T, \mathcal{X})$. Observe that if a tree decomposition $(T, \mathcal{X})$ has width less than $\ell$ and adhesions of size at most $h$, then its adhesion degree is at most

$$\sum_{i=0}^{h} \binom{\ell}{i} \leq (1 + \ell)^h.$$

However, in sparse graph classes we can prove a much better bound on the adhesion degree due to linear bounds on the number of cliques in such graphs; cf. Lemma 17.

## 3 Turing kernels

In this section we introduce a general toolbox and notation for our Turing kernel bounds.

### 3.1 Definitions and the auxiliary problem

For a parameterized problem $\Pi$ and a computable function $f$, a *Turing kernel of size* $f$ is an algorithm that solves an input instance $(x, k)$ of $\Pi$ in polynomial time, given access to an oracle that solves instances $(x', k')$ of $\Pi$ with $|x'|, k' \leq f(k)$. A Turing kernel is a *polynomial* one if $f$ is a polynomial.

If we are only interested in distinguishing between NP-complete problems admitting a polynomial Turing kernel from the ones that do not admit such a kernel, we can assume that the oracle solves an arbitrary problem in NP, not necessarily the $k$-Path problem. Indeed, note that by the definition of NP-completeness, an oracle to a problem in NP can be implemented with an oracle to $k$-Path with only polynomial blow-up in the size of the passed instances.

In our work, it will be convenient to reduce to the Auxiliary Linkage problem, defined as follows. The input consists of an undirected graph $G'$, an integer $k'$, a set of terminals $S \subseteq V(G')$, and a number of requests $R_1, R_2, \ldots, R_r$; a request is a set of at most two terminals. A path $P_i$ in $G$ is said to *satisfy a request* $R_i$ if $V(P_i) \cap S = R_i$ and every vertex of $V(P_i) \cap S$ is an endpoint of $P_i$. With such an input, the Auxiliary Linkage problem asks for a sequence of $r$ paths $P_1, P_2, \ldots, P_r$ such that $P_i$ satisfies $R_i$ for every $1 \leq i \leq r$, $|\bigcup_{i=1}^{r} V(P_i)| = k'$, and every vertex of $V(G) \setminus S$ is contained in at most one path $P_i$ (i.e.,

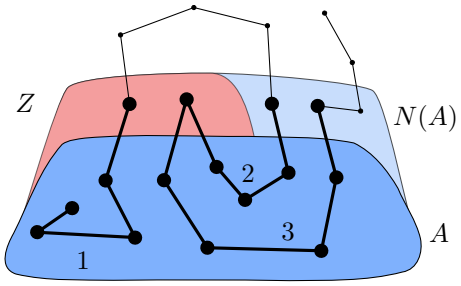



**Figure 1** A set $A$ (blue) and a path with three $A$-traverses (bold). The path is guarded w.r.t. $Z \subseteq N(A)$ (the red set), since each $A$-traverse has an endpoint in it. (A path fully contained in $A$ (with one $A$-traverse) or disjoint from $A$ (with no $A$-traverses) would also be guarded.)

the paths $P_i$ are vertex-disjoint, except that they may share an endpoint, but only if the requests ask them to do so).

We remark that Auxiliary Linkage is a more general problem than $k$-Path: an instance with $G' = G$, $k' = k$, $S = \emptyset$, $r = 1$, and $R_1 = \emptyset$ asks precisely for a $k$-path in $G$.

Clearly, the decision version of the Auxiliary Linkage problem belongs to the class NP. By using its self-reducibility (cf. [16, Lemma 2]), we assume that the oracle returns a sequence of paths $(P_i)_{i=1}^r$ in case of a positive answer. That is, in all subsequent bounds on the number of Auxiliary Linkage oracle calls, the bound adheres to the number of calls to an oracle that returns the actual paths $P_i$; if one wants to use a decision oracle, one should increase the bound by the blow-up implied by the self-reducibility application (i.e., at most $|E(H)|$ for calls on a graph $H$).

## 3.2 Generic reduction rule

We now show a generic reduction rule for the $k$-Path problem. We start with a few definitions.

▶ **Definition 3.** For a graph $G$, a subset $A \subseteq V(G)$, and a simple path $P$ in $G$, an *$A$-traverse* of $P$ is a maximal subpath of $P$ that contains at least one vertex of $A$ and has all its internal vertices in $A$.

Note that if $Q$ is an $A$-traverse of $P$, then every endpoint of $Q$ is either an endpoint of $P$ or lies in $N_G(A)$. See Figure 1.

▶ **Definition 4.** Let $G$ be a graph, $A \subseteq V(G)$, and let $k$ be an integer. A set $Z \subseteq N(A)$ is called a *$k$-guard of $A$* if the following implication holds: if $G$ admits a $k$-path, then there exists a $k$-path $P$ in $G$ that is either contained in $A$ or such that every $A$-traverse of $P$ has at least one endpoint in $Z$.

Given a graph $G$, a set $A \subseteq V(G)$, and a $k$-guard $Z \subseteq N(A)$ of $A$, a $k$-path $P$ satisfying properties as in the above definition is called *guarded* (w.r.t. $k$, $A$, and $Z$). If the integer $k$ and the set $A$ are clear from the context, we call such a set $Z$ simply a guard.

Observe that $Z = N(A)$ is always a guard, but sometimes we will be able to find smaller ones. Of particular interest will be guards of constant size, as our kernel sizes will depend exponentially on the guard size. To describe our single reduction rule, we show how solutions to Auxiliary Linkage can be used to preserve the existence of guarded $k$-paths.

Assume we are given a graph $G$, a set $A \subseteq V(G)$, an integer $k$, and a $k$-guard $Z \subseteq N(A)$ of $A$. Let $h = |Z|$ and $\ell = |N(A)|$. Furthermore, assume that $G$ admits a $k$-path, and let $P$ be a guarded one w.r.t. $A$ and $Z$. Let $(Q_1, Q_2, \ldots, Q_r)$ be the $A$-traverses of $P$, let

$R_i = V(Q_i) \setminus A = V(Q_i) \cap N(A)$ for $1 \leq i \leq r$, let $G' = G[N[A]]$, $S = N(A)$, and let $k' = |\bigcup_{i=1}^{r} V(Q_i)|$. Observe that $(Q_1, Q_2, \ldots, Q_r)$ is a feasible solution to the Auxiliary Linkage instance $\mathcal{I}_P := (G[N[A]], k', S, (R_i)_{i=1}^{r})$; the instance $\mathcal{I}_P$ is henceforth called *induced by $P$ and $A$*. Furthermore, it is easy to see that if $(Q'_1, Q'_2, \ldots, Q'_r)$ is a different feasible solution to $\mathcal{I}_P$, then a path $P'$ obtained from $P$ by replacing every subpath $Q_i$ with $Q'_i$ is also a guarded $k$-path in $G$.

The crucial observation is that a small guard limits the number of $A$-traverses.

▶ **Lemma 5.** *The number $r$ of traverses of the guarded $k$-path $P$ is bounded by* $\max(1, 2|Z|)$.

**Proof.** Every vertex of $Z$ can be an endpoint of at most two traverses. If $r > 1$, then none of the traverses $Q_i$ are contained in $G[A]$, and thus every traverse has at least one endpoint in the guard $Z$. ◀

Lemma 5 in turn limits the number of possible instances $\mathcal{I}$ that can be induced by a guarded $k$-path, for a fixed set $A$ and guard $Z$. Note that we have $0 \leq k' \leq k$ and $0 \leq r \leq \max(1, 2|Z|)$. Furthermore, unless $r = 1$ and $R_1 = \emptyset$, we have $R_i \subseteq N(A)$, $|R_i| \in \{1, 2\}$, and every set $R_i$ needs to have at least one element of $Z$; there are at most $|Z| \cdot (|N(A)| + 1) = h(\ell + 1)$ choices for such a set $R_i$. Consequently, the number of possibilities for the instance $\mathcal{I}$ is at most

$$(k+1) \cdot \left(1 + \sum_{r=0}^{2h} h^r(\ell+1)^r\right) \leq (k+1) \cdot (h(\ell+1))^{2h+1} =: \mathfrak{p}(k, \ell, h). \tag{1}$$

### Reduction rule

If $|A| > k \cdot \mathfrak{p}(k, \ell, h)$, then we can apply the following reduction rule. For each Auxiliary Linkage instance $\mathcal{I}$ out of at most $\mathfrak{p}(k, \ell, h)$ reasonable instances for $A$-traverses of a guarded $k$-path in $G$, we invoke an oracle on the instance $\mathcal{I}$, and mark the vertices of the solution if the oracle finds one. The whole process will mark at most $k \cdot \mathfrak{p}(k, \ell, h) < |A|$ vertices, thus at least one vertex of $|A|$ will remain unmarked. We delete any such vertices.

The observation that on a guarded $k$-path $P$ one can replace a solution to the instance $\mathcal{I}_P$ induced by $P$ and $A$ by a different solution provides safeness of this reduction. Finally, note that the reduction invokes at most $\mathfrak{p}(k, \ell, h)$ calls to the oracle; each call operates on a subgraph of the graph $G[N[A]]$ with $k' \leq k$ and $r \leq 2|Z|$.

We shall apply the Reduction Rule for a medium-sized set $A$ and a guard set $Z$ of constant size formed from adhesions of a tree decomposition. For most of the paper we will use $Z = N(A)$ with $\ell = h = |Z|$ a constant (depending on the excluded (topological) minor, in the results of Sections 4 and 5). Only in Section 6, when dealing with a modulator $M$ such that $G - M$ has an appropriate structure, it will be important to consider $N(A)$ potentially containing all of $M$, with a guard set $Z$ of constant size disjoint from $M$.

## 3.3 Separation oracles

The natural way of using our reduction rule is to find in a graph a large (but not too large) part of the graph with a small (preferably, constant) boundary. Let us first make an abstract definition of an algorithm finding such a separation.

▶ **Definition 6.** For a graph class $\mathcal{G}$, a constant $h$, and a computable coordinate-wise nondecreasing function $q : \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$, an algorithm $\mathcal{S}$ is called a $(h, q, T_{\mathcal{S}})$-*separation oracle* if, given a graph $G \in \mathcal{G}$ and integers $k$ and $p$, in time $T_{\mathcal{S}}(|G|, k, p)$ it finds a separation $(A, B)$ in $G$ of order at most $h$ with $p < |A| \leq q(k, p)$, or correctly concludes that $G$ contains a $k$-path.

For all considered graph classes, we will be able to provide a separation oracle with $q$ being a polynomial. This, in turn, allows the following generic Turing kernel.

▶ **Lemma 7.** *Let $\mathcal{S}$ be a $(h, q, T_{\mathcal{S}})$-separation oracle for a hereditary graph class $\mathcal{G}$. Take $\hat{h} := (2h)^{4h+3}$. Then, the $k$-Path problem restricted to graphs from $\mathcal{G}$ can be solved:*

- *in time $\mathcal{O}\big(T_{\mathcal{S}}(|G|, k, k^2\hat{h}) \cdot |V(G)| \ + \ k\hat{h} \cdot |V(G)| \cdot |E(G)|\big)$,*
- *using at most $k\hat{h} \cdot |V(G)|$ calls to* Auxiliary Linkage,
- *each call on an induced subgraph of the input graph of size at most $q(k, k^2\hat{h})$.*

**Proof.** Let $p = k \cdot \mathfrak{p}(k, h, h) + h \leq k(k+1)(h(h+1))^{2h+1} + h \leq 2k^2(2h)^{4h+2} \leq k^2\hat{h}$.

As long as $|V(G)| > p$, we proceed as follows. Invoke algorithm $\mathcal{S}$ on $G$. If $\mathcal{S}$ claims that $G$ admits a $k$-Path, we simply output the answer yes. Otherwise, let $(A', B')$ be the separation output by $\mathcal{S}$. Apply the Reduction Rule for $k$, $A := A' \setminus B'$, and $Z = N(A) \subseteq A' \cap B'$. Note that as $|Z| \leq h$, the Reduction Rule deletes at least one vertex of $A$. Furthermore, the Reduction Rule invokes at most $\mathfrak{p}(k, h, h) \leq k\hat{h}$ calls to the oracle, each call on an induced subgraph of $G$ of size at most $|A'| \leq q(k, p) \leq q(k, k^2\hat{h})$.

Once we obtain $|V(G)| \leq p$, we solve the instance using a single call to Auxiliary Linkage with $k' = k$, $r = 1$, and $R_1 = \emptyset$. The bounds follow, as there are at most $|V(G)|$ applications of the Reduction Rule, and each call to the oracle takes $\mathcal{O}(|E(G)|)$ time to prepare the instance and parse the output. ◀

Note that for any graph class where separations as in Definition 6 exist, there exists a trivial separation oracle which finds them, running in time $n^{h+\mathcal{O}(1)}$: one iterates over every candidate for $A \cap B$ and, for fixed set $A \cap B$, a straightforward knapsack-type dynamic programming algorithm checks if one can assemble $A \setminus B$ of the desired size from the connected components of $G - (A \cap B)$.

However, this running time bound is unsatisfactory, as it greatly exceeds the number of used oracle calls. For all considered graph classes we prove a much stronger property than just merely the prerequisites of Lemma 7, in particular providing a more efficient separation oracle. We provide necessary definitions in the next section.

## 3.4 Decomposable graph classes

▶ **Definition 8.** For a constant $h$ and a computable nondecreasing function $w : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$, a graph class $\mathcal{G}$ is called $(w, h)$-*decomposable* if for every positive integer $k$ and every $G \in \mathcal{G}$ that does not admit a $k$-path, the graph $G$ admits a tree decomposition of width less than $w(k)$ and adhesions of size at most $h$.

A standard argument shows that in a decomposable graph class, given the decomposition with appropriate parameters, it is easy to provide a separation oracle.

▶ **Lemma 9.** *Assume we are given a graph $G$ and a tree decomposition $(T, \mathcal{X})$ of $G$ of width less than $w$, adhesion at most $h$, and adhesion degree at most $a \geq 2$. Then, given an integer $p$ such that $|V(G)| > p$, one can in time $h^{\mathcal{O}(1)} \cdot (|V(G)| + |E(G)| + |V(T)| + \sum_{t \in V(T)} |\mathcal{X}(t)|)$ find a separation $(A, B)$ of order at most $h$ such that*

$$p < |A| \leq w + p \cdot a.$$

**Proof.** Root the tree $T$ in an arbitrary node, and for $t \in V(T)$ let $T_t$ be the subtree of $T$ rooted in $t$. Let $t_0$ be the lowest node of $T$ such that $|\mathcal{X}(V(T_{t_0}))| > p$; such a node can be computed in linear time in the size of $G$ and $(T, \mathcal{X})$.

Group the children $t'$ of $t_0$ according to their adhesions $\mathcal{X}(t') \cap \mathcal{X}(t_0)$. Due to the bound on the adhesion degree, there are at most $a$ groups. For every adhesion $S$, let $X_{t_0,S}$ be the set of the children of $t_0$ with $S = \mathcal{X}(t') \cap \mathcal{X}(t_0)$. Define $V_S = \bigcup_{t' \in X_{t_0,S}} \mathcal{X}(T_{t'})$.

We consider now two cases. First, assume that $|V_S| \leq p$ for every adhesion $S$. Then, by the adhesion degree bound, we have $|\mathcal{X}(T_{t_0})| \leq |\mathcal{X}(t_0)| + ap \leq w + ap$. Consequently, we can return the separation $(A, B)$ with $A = \mathcal{X}(T_{t_0})$ and $B = \mathcal{X}(T - V(T_{t_0}))$.

In the other case, there exists an adhesion $S$ with $|V_S| > p$. We greedily take a minimal subset $Y_{t_0,S} \subseteq X_{t_0,S}$ such that $V'_S := \bigcup_{t' \in Y_{t_0,S}} \mathcal{X}(T_{t'})$ is of size greater than $p$. By the minimality of $t_0$, for every $t' \in X_{t_0,S}$ we have $|\mathcal{X}(T_{t'})| \leq p$ and, consequently $|V'_S| \leq 2p$. Thus, we can return the separation $(A, B)$ for $A = V'_S$ and $B = N_G[V(G) \setminus V'_S]$, as then $A \cap B \subseteq S$. ◀

A critical insight is that the decomposition used by Cygan et al. [8] to solve the Minimum Bisection problem in fact provides an approximate decomposition in a decomposable graph class. Let us first recall the main technical result of [8].

▶ **Definition 10.** A vertex set $X \subseteq V(G)$ of a graph $G$ is called $(q, h)$-*unbreakable* if every separation $(A, B)$ of order at most $h$ satisfies $|(A \setminus B) \cap X| \leq q$ or $|(B \setminus A) \cap X| \leq q$.

▶ **Theorem 11** ([8])**.** *There is an algorithm that given a graph $G$ and integer $h$ runs in time $2^{\mathcal{O}(h^2)}|V(G)|^2|E(G)|$ and outputs a connected tree decomposition $(T, \mathcal{Y})$ of $G$ such that: (i) for each $t \in V(T)$, the bag $\mathcal{Y}(t)$ is $(2^{\mathcal{O}(h)}, h)$-unbreakable in $G$, and (ii) for each $tt' \in E(T)$ the adhesion $\mathcal{Y}(t) \cap \mathcal{Y}(t')$ has at most $2^{\mathcal{O}(h)}$ vertices and is $(2h, h)$-unbreakable in $G$.*

An unbreakable set can be bounded using only the existence of an appropriate tree decomposition. Thus, the decomposition computed by Theorem 11 approximates the decomposition required in a decomposable graph class:

▶ **Lemma 12** (♠)**.** *Let $G$ be a graph and suppose there exists a decomposition $(T, \mathcal{X})$ of $G$ of width less than $w$, adhesion $h$, and adhesion degree $a$. Let $(T', \mathcal{Y})$ be a tree decomposition of $G$ such that for each $t \in V(T')$, the bag $\mathcal{Y}(t)$ is $(2^{\mathcal{O}(h)}, h)$-unbreakable in $G$. Then $|\mathcal{Y}(t)| \leq w + a \cdot 2^{\mathcal{O}(h)}$.*

▶ **Corollary 13.** *Let $\mathcal{G}$ be a $(w, h)$-decomposable graph class. Then, for every $G \in \mathcal{G}$ and $k \in \mathbb{N}$, one can in $2^{\mathcal{O}(h^2)}|V(G)|^2|E(G)|$ time either correctly conclude that $G$ admits a $k$-path, or find a tree decomposition of $G$ of width at most $(w(k)+1)^{\mathcal{O}(h)}$ and adhesion at most $2^{\mathcal{O}(h)}$.*

Let us now combine all the above. That is, given an integer $k$ and a graph $G$ from a hereditary $(w, h)$-decomposable graph class $\mathcal{G}$, we start by computing the tree decomposition of Corollary 13 (or conclude there is a $k$-path). In general this approximated decomposition has adhesion degree $(w(k) + 1)^{2^{\mathcal{O}(h)}}$. We use this decomposition to find separations of any induced subgraphs of $G$ using the algorithm of Lemma 9 in time $2^{\mathcal{O}(h)}$ times linear in the size of $G$ and the computed decomposition. This gives a $(h, q, T)$-separation oracle with $q(k, p) = p \cdot (w(k) + 1)^{2^{\mathcal{O}(h)}}$ and $T(n, k, p) = 2^{\mathcal{O}(h)} \cdot n \cdot (w(k) + 1)^{\mathcal{O}(h)}$, for any hereditary $(w, h)$-decomposable graph class. By plugging it into Lemma 7, we obtain the following.

▶ **Corollary 14.** *Let $\mathcal{G}$ be a hereditary $(w, h)$-decomposable graph class. Then, the $k$-Path problem, restricted to graphs from $\mathcal{G}$, can be solved in time $2^{2^{\mathcal{O}(h)}}|V(G)|^2|E(G)|$ using $2^{2^{\mathcal{O}(h)}}kn$ calls to* Auxiliary Linkage *on induced subgraphs of the input graph of size $k^2(1+w(k))^{2^{\mathcal{O}(h)}}$.*

We would like to remark that we do not want to claim in this paper the idea that, in the context of $H$-(topological)-minor-free graphs, the decomposition of Theorem 11 should be related to the decomposition of the Global Structure Theorem via an argument as in the proof of Lemma 12. In particular, this observation appeared previously in a work of the second author with Daniel Lokshtanov, Michał Pilipczuk, and Saket Saurabh [20].

## 4 Excluding a minor

In this section we tackle proper minor-closed graph classes, that is, we prove Theorem 1 for graph classes excluding a fixed minor, by proving the following.

▶ **Theorem 15.** *For every fixed graph $H$, the $k$-Path problem restricted to $H$-minor-free graphs can be solved in time $\mathcal{O}_H(n^2m)$ using $\mathcal{O}_H(kn)$ calls to* Auxiliary Linkage *on instances being induced subgraphs of the input graph of size $\mathcal{O}_H(k^{24})$.*

Our main technical result is the following:

▶ **Theorem 16 (♠).** *For every $H$, the class of $H$-minor-free graphs is $(w,h)$-decomposable for $w(k) = \mathcal{O}_H(k^{22})$ and $h = \mathcal{O}_H(1)$.*

By plugging the above into Corollary 14, we obtain the desired polynomial Turing kernel, but with worse bounds than promised by Theorem 15. To obtain better bounds, we need to recall the folklore bound on the adhesion degree in sparse graph classes.

▶ **Lemma 17 (♠).** *Let $G$ be a graph not containing $H$ as a topological minor, and let $(T,\mathcal{X})$ be a connected tree decomposition of $G$ of width less than $\ell$ and adhesion $h$. Then the adhesion degree of $(T,\mathcal{X})$ is bounded by $f(h,H)\cdot\ell$ for some integer $f(h,H)$ depending only on $h$ and $H$.*

This way, we conclude that $H$-minor-free graphs without $k$-paths have tree decompositions of width $\mathcal{O}_H(k^{22})$, adhesion $\mathcal{O}_H(1)$ and adhesion degree $\mathcal{O}_H(k^{22})$, which we can approximate with Theorem 11. Then Theorem 15 follows from Lemma 7 if we find separations applying Lemma 9 to this decomposition.

Thus, it remains to prove Theorem 16. For the proof, we use the graph minors structure theorem, decomposing an $H$-minor-free graph $G$ into parts 'nearly embeddable' in surfaces (precise definitions are given in the full version). By carefully analyzing details of the structure, we either find a large triconnected embedded part, which must contain a long path by the following theorem of Chen et al. [7], or we tighten the graph structure to give a tree decomposition where all parts are small (polynomial in $k$) and adhesions ('boundaries') between them are of constant size.

▶ **Theorem 18** ([7]). *There is a constant $\varepsilon > 0$ such that for every integer $t$, every triconnected graph on $n \geq 3$ vertices embeddable in a surface of (Euler) genus $g$ contains a cycle of length at least $n^{\varepsilon}/2^{(2g+3)^2}$.*

Two intertwined problems that arise with this approach is that torsos of decompositions are not necessarily triconnected, and long paths in them do not necessarily imply long paths in the original graph, because of virtual edges added in torsos. Torsos can be made triconnected if their near-embeddings include cycles or paths around each vortex, but these may use virtual edges in essential ways. On the other hand, the decomposition can be modified so that virtual edges can be replaced with paths in the original graph, but this requires changes that remove virtual edges, hence potentially removing paths around vortices and destroying triconnectedness.

Because of that, we need to go a little deeper and use a local, strong version of the structure theorem from Graph Minors XVII [21]. For the same reason we cannot use existing algorithms for finding the graph minors decompositions. Instead, we only prove the *existence* of a tree decomposition of bounded adhesion, small width, and with nearly embeddable bags.

The statements of [21] (as exposed in [10]) allow us to assume that: (i) any virtual edges in torsos coming from distinct adhesions (from distinct parts of the decomposition) can be

replaced by paths, (ii) every large vortex has a path of at least the same length, (iii) a torso of the decomposition is triconnected, even if each vortex is replaced by a wheel (a cycle plus a universal vertex). These ingredients allow us to use Theorem 18 on this variant of the torso. Thus, if it contains a long path, we prove the original graph must also contain one, otherwise we conclude the bound required in Theorem 16.

## 5 Excluding a topological minor

In this section we tackle graph classes excluding a topological minor, that is, we prove Theorem 1 by proving the following.

▶ **Theorem 19.** *For every fixed graph $H$, the $k$-Path problem restricted to $H$-topological-minor-free graphs can be solved in time $\mathcal{O}_H(n^2m)$ using $\mathcal{O}_H(kn)$ calls to* Auxiliary Linkage *on instances being induced subgraphs of the input graph of size $k^{\mathcal{O}_H(1)}$.*

This follows as before from the following decomposability theorem. Note the exponent in the polynomial bound on width (bag size) now depends on $H$.

▶ **Theorem 20 (♠).** *For every graph $H$, the class of $H$-topological-minor-free graphs is $(w,h)$-decomposable for $w(k) = k^{\mathcal{O}_H(1)}$ and $h = \mathcal{O}_H(1)$.*

To prove the above theorem, we use the structure theorem of Grohe and Marx [13]: when excluding a topological minor, graphs admit a similar structure as for excluding a minor, but apart from nearly embeddable parts, one needs to consider parts that have bounded degree except for a bounded number of vertices. By appropriately contracting everything outside such a part, we obtain a single graph of almost bounded degree. We remove the few vertices of high degree, find the Tutte decomposition into triconnected components, which can be bounded using the following theorem by Shan [23] (see also Chen et al. [6]).

▶ **Theorem 21** ([23])**.** *If $G$ is a triconnected graph with maximum degree at most $\Delta \geq 425$, then $G$ has a cycle of length at least $n^{1/\log_2(\Delta-1)}/4 + 2$.*

Such a small-width Tutte decomposition can then be lifted back to a decomposition of the part we originally considered. Finally, arguments involving tangles allow us to conclude that such a part can be assumed to be small itself. Together with the bound for nearly-embeddable parts, this proves Theorem 20.

## 6 Adding a modulator

In this section we prove Theorem 2 in a more general setting of Section 3. More precisely, Theorem 2 follows directly from the following theorem via Theorems 16 and 20.

▶ **Theorem 22.** *One can solve in polynomial time a given $k$-Path instance $(G,k)$, given access to a set $M \subseteq V(G)$ such that $G - M$ admits a tree decomposition of width less than $w$ and adhesion $h = \mathcal{O}(1)$, and an oracle that solves the* Auxiliary Linkage *problem for instances $(G', k', S, (R_i)_{i=1}^r)$ with $G'$ being a subgraph of $G$, $r, k' \leq k$, $|S| \leq |M| + \mathcal{O}(1)$, and $|V(G')|$ being bounded polynomially in $k$, $w$, and $|M|$.*

Here the exponent in the polynomial bound on the size of oracle calls substantially depends on $h = \mathcal{O}(1)$. Therefore, the result of this section is a purely theoretical result classifying the aforementioned parameterization as admitting a polynomial Turing kernel. Let us sketch how, after approximating a tree decomposition of $G-M$, we find a set $A$ with a guard $Z$ of constant

size, to which we can apply the Reduction Rule. For each $u, v \in M$, we mark up to $k+1$ disjoint $u-v$ paths within $G-M$. We mark all their vertices, the bags that contain them, and the lowest-common-ancestor closure of these bags in the decomposition. In total, this still gives polynomially many marked vertices and polynomially many components of the decomposition between marked bags. Using the tree decomposition we can find a medium sized part $A$ of such a component, with a boundary $Z$ contained in at most two adhesions of the decomposition. Now $N(A) \subseteq Z \cup M$, but additionally it is easy to check that any $A$-traverse of any $k$-path with endpoints $u, v \in M$ can be replaced with a marked $u-v$ path. Hence $Z$ guards $A$.

## 7 Conclusions

We significantly extended the graph classes on which $k$-Path has a polynomial Turing kernel. In addition, we showed that even an instance that does not belong to such a class, but has a small vertex modulator whose deletion makes it so, can be solved efficiently using small queries to an oracle. A subdivision-based argument (cf. [11]) shows that we cannot generalize much beyond $H$-topological-minor-free graphs without settling the problem in general. In particular, the existence of a polynomial Turing kernel for graphs of bounded expansion implies its existence in general graphs.

While our narrative focused on $k$-Path, after small modifications our techniques can also be applied to prove analogues of Theorems 1 and 2 for the $k$-Cycle problem of detecting a simple cycle of length *at least* $k$. The main difficulty in adapting our arguments to $k$-Cycle is the fact that, a priori, the only cycles of length at least $k$ may be arbitrarily much larger than $k$. However, this issue can easily be resolved in the following way. Since a cycle is contained within a single biconnected component, a Turing kernelization can decompose its input into biconnected components and solve the problem independently in each of them. We then start by testing for the existence of a *path* with $k^2$ vertices using the algorithms developed in the paper. If there is a path of length $k^2$ in a biconnected component, then by a classic theorem of Dirac [9] there is a cycle of length at least $k$, and we are done. If no such path exists, then the longest cycle in $G$ has length less than $2k$, and we can continue under the guarantee that the cycle we are looking for has length at least $k$ and less than $2k$. In this setting, our arguments can be easily adapted. In particular, the absence of a path of length $k^2$ implies the existence of suitable tree decompositions from which reducible separations can be extracted.

A significant portion of the technical work in this paper was devoted to modifying the graph minors decomposition to obtain the win/win that either answers the problem or finds a reducible separation. In this way, the algorithmic question has driven a challenging graph-theoretic project. It would be interesting to find more problems amenable to such an approach. We conclude with some concrete open problems. Does $k$-Path have a polynomial Turing kernel on chordal graphs? How about Induced or Directed $k$-Path, on planar graphs?

## References

1 Abhimanyu M. Ambalath, Radheshyam Balasundaram, Chintan Rao H., Venkata Koppula, Neeldhara Misra, Geevarghese Philip, and M. S. Ramanujan. On the kernelization complexity of colorful motifs. In Venkatesh Raman and Saket Saurabh, editors, *Parameterized and Exact Computation - 5th International Symposium, IPEC 2010, Chennai, India, December 13-15, 2010. Proceedings*, volume 6478 of *Lecture Notes in Computer Science*, pages 14–25. Springer, 2010. doi:10.1007/978-3-642-17493-3_4.

2 Florian Barbero, Christophe Paul, and MichałPilipczuk. Exploring the complexity of layout parameters in tournaments and semi-complete digraphs. In *Proc. 44th ICALP*, 2017. In press.

3 Daniel Binkele-Raible, Henning Fernau, Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. *ACM Trans. Algorithms*, 8(4):38:1–38:19, 2012. doi:10.1145/2344422.2344428.

4 Hans L. Bodlaender, Erik D. Demaine, Michael R. Fellows, Jiong Guo, Danny Hermelin, Daniel Lokshtanov, Moritz Müller, Venkatesh Raman, Johan van Rooij, and Frances A. Rosamond. Open problems in parameterized and exact computation - IWPEC 2008. Technical Report UU-CS-2008-017, Utrecht University, 2008.

5 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014.

6 Guantao Chen, Zhicheng Gao, Xingxing Yu, and Wenan Zang. Approximating longest cycles in graphs with bounded degrees. *SIAM J. Comput.*, 36(3):635–656, 2006.

7 Guantao Chen, Xingxing Yu, and Wenan Zang. The circumference of a graph with no $K_{3,t}$-minor, II. *J. Comb. Theory, Ser. B*, 102(6):1211–1240, 2012.

8 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Minimum bisection is fixed parameter tractable. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 323–332. ACM, 2014. doi:10.1145/2591796.2591852.

9 G. A. Dirac. Some theorems on abstract graphs. *Proceedings of the London Mathematical Society*, s3-2(1):69–81, 1952. doi:10.1112/plms/s3-2.1.69.

10 Jan-Oliver Fröhlich and Theodor Müller. Linear connectivity forces large complete bipartite minors: An alternative approach. *J. Comb. Theory, Ser. B*, 101(6):502–508, 2011. doi:10.1016/j.jctb.2011.02.002.

11 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. *J. Comput. Syst. Sci.*, 84:219–242, 2017. doi:10.1016/j.jcss.2016.09.002.

12 Valentin Garnero and Mathias Weller. Parameterized certificate dispersal and its variants. *Theor. Comput. Sci.*, 622:66–78, 2016. doi:10.1016/j.tcs.2016.02.001.

13 Martin Grohe and Dániel Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. *SIAM J. Comput.*, 44(1):114–159, 2015. doi:10.1137/120892234.

14 Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. A completeness theory for polynomial (turing) kernelization. *Algorithmica*, 71(3):702–730, 2015. doi:10.1007/s00453-014-9910-8.

15 Falk Hüffner, Christian Komusiewicz, and Manuel Sorge. Finding highly connected subgraphs. In *Proc. 41st SOFSEM*, pages 254–265, 2015.

16 Bart M. P. Jansen. Turing kernelization for finding long paths and cycles in restricted graph classes. *J. Comput. Syst. Sci.*, 85:18–37, 2017. doi:10.1016/j.jcss.2016.10.008.

17 Bart M. P. Jansen and Dániel Marx. Characterizing the easy-to-find subgraphs from the viewpoint of polynomial-time algorithms, kernels, and turing kernels. 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 616–629. SIAM, 2015. doi:10.1137/1.9781611973730.42.

18 Bart M. P. Jansen, Marcin Pilipczuk, and Marcin Wrochna. Turing kernelization for finding long paths in graph classes excluding a topological minor. *ArXiv e-prints*, 2017. arXiv:1707.01797.

**19** Sudeshna Kolay and Fahad Panolan. Parameterized algorithms for deletion to (r, ell)-graphs. In Prahladh Harsha and G. Ramalingam, editors, *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, volume 45 of *LIPIcs*, pages 420–433. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.420.

**20** Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Manuscript, 2017.

**21** Neil Robertson and Paul D. Seymour. Graph minors: XVII. taming a vortex. *J. Comb. Theory, Ser. B*, 77(1):162–210, 1999. doi:10.1006/jctb.1999.1919.

**22** Alexander Schäfer, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Parameterized computational complexity of finding small-diameter subgraphs. *Optimization Letters*, 6(5):883–891, 2012. doi:10.1007/s11590-011-0311-5.

**23** Songling Shan. *Homeomorphically Irreducible Spanning Trees, Halin Graphs, and Long Cycles in 3-connected Graphs with Bounded Maximum Degrees*. PhD thesis, Georgia State University, 2015. URL: http://scholarworks.gsu.edu/math_diss/23/.

**24** Stéphan Thomassé, Nicolas Trotignon, and Kristina Vuskovic. A polynomial Turing-kernel for weighted independent set in bull-free graphs. In *Proc. 40th WG*, pages 408–419. Springer, 2014.

# An Exponential Lower Bound for Cut Sparsifiers in Planar Graphs*

**Nikolai Karpov†[1], Marcin Pilipczuk[2], and Anna Zych-Pawlewicz[3]**

1 **St. Petersburg Department of V.A. Steklov Institute of Mathematics of the Russian Academy of Sciences, St. Petersburg, Russia and Institute of Informatics, University of Warsaw, Warsaw, Poland**
kimaska@gmail.com.

2 **Institute of Informatics, University of Warsaw, Warsaw, Poland**
malcin@mimuw.edu.pl

3 **Institute of Informatics, University of Warsaw, Warsaw, Poland**
anka@mimuw.edu.pl

## Abstract

Given an edge-weighted graph $G$ with a set $Q$ of $k$ terminals, a *mimicking network* is a graph with the same set of terminals that exactly preserves the sizes of minimum cuts between any partition of the terminals. A natural question in the area of graph compression is to provide as small mimicking networks as possible for input graph $G$ being either an arbitrary graph or coming from a specific graph class.

In this note we show an exponential lower bound for cut mimicking networks in planar graphs: there are edge-weighted planar graphs with $k$ terminals that require $2^{k-2}$ edges in any mimicking network. This nearly matches an upper bound of $\mathcal{O}(k2^{2k})$ of Krauthgamer and Rika [SODA 2013, arXiv:1702.05951] and is in sharp contrast with the $\mathcal{O}(k^2)$ upper bound under the assumption that all terminals lie on a single face [Goranci, Henzinger, Peng, arXiv:1702.01136]. As a side result we show a hard instance for the double-exponential upper bounds given by Hagerup, Katajainen, Nishimura, and Ragde [JCSS 1998], Khan and Raghavendra [IPL 2014], and Chambers and Eppstein [JGAA 2013].

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** mimicking networks, planar graphs

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.24

## 1 Introduction

One of the most popular paradigms when designing effective algorithms is preprocessing. These days in many applications, in particular mobile ones, even though fast running time is desired, the memory usage is the main limitation. The preprocessing needed for such applications is to reduce the size of the input data prior to some resource-demanding computations, without (significantly) changing the answer to the problem being solved. In this work we focus on this kind of preprocessing, known also as graph compression, for flows

* This research is a part of projects that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreements No 714704 (Marcin Pilipczuk and Anna Zych-Pawlewicz).

† Nikolai Karpov has been supported by the Warsaw Centre of Mathematics and Computer Science and the Government of the Russian Federation (grant 14.Z50.31.0030).





and cuts. The input graph needs to be compressed while preserving its essential flow and cut properties.

Central to our work is the concept of a *mimicking network*, introduced by Hagerup, Katajainen, Nishimura, and Ragde [6]. Let $G$ be an edge-weighted graph with a set $Q \subseteq V(G)$ of $k$ terminals. For a partition $Q = S \uplus \bar{S}$, a minimum cut between $S$ and $\bar{S}$ is called a *minimum $S$-separating cut*. A *mimicking network* is an edge-weighted graph $G'$ with $Q \subseteq V(G')$ such that the weights of minimum $S$-separating cuts are equal in $G$ and $G'$ for every partition $Q = S \uplus \bar{S}$. Hagerup et al [6] observed the following simple preprocessing step: if two vertices $u$ and $v$ are always on the same side of the minimum cut between $S$ and $\bar{S}$ for every choice of the partition $Q = S \uplus \bar{S}$, then they can be merged without changing the size of any minimum $S$-separating cut. Such a procedure always leads to a mimicking network with at most $2^{2^k}$ vertices.

The above upper bound can be improved to a still double-exponential bound of roughly $2^{\binom{k-1}{\lfloor (k-1)/2 \rfloor}}$, as observed both by Khan and Raghavendra [7] and by Chambers and Eppstein [2]. In 2013, Krauthgamer and Rika [10] observed that the aforementioned preprocessing step can be adjusted to yield a mimicking network of size $\mathcal{O}(k^2 2^{2k})$ for planar graphs. Furthermore, they introduced a framework for proving lower bounds, and showed that there are (non-planar) graphs, for which any mimicking network has $2^{\Omega(k)}$ edges; a slightly stronger lower bound of $2^{(k-1)/2}$ has been shown by Khan and Raghavendra [7]. On the other hand, for planar graphs the lower bound of [10] is $\Omega(k^2)$. Furthermore, the planar graph lower bound applies even in the special case when all the terminals lie on the same face.

Very recently, two improvements upon these results for planar graphs have been announced. In a sequel paper, Krauthgamer and Rika [11] improve the polynomial factor in the upper bound for planar graphs to $\mathcal{O}(k2^{2k})$ and show that the exponential dependency actually adheres only to the *number of faces containing terminals*: if the terminals lie on $\gamma$ faces, one can obtain a mimicking network of size $\mathcal{O}(\gamma 2^{2\gamma} k^4)$. In a different work, Goranci, Henzinger, and Peng [5] showed a tight $\mathcal{O}(k^2)$ upper bound for mimicking networks for planar graph with all terminals on a single face.

## Our results

We complement these results by showing an exponential lower bound for mimicking networks in planar graphs.

▶ **Theorem 1.1.** *For every integer $k \geq 3$, there exists a planar graph $G$ with a set $Q$ of $k$ terminals and edge cost function under which every mimicking network for $G$ has at least $2^{k-2}$ edges.*

This nearly matches the upper bound of $\mathcal{O}(k2^{2k})$ of Krauthgamer and Rika [11] and is in sharp contrast with the polynomial bounds when the terminals lie on a constant number of faces [5, 11]. Note that it also nearly matches the improved bound of $\mathcal{O}(\gamma 2^{2\gamma} k^4)$ for terminals on $\gamma$ faces [11], as $k$ terminals lie on at most $k$ faces.

As a side result, we also show a hard instance for mimicking networks in general graphs.

▶ **Theorem 1.2.** *For every integer $k \geq 1$, there exists a graph $G$ with a set $Q$ of $3k+1$ terminals and $2^{2^{\Omega(k)}}$ vertices such that no two vertices can be identified without strictly increasing the size of some minimum $S$-separating cut.*

The example of Theorem 1.2, obtained by essentially reiterating the construction of Krauthgamer and Rika [10], shows that the doubly exponential bound is natural for the preprocessing step of Hagerup et al [6], and one needs different techniques to improve upon it.

### Related work

Apart from the aforementioned work on mimicking networks [5, 6, 7, 10, 11], there has been substantial work on preserving cuts and flows approximately, see e.g. [1, 4, 12]. If one wants to construct mimicking networks for vertex cuts in unweighted graphs with deletable terminals (or with small integral weights), the representative sets approach of Kratsch and Wahlström [8] provides a mimicking network with $\mathcal{O}(k^3)$ vertices, improving upon a previous quasipolynomial bound of Chuzhoy [3].

We prove Theorem 1.1 in Section 2 and show the example of Theorem 1.2 in Section 3.

## 2 Exponential lower bound for planar graphs

In this section we present the main result of the paper. We provide a construction that proves that there are planar graphs with $k$ terminals whose mimicking networks are of size $\Omega(2^k)$.

In order to present the desired graph, for the sake of simplicity, we describe its dual graph $(G, c)$. We let $Q = \{f_n, f_s, f_1, f_2, \ldots, f_{k-2}\}$ be the set of faces in $G$ corresponding to terminals in the primal graph $G^*$.[1] There are two special terminal faces $f_n$ and $f_s$, referred to as the north face and the south face. The remaining faces of $Q$ are referred to as equator faces.

A set $S \subset Q$ is *important* if $f_n \in S$ and $f_s \notin S$. Note that there are $2^{k-2}$ important sets; in what follows we care only about minimum cuts in the primal graph for separations between important sets and their complements. For an important set $S$, we define its *signature* as a bit vector $\chi(S) \in [2]^{|Q|-2}$ whose $i$'th position is defined as $\chi(S)[i] = 1$ iff $f_i \in S$. Graph $G$ will be composed of $2^{k-2}$ cycles referred to as important cycles, each corresponding to an important subset $S \subset Q$. A cycle corresponding to $S$ is referred to as $\mathcal{C}_{\chi(S)}$ and it separates $S$ from $\overline{S}$. Topologically, we draw the equator faces on a straight horizontal line that we call the equator. We put the north face $f_n$ above the equator and the south face $f_s$ below the equator. For any important $S \subset Q$, in the plane drawing of $G$ the corresponding cycle $\mathcal{C}_{\chi(S)}$ is a curve that goes to the south of $f_i$ if $f_i \in S$ and otherwise to the north of $f_i$. We formally define important cycles later on, see Definition 2.1.

We now describe in detail the construction of $G$. We start with a graph $H$ that is almost a tree, and then embed $H$ in the plane with a number of edge crossings, introducing a new vertex on every edge crossing. The graph $H$ consists of a complete binary tree of height $k-2$ with root $v$ and an extra vertex $w$ that is adjacent to the root $v$ and every one of the $2^{k-2}$ leaves of the tree. In what follows, the vertices of $H$ are called *branching vertices*, contrary to *crossing vertices* that will be introduced at edge crossings in the plane embedding of $H$.

To describe the plane embedding of $H$, we need to introduce some notation of the vertices of $H$. The starting point of our construction is the edge $e = \{w, v\}$. Vertex $v$ is the first branching vertex and also the root of $H$. In vertex $v$, edge $e$ branches into $e_0 = \{v, v_0\}$ and $e_1 = \{v, v_1\}$. Now $v_0$ and $v_1$ are also branching vertices. The branching vertices are partitioned into layers $L_0, \ldots, L_{k-2}$. Vertex $v$ is in layer $L_0 = \{v\}$, while $v_0$ and $v_1$ are in layer $L_1 = \{v_0, v_1\}$. Similarly, we partition edges into layers $\mathcal{E}^H_0, \ldots \mathcal{E}^H_{k-1}$. So far we have $\mathcal{E}^H_0 = \{e\}$ and $\mathcal{E}^H_1 = \{e_0, e_1\}$.

[1] Since the argument mostly operates on the dual graph, for notational simplicity, we use regular symbols for objects in the dual graph, e.g., $G$, $c$, $f_i$, while starred symbols refer to the dual of the dual graph, that is, the primal graph.

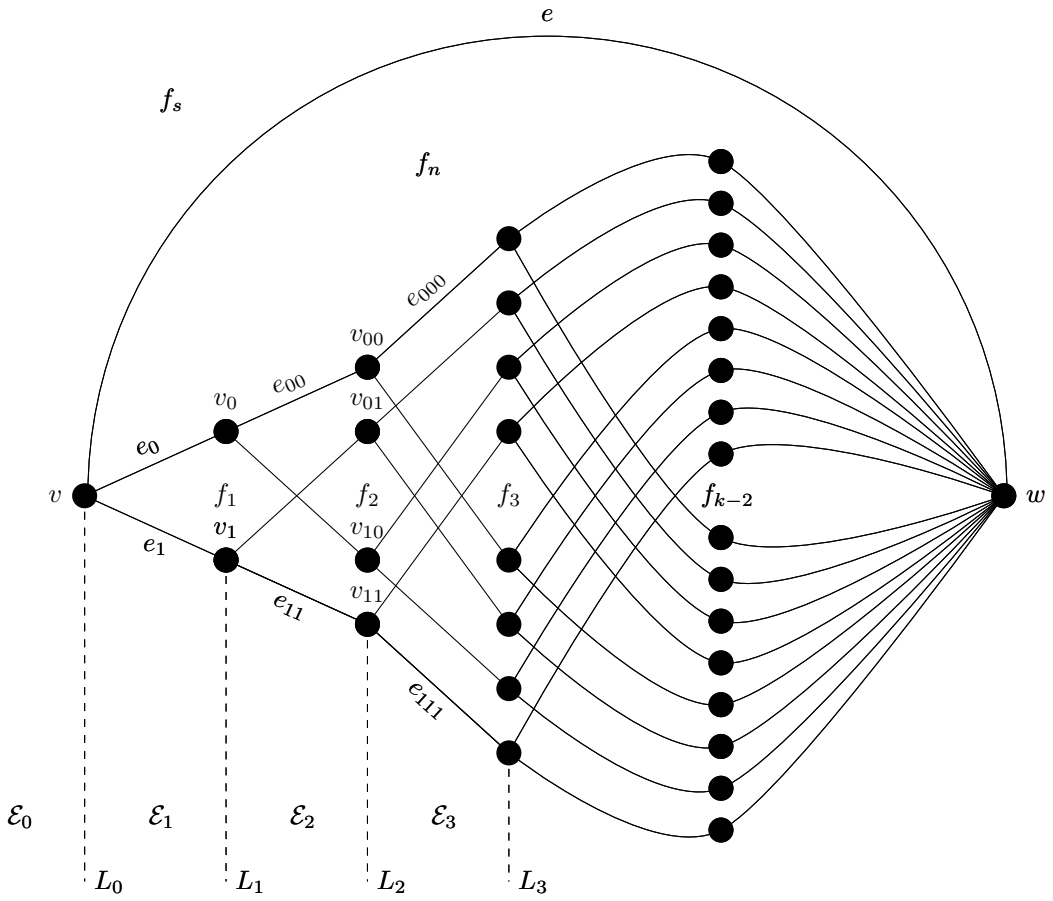



**Figure 1** The graph $G$.

The construction continues as follows. For any layer $L_i, i \in \{1, \ldots, k-3\}$, all the branching vertices of $L_i = \{v_{00\ldots0} \ldots v_{11\ldots1}\}$ are of degree 3. In a vertex $v_a \in L_i$, $a \in [2]^i$, edge $e_a \in \mathcal{E}_i^H$ branches into edges $e_{0a} = \{v_a, v_{0a}\}, e_{1a} = \{v_a, v_{1a}\} \in \mathcal{E}_{i+1}^H$, where $v_{0a}, v_{1a} \in L_{i+1}$. We emphasize here that the new bit in the index is added *as the first symbol*. Every next layer is twice the size of the previous one, hence $|L_i| = |\mathcal{E}_i^H| = 2^i$. Finally the vertices of $L_{k-2}$ are all of degree 2. Each of them is connected to a vertex in $L_{k-3}$ via an edge in $\mathcal{E}_{k-2}^H$ and to the vertex $w$ via an edge in $\mathcal{E}_{k-1}^H$.

We now describe the drawing of $H$, that we later make planar by adding crossing vertices, in order to obtain the graph $G$. As we mentioned before, we want to draw equator faces $f_1, \ldots f_{k-2}$ in that order from left to right on a horizontal line (referred to as an equator). Consider equator face $f_i$ and vertex layer $L_i$ for some $i > 0$. Imagine a vertical line through $f_i$ perpendicular to the equator, and let us refer to it as an $i$'th meridian. We align the vertices of $L_i$ along the $i$'th meridian, from the north to the south. We start with the vertex of $L_i$ with the (lexicographically) lowest index, and continue drawing vertices of $L_i$ more and more to the south while the indices increase. Moreover, the first half of $L_i$ is drawn to the north of $f_i$, and the second half to the south of $f_i$. Every edge of $H$, except for $e$, is drawn as a straight line segment connecting its endpoints. The edge $e$ is a curve encapsulating the north face $f_n$ and separating it from $f_s$-the outer face of $G$.

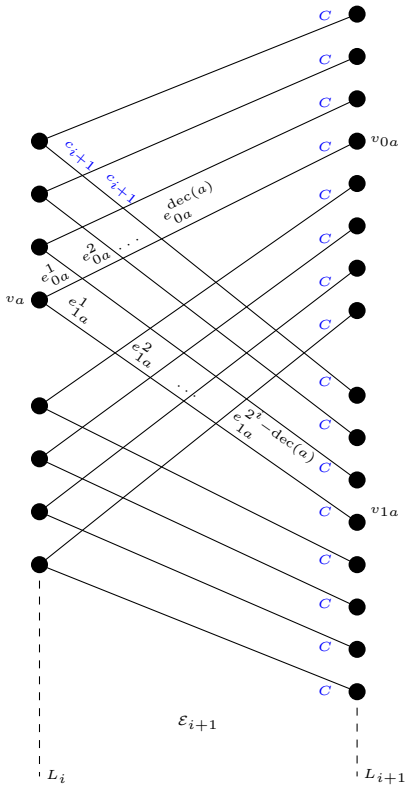



**Figure 2** The layer $\mathcal{E}_{i+1}$. The vertex and edge names are black, their weights are blue.

The crossing vertices are added whenever the line segments cross. This way the edges of $H$ are subdivided and the resulting graph is denoted by $G$. This completes the description of the structure and the planar drawing of $G$. We refer to Figure 1 for an illustration of the graph $G$. The set $\mathcal{E}_i$ consists of all edges of $G$ that are parts of the (subdivided) edges of $\mathcal{E}_i^H$ from $H$, see Figure 2. We are also ready to define important cycles formally.

▶ **Definition 2.1.** Let $S \subset Q$ be important. Let $\pi$ be a unique path in the binary tree $H - \{w\}$ from the root $v$ to $v_{\overleftarrow{\chi(S)}}$, where $\overleftarrow{\cdot}$ operator reverses the bit vector. Let $\pi'$ be the path in $G$ corresponding to $\pi$. The important cycle $\mathcal{C}_{\chi(S)}$ is composed of $e$, $\pi'$, and an edge in $\mathcal{E}_{k-1}$ adjacent to $v_{\overleftarrow{\chi(S)}}$.

We now move on to describing how weights are assigned to the edges of $G$. The costs of the edges in $G$ admit $k-1$ values: $c_1, c_2, \ldots c_{k-2}$, and $C$. Let $c_{k-2} = 1$. For $i \in \{1 \ldots k-3\}$ let $c_i = \sum_{j=i+1}^{k-2} |\mathcal{E}_j| c_j$. Let $C = \sum_{j=1}^{k-2} |\mathcal{E}_i| c_i$. Let us consider an arbitrary edge $e_{ba} = \{v_a, v_{ba}\}$ for some $a \in [2]^i, i \in \{0 \ldots k-3\}, b \in \{0,1\}$ (see Figure 2 for an illustration). As we mentioned before, $e_{ba}$ is subdivided by crossing vertices into a number of edges. If $b = 0$, then edge $e_{ba}$ is subdivided by[2] $\mathrm{dec}(a)$ crossing vertices into $\mathrm{dec}(a)+1$ edges: $e_{ba}^1 = \{v_a, x_{ba}^1\}, e_{ba}^2 = \{x_{ba}^1, x_{ba}^2\} \ldots e_{ba}^{\mathrm{dec}(a)+1} = \{x_{ba}^{\mathrm{dec}(a)}, v_{ba}\}$. Among those edges $e_{ba}^{\mathrm{dec}(a)+1}$ is assigned cost $C$, and the remaining edges subdividing $e_{ba}$ are assigned cost $c_i$. Analogically, if $b = 1$, then edge $e_{ba}$ is subdivided by $2^i - 1 - \mathrm{dec}(a)$ crossing vertices into $2^i - \mathrm{dec}(a)$ edges: $e_{ba}^1 = \{v_a, x_{ba}^1\}, e_{ba}^2 = \{x_{ba}^1, x_{ba}^2\} \ldots e_{ba}^{2^i - \mathrm{dec}(a)} = \{x_{ba}^{2^i - 1 - \mathrm{dec}(a)}, v_{ba}\}$. Again, we let

[2] For a bit vector $a$, $\mathrm{dec}(a)$ denotes the integral value of $a$ read as a number in binary.

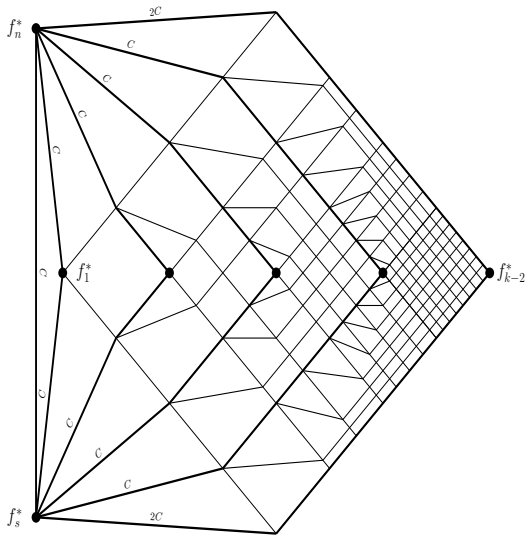



**Figure 3** Primal graph $G^*$.

edge $e_{ba}^{2^i-\mathrm{dec}(a)}$ have cost $C$, and the remaining edges subdividing $e_{ba}$ are assigned cost $c_i$. Finally, all the edges connecting the vertices of the last layer with $w$ have weight $c_{k-2} = 1$. The cost assignment within an edge layer is presented in Figure 2.

This finishes the description of the dual graph $G$. We now consider the primal graph $G^*$ with the set of terminals $Q^*$ consisting of the $k$ vertices of $G^*$ corresponding to the faces $Q$ of $G$. In the remainder of this section we show that there is a cost function on the edges of $G^*$, under which any mimicking network for $G^*$ contains at least $2^{k-2}$ edges. This cost function is in fact a small perturbation of the edge costs implied by the dual graph $G$.

In order to accomplish this, we use the framework introduced in [10]. In what follows, $\mathrm{mincut}_{G,c}(S, S')$ stands for the minimum cut separating $S$ from $S'$ in a graph $G$ with cost function $c$. Below we provide the definition of the cutset-edge incidence matrix and the Main Technical Lemma from [10].

▶ **Definition 2.2** (Incidence matrix between cutsets and edges). Let $(G, c)$ be a $k$-terminal network, and fix an enumeration $S_1, \ldots S_m$ of all $2^{k-1} - 1$ distinct and nontrivial bipartitions $Q = S_i \cup \overline{S}_i$. The cutset-edge incidence matrix of $(G, c)$ is the matrix $A_{G,c} \in \{0, 1\}^{m \times E(G)}$ given by

$$(A_{G,c})_{i,e} = \begin{cases} 1 \text{ if } e \in \mathrm{mincut}_{G,c}(S_i, \overline{S}_i) \\ 0 \text{ otherwise.} \end{cases}$$

▶ **Lemma 2.3** (Main Technical Lemma of [10]). *Let $(G, c)$ be a $k$-terminal network. Let $A_{G,c}$ be its cutset-edge incidence matrix, and assume that for all $S \subset Q$ the minimum $S$-separating cut of $G$ is unique. Then there is for $G$ an edge cost function $\tilde{c} : E(G) \mapsto \mathbb{R}^+$, under which every mimicking network $(G', c')$ satisfies $|E(G')| \geq \mathrm{rank}(A_{G,c})$.*

Recall that $G^*$ is the dual graph to the graph $G$ that we constructed. By slightly abusing the notation, we will use the cost function $c$ defined on the dual edges also on the corresponding primal edges. Let $Q^* = \{{f_n}^*, {f_s}^*, {f_1}^*, \ldots {f_{k-2}}^*\}$ be the set of terminals in $G^*$ corresponding to $f_n, f_s, f_1, \ldots f_{k-2}$ respectively. We want to apply Lemma 2.3 to $G^*$ and $Q^*$. For that we need to show that the cuts in $G^*$ corresponding to important sets are unique and that $\mathrm{rank}(A_{G^*,c})$ is high.

As an intermediate step let us argue that the following holds.

▶ **Claim 1.** *There are $k$ edge disjoint simple paths in $G^*$ from ${f_n}^*$ to ${f_s}^*$: $\pi_0, \pi_1, \ldots, \pi_{k-2}$, $\pi_{k-1}$. Each $\pi_i$ is composed entirely of edges dual to the edges of $\mathcal{E}_i$ whose cost equals $C$. For $i \in \{1 \ldots k-2\}$, $\pi_i$ contains vertex ${f_i}^*$. Let $\pi_i^n$ be the prefix of $\pi_i$ from ${f_n}^*$ to ${f_i}^*$ and $\pi_i^s$ be the suffix from ${f_i}^*$ to ${f_s}^*$. The number of edges on $\pi_i$ is $2^i$, and the number of edges on $\pi_i^n$ and $\pi_i^s$ is $2^{i-1}$.*

**Proof.** The primal graph $G^*$ together with paths $\pi_0, \pi_1 \ldots \pi_{k-2}, \pi_{k-1}$ is pictured in Figure 3. The paths $\pi_{k-2}, \pi_{k-1}$ visit the same vertices in the same manner, so for the sake of clarity only one of these paths is shown in the picture. This proof contains a detailed description of these paths and how they emerge from in the dual graph $G$.

Consider a layer $L_i$. Recall that for any $ba \in [2]^i$ edge $e_{ba}$ of the almost tree is subdivided in $G$, and all the resulting edges are in $\mathcal{E}_i$. If $b = 0$, then edge $e_{ba}$ is subdivided by $\mathrm{dec}(a)$ crossing vertices into $\mathrm{dec}(a)+1$ edges: $e^1_{ba} = \{v_a, x^1_{ba}\}, e^2_{ba} = \{x^1_{ba}, x^2_{ba}\} \ldots e^{\mathrm{dec}(a)+1}_{ba} = \{x^{\mathrm{dec}(a)}_{ba}, v_{ba}\}$, where $c(e^{\mathrm{dec}(a)+1}_{ba}) = C$. Analogically, if $b = 1$, then edge $e_{ba}$ is subdivided by $2^i - 1 - \mathrm{dec}(a)$ crossing vertices into $2^i - \mathrm{dec}(a)$ edges: $e^1_{ba} = \{v_a, x^1_{ba}\}, e^2_{ba} = \{x^1_{ba}, x^2_{ba}\} \ldots e^{2^i-\mathrm{dec}(a)}_{ba} = \{x^{2^i-1-\mathrm{dec}(a)}_{ba}, v_{ba}\}$. Again, $c(e^{2^i-\mathrm{dec}(a)}_{ba}) = C$. Consider the edges of $\mathcal{E}_i$ incident to vertices in $L_i$. If we order these edges lexicographically by their lower index, then each consecutive pair of edges shares a common face. Moreover, the first edge $e^1_{00\ldots0}$ is incident to $f_n$ and the last edge $e^1_{11\ldots1}$ is incident to $f_s$. This gives a path $\pi_i$ from $f_n$ to $f_s$ through $f_i$ in the primal graph where all the edges on $\pi_i$ have cost $C$. Path $\pi_{k-1}$ is given by the edges of $\mathcal{E}_{k-1}$ in a similar fashion and path $\pi_0$ is composed of a single edge dual to $e$. ◀

We move on to proving that the condition in Lemma 2.3 holds. We extend the notion of important sets $S \subseteq Q$ to sets $S^* \subseteq Q^*$ in the natural manner.

▶ **Lemma 2.4.** *For every important $S^* \subset Q^*$, the minimum cut separating $S^*$ from $\overline{S^*}$ is unique and corresponds to cycle $\mathcal{C}_{\chi(S)}$ in $G$.*

**Proof.** Let $\mathcal{C}$ be the set of edges of $G$ corresponding to some minimum cut between $S^*$ and $\overline{S}^*$ in $G^*$. Let $S \subseteq Q$ be the set of faces of $G$ corresponding to the set $S^*$. We start by observing that the edges of $G^*$ corresponding to $\mathcal{C}_{\chi(S)}$ form a cut between $S^*$ and $\overline{S}^*$. Consequently, the total weight of edges of $\mathcal{C}$ is at most the total weight of the edges of $\mathcal{C}_{\chi(S)}$.

By Claim 1, $\mathcal{C}$ contains at least $k$ edges of cost $C$, at least one edge of cost $C$ per edge layer (it needs to hit an edge in every path $\pi_0, \ldots \pi_{k-1}$). Note that $\mathcal{C}_{\chi(S)}$ contains exactly $k$ edges of cost $C$. We assign the weights in a way that $C$ is larger than all other edges in the graph taken together. This implies that $\mathcal{C}$ contains exactly one edge of cost $C$ in every edge layer $\mathcal{E}_i$. In particular, $\mathcal{C}$ contains the edge $e = \{v, w\}$.

Furthermore, the fact that ${f_i}^*$ lies on $\pi_i$ implies that the edge of weight $C$ in $\mathcal{E}_i \cap \mathcal{C}$ lies on $\pi_i^n$ if ${f_i}^* \notin S$ and lies on $\pi_i^s$ otherwise. Consequently, in $G^* - \mathcal{C}$ there is one connected component containing all vertices of $S^*$ and one connected component containing all vertices of $\overline{S^*}$. By the minimality of $\mathcal{C}$, we infer that $G^* - \mathcal{C}$ contains no other connected components apart from the aforementioned two components. By planarity, since any minimum cut in a planar graph corresponds to a collection of cycles in its dual, this implies that $\mathcal{C}$ is a single cycle in $G$.

Let $e_i$ be the unique edge of $\mathcal{E}_i \cap \mathcal{C}$ of weight $C$ and let $e'_i$ be the unique edge of $\mathcal{E}_i \cap \mathcal{C}_{\chi(S)}$ of weight $C$. We inductively prove that $e_i = e'_i$ and that the subpath of $\mathcal{C}$ between $e_i$ and $e_{i+1}$ is the same as on $\mathcal{C}_{\chi(S)}$. For the base of the induction, note that $e_0 = e'_0 = e$.

Consider an index $i > 0$ and the face $f_i$. If $f_i \in S$, i.e., $f_i$ belongs to the north side, then $e_i$ lies south of $f_i$, that is, lies on $\pi_i^s$. Otherwise, if $f_i \notin S$, then $e_i$ lies north of $f_i$, that is, lies on $\pi_i^n$.

Let $v_a$ and $v_{ba}$ be the vertices of $\mathcal{C}_{\chi(S)}$ that lie on $L_{i-1}$ and $L_i$, respectively. By the inductive assumption, $v_a$ is an endpoint of $e'_{i-1} = e_{i-1}$ that lies on $\mathcal{C}$. Let $e_i = xv_{bc}$, where $v_{bc} \in L_i$ and let $e'_i = x'v_{ba}$. Since $\mathcal{C}$ is a cycle in $G$ that contains exactly one edge on each path $\pi_i$, we infer that $\mathcal{C}$ contains a path between $v_a$ and $v_{bc}$ that consists of $e_i$ and a number of edges of $\mathcal{E}_i$ of weight $c_i$. A direct check shows that the subpath from $v_a$ to $v_{ba}$ on $\mathcal{C}_{\chi(S)}$ is the unique such path with minimum number of edges of weight $c_i$. Since the weight $c_i$ is larger than the total weight of all edges of smaller weight, from the minimality of $\mathcal{C}$ we infer that $v_{ba} = v_{bc}$ and $\mathcal{C}$ and $\mathcal{C}_{\chi(S)}$ coincide on the path from $v_a$ to $b_{ba}$.

Consequently, $\mathcal{C}$ and $\mathcal{C}_{\chi(S)}$ coincide on the path from the edge $e = vw$ to the vertex $v_{\overleftarrow{\chi(S)}} \in L_{k-2}$. From the minimality of $\mathcal{C}$ we infer that also the edge $\{w, v_{\overleftarrow{\chi(S)}}\}$ lies on the cycle $\mathcal{C}$ and, hence, $\mathcal{C} = \mathcal{C}_{\chi(S)}$. This completes the proof. ◀

▶ **Claim 2.** $\operatorname{rank}(A_{G,c}) \geq 2^{k-2}$.

**Proof.** Recall Definition 2.1 and the fact that $\mathcal{C}_{\chi(S)}$ is defined for every important $S \subseteq Q$. This means that the only edge in $\mathcal{E}_{k-1}$ that belongs to $\mathcal{C}_{\chi(S)}$ is the edge adjacent to $v_{\overleftarrow{\chi(S)}}$. Let us consider the part of adjacency matrix where rows correspond to the cuts corresponding to $\mathcal{C}_{\chi(S)}$ for important $S \subset Q$ and where columns correspond to the edges in $\mathcal{E}_{k-1}$ of weight $C$. Let us order the cuts according to $\overleftarrow{\chi(S)}$ and the edges by the index of the adjacent vertex in $L_{k-2}$ (lexicographically). Then this part of $A_{G,c}$ is an identity matrix. Hence, $\operatorname{rank}(A_{G,c}) \geq 2^{k-2}$. ◀

Lemma 2.4 and Claim 2 provide the conditions necessary for Lemma 2.3 to apply. This proves our main result stated in Theorem 1.1.

## 3 Doubly exponential example

In this section we show an example graph for which the compression technique introduced by Hagerup et al [6] does indeed produce a mimicking network on $2^{2^{\Omega(k)}}$ vertices. Our example relies on doubly exponential edge costs. Note that an example with single exponential costs can be compressed into a mimicking network of size single exponential in $k$ using the techniques of [8].

Before we go on, let us recall the technique of Hagerup et al [6]. Let $G$ be a weighted graph and $Q$ be the set of terminals. Observe that a minimum cut separating $S \subset Q$ from $\overline{S} = Q \setminus S$, when removed from $G$, divides the vertices of $G$ into two sides: the side of $S$ and the side of $\overline{S}$. The side is defined for each vertex, as all connected components obtained by removing the minimum cut contain a terminal. Now if two vertices $u$ and $v$ are on the same side of the minimum cut between $S$ and $\overline{S}$ for every $S \subset Q$, then they can be merged without changing the size of any minimum $S$-separating cut. As a result there is at most $2^{2^k}$ vertices in the graph. After this brief introduction we move on to describing our example.

Our construction builds up on the example provided in [10] in the proof of Theorem 1.2. Without loss of generality, assume that $k$ is divisible by 3 and that $l := \binom{k}{\frac{2}{3}k}$ is even. Their graph is a complete bipartite graph $G = (Q, U, E)$, where one side of the graph consists of the $k$ terminals $Q = \{q_1, \ldots, q_k\}$, and the other side of the graph consists of $l = \binom{k}{\frac{2}{3}k}$ non-terminals $U = \{u_{S_1}, \ldots, u_{S_l}\}$, with $S_1, \ldots, S_l$ denoting the different subsets of terminals of size $2/3k$. The costs of the edges of $G$ are as follows: every non-terminal $u_{S_i}$ is connected by edges of cost 1 to every terminal in $S_i$ , and by edges of cost $2 + \epsilon$ to every terminal in $\overline{S_i} = Q \setminus S_i$, for $\epsilon = 1/k$. We modify the cost function defined in this example by multiplying each edge cost by a constant $\alpha$ that we define later. We also need to be more careful with

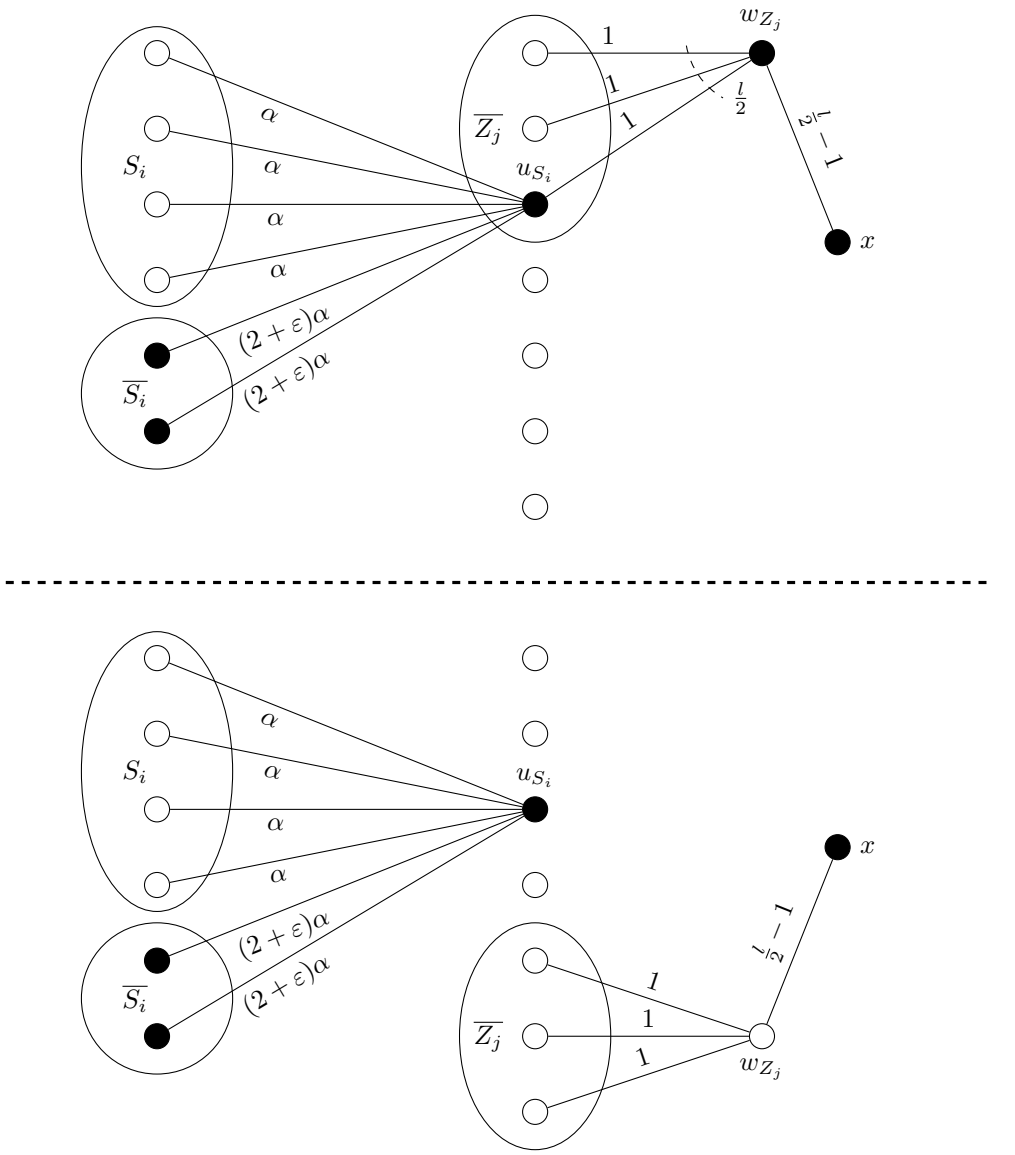



**Figure 4** Illustration of the construction. The two panels correspond to two cases in the proof, either $u_{S_i} \in Z_j$ (top panel) or $u_{S_i} \notin Z_j$ (bottom panel).

$\epsilon$. We set $\epsilon = \frac{3}{k} + \frac{6}{k^2}$. In addition to that we build a third layer of $m = \binom{l}{l/2}$ vertices $W = \{w_{Z_1}, \ldots, w_{Z_m}\}$, where $Z_1, \ldots, Z_m$ denote different subsets of $U$ of size $l/2$. There is a complete bipartite graph between $U$ and $W$. An edge $\{u_{S_i}, w_{Z_j}\}$ has cost 0 if $u_{S_i} \in Z_j$ and has cost 1 otherwise. We add one more vertex to the graph which we refer to as $x$ and connect it with edges of cost $l/2 - 1$ to each vertex in $W$. Let us refer to the resulting graph as $G'$. We let $Q' = Q \cup \{x\}$ be the corresponding terminal set; a set $S \subseteq Q'$ is *important* if $x \notin S$ and $|S| = \frac{2}{3}k$.

▶ **Lemma 3.1.** *Let $S'_i \subset Q$ be important. For $\alpha = 2^{2^k} \cdot 2^k$, the vertex $w_{Z_j}$ is on the $S_i$-side of the minimum cut between $S_i$ and $\overline{S_i}$ if and only if $u_{S_i} \in Z_i$.*

**Proof.** In [9] it is proven that the unique minimum cut separating $S_i$ from $\overline{S_i}$ in $G$, $|S_i| = 2/3k$, partitions vertices into $\overline{S_i}$ side: $Y = \{u_{S_i}\} \cup \overline{S_i}$ and $S_i$ side: $V(G) \setminus Y = \{u_{S_j} : j \neq i\} \cup S_i$. We

refer to [9] for the proof details, but the reason why this holds is the following. The terminals are connected only via vertices of $U$. Every vertex $u_{S_j}$ can either cut the edges $E(u_{S_j}, S_i)$ connecting $u_{S_j}$ with $S_i$ (choosing $\overline{S_i}$ side) or cut the edges $E(u_{S_j}, \overline{S_i})$ connecting $u_{S_j}$ with $\overline{S_i}$ (choosing $S_i$ side). For $j = i$ it holds that $c(E(U_{S_j}, S_i)) = 2/3k$ while $c(E(u_{S_j}, \overline{S_i})) = (2+\epsilon)k/3$, so it is better for $u_{S_i}$ to join $\overline{S_i}$ side. Moreover, the difference between these two values is greater than 1 for $\epsilon = \frac{3}{k} + \frac{6}{k^2}$. For $j \neq i$ it holds that $c(E(u_{S_j}, S_i)) \geq 2/3k + (1+\epsilon)$ while $c(E(u_{S_j}, \overline{S_i})) \leq k/3(2+\epsilon) - (1+\epsilon)$. It is easy to verify that for $\epsilon = \frac{3}{k} + \frac{6}{k^2}$ it is better for $u_{S_j}$ to join $S_i$ side and that the difference between the two alternative cut values is greater than 1. The bottom line is that $u_{S_i}$ picks $\overline{S_i}$ side, whereas all other $u_{S_j}$ vertices pick $S_i$ side. If a vertex switches sides, the value of the minimum cut increases by more than 1.

In our example we multiply all the edge weights in this example by $\alpha$, so the increase in the cut value is more than $\alpha$. Let us now consider graph $G'$ with terminal set $Q'$. Consider the cut between $S_i$ and $\overline{S_i} = Q' \setminus S_i$ (so $\overline{S_i}$ contains $x$). Graph $G'$ contains $G$ as a subgraph, so to disconnect $S_i$ from $Q \setminus S_i$, each vertex $u_{S_j}$ again has to cut either $E(u_{S_j}, S_i)$ or $E(u_{S_j}, Q \setminus S_i)$. Set $\alpha = 2^{2^k} \cdot 2^k$. Consider the minimum cut in $G$. The minimum cut in $G'$ restricted to $G$ uses the same edges. It does not pay of to flip sides for any vertex in $U$, as we can never make up for the difference $\alpha$ with no more than $|U| \cdot |W|$ edges of cost 1. Now fix a vertex $w_{Z_j} \in W$. We consider two cases: $u_{S_i} \in Z_j$ and $u_{S_i} \notin Z_j$; see also Figure 4.

**Case 1:** $u_{S_i} \in Z_j$**.**

As argued above, all vertices of $U$ choose their side according to what is best in $G$, so $u_{S_i}$ is the only vertex in $U$ on the $\overline{S_i}$ side. To join the $S_i$ side, $w_{Z_j}$ has to cut edges $\{x, w_{Z_j}\}$ and $\{u_{S_i}, w_{Z_j}\}$ of total cost $l/2 - 1$. To join the $\overline{S_i}$ side, $w_{Z_j}$ needs to cut $l/2$ edges of cost 1 to vertices $u_{S_{i'}}$ for $u_{S_{i'}} \notin Z_j, i' \neq i$. Thus, it is less costly if $w_{Z_j}$ joints the $S_i$ side.

**Case 2:** $u_{S_i} \notin Z_j$**.**

Again all vertices of $U$ choose their side according to what is best in $G$, so $u_{S_i}$ is the only vertex in $U$ on the $\overline{S_i}$ side. To join the $S_i$ side, $w_{Z_j}$ has to cut edges $\{x, w_{Z_j}\}$ and $\{u_{S_i}, w_{Z_j}\}$ of total cost $l/2$. To join the $\overline{S_i}$ side, $w_{Z_j}$ needs to cut $l/2 - 1$ edges of cost 1 to vertices $u_{S_{i'}}$ for $u_{S_{i'}} \notin Z_j, i' \neq i$. Thus, it is less costly for $w_{Z_j}$ to join the $\overline{S_i}$ side. ◀

Lemma 3.1 shows that $G'$ cannot be compressed using the technique presented in [6]. To see that let us fix two vertices $w_{Z_j}$ and $w_{Z_{j'}}$ and let $S_i \in Z_j \setminus Z_{j'}$. Then, Lemma 3.1 shows that $w_{Z_j}$ and $w_{Z_{j'}}$ lie on different sides of the minimum cut between $S_i$ and $\overline{S_i}$. Thus, $w_{Z_j}$ and $w_{Z_{j'}}$ cannot be merged. Similar but simpler arguments show that no other pair of vertices in $G'$ can be merged, finishing the proof of Theorem 1.2.

## References


1 Alexandr Andoni, Anupam Gupta, and Robert Krauthgamer. Towards $(1+\epsilon)$-approximate flow sparsifiers. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 279–293. SIAM, 2014. doi:10.1137/1.9781611973402.20.

2 Erin W. Chambers and David Eppstein. Flows in one-crossing-minor-free graphs. *J. Graph Algorithms Appl.*, 17(3):201–220, 2013. doi:10.7155/jgaa.00291.

3 Julia Chuzhoy. On vertex sparsifiers with steiner nodes. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference,*

*STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 673–688. ACM, 2012. doi:10.1145/2213977.2214039.

4 Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Räcke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. *SIAM J. Comput.*, 43(4):1239–1262, 2014. doi:10.1137/130908440.

5 Gramoz Goranci, Monika Henzinger, and Pan Peng. Improved guarantees for vertex sparsification in planar graphs. *CoRR*, abs/1702.01136, 2017. arXiv:1702.01136.

6 Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *J. Comput. Syst. Sci.*, 57(3):366–375, 1998. doi:10.1006/jcss.1998.1592.

7 Arindam Khan and Prasad Raghavendra. On mimicking networks representing minimum terminal cuts. *Inf. Process. Lett.*, 114(7):365–371, 2014. doi:10.1016/j.ipl.2014.02.011.

8 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 450–459. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.46.

9 Robert Krauthgamer and Inbal Rika. Mimicking networks and succinct representations of terminal cuts. *CoRR*, abs/1207.6246, 2012. arXiv:1207.6246.

10 Robert Krauthgamer and Inbal Rika. Mimicking networks and succinct representations of terminal cuts. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1789–1799. SIAM, 2013. doi:10.1137/1.9781611973105.128.

11 Robert Krauthgamer and Inbal Rika. Refined vertex sparsifiers of planar graphs. *CoRR*, abs/1702.05951, 2017. arXiv:1702.05951.

12 Konstantin Makarychev and Yury Makarychev. Metric extension operators, vertex sparsifiers and lipschitz extendability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 255–264. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.31.

# An Improved Fixed-Parameter Algorithm for One-Page Crossing Minimization

**Yasuaki Kobayashi[1], Hiromu Ohtsuka[2], and Hisao Tamaki[3]**

1 **Kyoto University, Kyoto, Japan**
`kobayashi@iip.ist.i.kyoto-u.ac.jp`
2 **Meiji University, Kanagawa, Japan**
`ohtsuka_yumecs.meiji.ac.jp`
3 **Meiji University, Kanagawa, Japan**
`tamaki@cs.meiji.ac.jp`

## Abstract

*Book embedding* is one of the most well-known graph drawing models and is extensively studied in the literature. The special case where the number of pages is one is of particular interest: an embedding in this case has a natural circular representation useful for visualization and graphs that can be embedded in one page without crossings form an important graph class, namely that of outerplanar graphs.

In this paper, we consider the problem of minimizing the number of crossings in a one-page book embedding, which we call *one-page crossing minimization*. Here, we are given a graph $G$ with $n$ vertices together with a non-negative integer $k$ and are asked whether $G$ can be embedded into a single page with at most $k$ crossings. Bannister and Eppstein (GD 2014) showed that this problem is fixed-parameter tractable. Their algorithm is derived through the application of Courcelle's theorem (on graph properties definable in the monadic second-order logic of graphs) and runs in $f(L)n$ time, where $L = 2^{O(k^2)}$ is the length of the formula defining the property that the one-page crossing number is at most $k$ and $f$ is a computable function without any known upper bound expressible as an elementary function. We give an explicit dynamic programming algorithm with a drastically improved running time of $2^{O(k \log k)}n$.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

**Keywords and phrases** Book Embedding, Fixed-Parameter Tractability, Graph Drawing, Treewidth

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.25

## 1 Introduction

In *book embeddings*, a graph is drawn in such a way that the vertices are aligned on a straight line, called the *spine*, as distinct points and each edge is drawn as a semicircle in a half plane defined by the spine. We call this half plane a *page*. In general, multiple pages are required to draw a graph without introducing any edge crossings, where a *crossing* is defined by a pair of edges that has a non-empty intersection distinct from their end vertices. The minimum number of pages we need to draw a graph without edge crossings, called *page number* or *book thickness*, is extensively studied in the literature (e.g. [3, 23]). The problem of computing page number is known to be NP-hard. More precisely, deciding if a given graph can be drawn in two pages without any crossing is NP-complete [7].

An optimization problem with an objective function different from the crossing number has also been studied. In this problem, given a graph $G$ and a small integer $p$, the objective

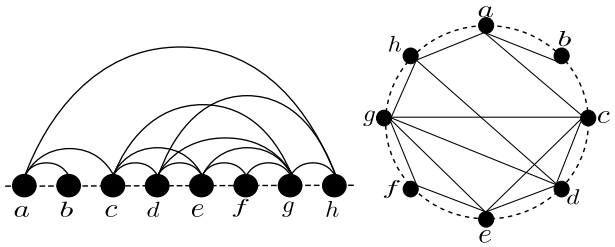




**Figure 1** A one-page drawing and a circular drawing of a graph.

is to minimize the total number of crossings in a drawing of $G$ with at most $p$ pages. This optimization problem is known as *p-page crossing minimization* and is introduced by Shahrokhi et al. [20]. Since the problem of deciding whether the page number of a graph is at most two is NP-complete [7], two-page crossing minimization is NP-hard. Indeed, the simplest case of one-page crossing minimization is already interesting. One-page crossing number is studied under various names such as *circular crossing number*, *convex crossing number*, and *outerplanar crossing number*. See [15] for example. In circular drawings, each vertex is placed on the circumference of a circular disk and each edge is drawn inside of the disk as a straight line segment. A one-page drawing may be turned into a circular drawing by topologically mapping the spine into a circle, identifying the positive and negative infinities, and mapping the half-plane into the disc enclosed by the circle. (See Figure 1, for an example). This mapping clearly preserves crossings.

One-page drawing (and hence circular drawing) with fewer crossings is important in several fields. This drawing style is frequently studied in the graph drawing community. Some well-known graph drawing software such as Graphviz[1] and yFiles[2] can produce good circular drawings. Blin et al. [4] suggested to use one-page crossing number for computing a similarity of mRNA sequences that have some secondary structures. In their paper, they asked for a fast algorithm to compute a one-page drawing with the fewest crossings.

Unfortunately, the problem of computing one-page crossing number is NP-hard as shown by Masuda et al. [18]. There are some results for special graph classes [12, 13] and some heuristics [2, 17, 21]. Bannister and Eppstein [1] tackled this problem from the perspective of parameterized complexity. They showed that the treewidth of graphs with at most $k$ crossings is $O(\sqrt{k})$ and that the graph property of having at most $k$ crossings can be defined by a formula of length $L = 2^{O(k^2)}$ in monadic second-order logic of graphs. These results are sufficient for the application of Courcelle's theorem [8, 9] to obtain an $f(L)n$ time algorithm for deciding if the one-page crossing number of a given graph is at most $k$, where $n$ is the number of vertices and $f$ is a computable function without any known upper bound expressible as an elementary function [11].

This situation is in contrast to that in the related research area of layered graph drawings, where vertices are placed on $h$ parallel lines and each edge is drawn as a straight line segment between two adjacent parallel lines. Dujmović et al. [10] gave an explicit $2^{O(h+k)^3}n$ time dynamic programming algorithm based on path-decompositions to decide if a given graph has $h$-layer drawing with at most $k$ crossings.

In this paper, we give an explicit tree decomposition based dynamic programming algorithm for one-page crossing minimization.

[1] `http://www.graphviz.org/`
[2] `https://www.yworks.com/`

▶ **Theorem 1.** *There is an algorithm which, given a graph $G$ and a non-negative integer $k$, decides whether $G$ has a one-page drawing with at most $k$ crossings in $2^{O(k \log k)}n$ time, where $n$ is the number of vertices of $G$. Moreover, if the answer is affirmative, the algorithm produces an optimal one-page drawing within the same running time.*

We would like to mention that this and the algorithm of Bannister and Eppstein [1] run in linear time, which generalize linear time algorithms recognizing and drawing outerplanar graphs [19, 22].

We borrow two tools from Bannister and Eppstein [1]. One is the upper bound on the treewidth of a graph of one-page crossing number $k$ mentioned above. The other is the concept of "crossing diagrams" which are used to classify YES-instances of the decision problem. For each such diagram, they construct a formula of length $k^{O(1)}$ for recognizing YES-instances conforming to the diagram and then take a disjunction of the formulas for all the diagrams. There are $2^{O(k^2)}$ crossing diagrams and therefore the total formula length is $2^{O(k^2)}$. We use a similar structure, which we call a "sketch", in our dynamic programming algorithm and obtain an upper bound of $2^{O(k \log k)}$ on the number of sketches through a similar but indeed somewhat finer analysis (see the proof of Lemma 14). We, however, remark that there is a fundamental difference between crossing diagrams and sketches. A crossing diagram represents a "type" of YES-instances and, for each fixed type, we need to examine each given instance for acceptance by a formula or an algorithm. On the other hand, a sketch is a succinct summary of a drawing of a subgraph. We define the "validity" of a sketch in such a way that a valid sketch of the entire graph is an immediate certificate for the positive answer to the instance and the validity of sketches can be efficiently determined by dynamic programming on a tree decomposition. We also remark that such a succinct representation is made possible by our observation on one-page drawings of biconnected graphs, which we call the chain lemma (see Section 2).

## 2 Preliminaries

Let $G$ be a graph. The set of vertices of $G$ is denoted by $V(G)$ and the set of edges of $G$ by $E(G)$. For each $v \in V(G)$, $N_G(v)$ denotes the set of neighbors of $v$ in $G$: $N_G(v) = \{u \in V(G) \mid \{u, v\} \in E(G)\}$. For $X \subseteq V(G)$, $G[X]$ denotes the subgraph of $G$ induced by $X$ and $N_G(X) = (\bigcup_{v \in X} N_G(v))$ denotes the set of neighbors of $X$.

As we have mentioned, a one-page drawing and a circular drawing are equivalent for our purposes. Therefore, we will work on circular drawings, and whenever we refer to drawings, we always refer to circular drawings. For a drawing $D$ of $G$, we write $\mathrm{cr}(D)$ to denote the number of crossings in $D$. The *one-page crossing number* $\mathrm{cr}_1(G)$ of $G$ is the minimum integer $k$ such that $G$ has a circular drawing of $k$ crossings.

Let $D$ be a drawing of $G$. We write $V(D)$ and $E(D)$ to denote $V(G)$ and $E(G)$, respectively. We denote by $\mathrm{cycle}(D)$ the cycle on $V(D)$ induced by the circle on which the vertices are drawn: two vertices are adjacent to each other in $\mathrm{cycle}(D)$ if they are consecutively placed on this circle. As special cases, if $V(D)$ is empty then $\mathrm{cycle}(D)$ is an empty graph; if $V(D)$ is a singleton, then $\mathrm{cycle}(D)$ is a self-loop; if $|V(D)| = 2$, then $\mathrm{cycle}(D)$ is a multigraph consisting of two parallel edges between the two vertices. Note that $D$ is essentially determined by $V(D)$, $E(D)$, and $\mathrm{cycle}(D)$. For $X \subseteq V(D)$, we denote by $D|X$ the subdrawing of $D$ induced by $X$, that is, the drawing obtained from $D$ by deleting all vertices in $V(D) \setminus X$. An *extension* of $D$ is a drawing obtained by adding some vertices and edges to $D$.

▶ **Definition 2.** A *tree decomposition* of $G$ is a tree $T$ where each $t \in V(T)$ is associated with $X_t \subseteq V(G)$, called a *bag*, such that

- $\bigcup_{t \in V(T)} X_t = V(G)$,
- for each $\{u, v\} \in E(G)$, there is $t \in V(T)$ with $\{u, v\} \subseteq X_t$, and
- for each $u \in V(G)$, the subgraph of $T$ induced by $\{t \in V(T) : u \in X_t\}$ is connected.

The *width* of a tree decomposition $(T, \{X_t : t \in V(T)\})$ is the maximum size of a bag minus one. The *treewidth* of $G$ is the minimum integer $w$ such that $G$ has a tree decomposition of width $w$.

To distinguish between vertices of $G$ and those of $T$, we call the vertices of $T$ *nodes*. We assume, in the rest of the paper, that tree decompositions are rooted. For a node $t \in V(T)$, we define $V_t = \bigcup_{t' \in V(T_t)} X_{t'}$, where $T_t$ is the subtree of $T$ rooted at $t$.

▶ **Definition 3.** A tree decomposition $T$ is *nice* if for each node $t$ of $T$, exactly one of the following conditions is satisfied.

- $t$ is a leaf of $T$ with $X_t = \emptyset$,
- $t$ has exactly one child $t'$ with $X_t = X_{t'} \setminus \{v\}$ for some $v \in X_{t'}$,
- $t$ has exactly one child $t'$ with $X_t = X_{t'} \cup \{v\}$ for some $v \notin X_{t'}$, or
- $t$ has exactly two children $t_1$ and $t_2$ with $X_t = X_{t_1} = X_{t_2}$.

We respectively call nodes that satisfy the above conditions, *leaf nodes*, *forget nodes*, *introduce nodes*, and *join nodes*.

▶ **Lemma 4** ([16]). *Suppose we are given a graph $G$ and its tree decomposition of width $w$. Then, there is a nice tree decomposition of $G$ of width at most $w$ such that it has $O(wn)$ nodes, where $n$ is the number of vertices of $G$. Moreover, such a nice tree decomposition can be computed in $O(w^2 n)$ time.*

The following lemma is a well-known characterization of crossing-free drawings.

▶ **Lemma 5** (Theorem 2.5 in [3]). *For every graph $G$, $\mathrm{cr}_1(G) = 0$ if and only if $G$ is outerplanar.*

Since every outerplanar graph has treewidth at most 2, we immediately have an upper bound on the treewidth with respect to its one-page crossing number: $\mathrm{tw}(G) = O(\mathrm{cr}_1(G))$. Bannister and Eppstein [1] gave a tighter bound.

▶ **Lemma 6** (Lemma 5 in [1]). *For every graph $G$, $\mathrm{tw}(G) = O(\sqrt{\mathrm{cr}_1(G)})$.*

The following simple lemma is used in some previous results (see [1], for example).

▶ **Lemma 7.** *Let $G_1, G_2, \ldots, G_t$ be the biconnected components of $G$. Then, $\mathrm{cr}_1(G) = \sum_{1 \le i \le t} \mathrm{cr}_1(G_i)$.*

Owing to this lemma, we will henceforth assume that the given graph is biconnected. We prove below a lemma crucial in exploiting the biconnectivity in our algorithm. This lemma generalizes the Hamiltonicity of biconnected outerplanar graphs with at least three vertices [6].

Let $D$ be a drawing of $G$. A path in $G$ is a *chain* in $D$ if it is also a path in $\mathrm{cycle}(D)$. We say that a vertex is *chained* in $D$ if it is an internal vertex of a chain.

▶ **Lemma 8** (Chain lemma). *Let $G$ be a biconnected graph with at least three vertices and let $D$ be a drawing of $G$. Then every vertex not incident to any crossing edge in $D$ is chained in $D$.*

**Proof.** Let $u$ be a vertex of $G$ not incident to any crossing edge. Let $v_1$ and $v_2$ be the two neighbors of $u$ in cycle$(D)$. As $G$ is biconnected, $u$ has at least two neighbors in $G$. If $u$ has no neighbor distinct from both $v_1$ and $v_2$, we are done. Suppose otherwise: $u$ has a neighbor $x$ with $x \notin \{v_1, v_2\}$. Let $P_1$ and $P_2$ be internally disjoint paths between $v_1$ and $v_2$, which exist since $G$ is biconnected. Since $\{u, x\}$ is not a crossing edge, one of the path, say $P_1$, contains $u$ and the other one, say $P_2$, contains $x$. Suppose $P_1$ does not go through edge $\{v_1, u\}$. Let $y$ be the vertex adjacent to $u$ on the subpath of $P_1$ between $v_1$ and $u$. Then, the edge $\{y, u\}$ must cross an edge in $P_2$, contradicting the assumption that $u$ is not incident to a crossing edge. Hence $P_1$ must go through edge $\{v_1, u\}$. A symmetric argument shows that $P_1$ also goes through $\{v_2, u\}$. Therefore, $u$ is chained in $D$. ◀

## 3 Colored drawings and sketches

In our dynamic programming algorithm, for each node $t$ of a given tree decomposition, we enumerate structures we call "sketches" which succinctly describe drawings of $G[V_t]$. To establish recurrences among sketches, it turns out necessary for the drawings described by sketches to carry some information on our plan on how to extend those drawings to the final drawing of $G$.

We use colors on vertices, black, white, and gray, to represent this information. A *colored drawing* of graph $H$ is a triple $(D, B, W)$ where $B$ and $W$ are disjoint subsets of $V(H)$ and $D$ is a drawing of $H$, such that every vertex incident to a crossing edge in $D$ belongs to $B$. We call the vertices in $B$ *black*, those in $W$ *white*, and all others *gray*. If $C$ is the colored drawing $(D, B, W)$, then we write $V(C)$, $E(C)$, and cycle$(C)$ to denote $V(D)$, $E(D)$, and cycle$(D)$, respectively. We use $B(C)$ and $W(C)$ to denote the set of black vertices and the set of white vertices of $C$, respectively. For $U \subseteq V(C)$, we write $C|U$ to denote the colored drawing $(D|U, B \cap U, W \cap U)$.

In the above definition, both ends of a crossing edge must be black in a colored drawing, but not vice versa: a black vertex is not necessarily incident to any crossing edge. A vertex being black rather indicates our plan that it can be incident to any crossing edges in the extension of the drawing of a subgraph of $G$ into the drawing of the entire $G$. We need some more definitions to explain the intention of the other two colors.

Let $C$ be a colored drawing. We say that $C$ *respects* $X \subseteq V(C)$ if $V(C) = B(C) \cup W(C) \cup X$ and $W(C) \cap X = \emptyset$. Note that if $C$ respects $X$ then every vertex in $X$ is either black or gray, while every vertex not in $X$ is either black or white. We will often consider a colored drawing of an induced subgraph $G[U]$ of the given graph $G$, where $U$ is accompanied with a *boundary* $X$ of $U$, a subset of $U$ such that $N_G(U \setminus X) \subseteq X$. In those situations, we will require each colored drawing of $U$ to respect $X$. This ensures that there is no edge between white vertices in $U$ and the vertices in $V(G) \setminus U$. Thus, white vertices will never be adjacent to vertices introduced in the extensions.

Let $C$ and $C'$ be colored drawings of graphs $H$ and $H'$ respectively, such that $|V(H)| = |V(H')|$. A bijection $\phi : V(H) \to V(H')$ is an *isomorphism* from $C$ to $C'$ if it is an isomorphism from $H$ to $H'$ ($u, v \in V(H)$ are adjacent to each other in $H$ if and only if $\phi(u)$ and $\phi(v)$ are adjacent to each other in $H'$), is an isomorphism of cycle$(C)$ to cycle$(C')$, and preserves the coloring ($\phi(B) = B'$, and $\phi(W) = W'$). Suppose $V(H)$ and $V(H')$ intersect each other and let $X \subseteq V(H) \cap V(H')$. We call an isomorphism $\phi$ from $C$ to $C'$ an *$X$-isomorphism* if $\phi$ fixes each vertex of $x \in X$, that is, $\phi(x) = x$. If there is an isomorphism ($X$-isomorphism) from $C$ to $C'$, then we say $C$ and $C'$ are *isomorphic* (*$X$-isomorphic*) to each other and that each of them is an *isomorph* (*$X$-isomorph*) of each other.

Let $C$ be a colored drawing. We say that $C$ is *well-formed* if every white vertex of $C$ is chained in $C$. It will turn out in Section 4 that to inductively construct drawings of $G$ (which we are assuming is biconnected), it suffices to consider only well-formed drawings.

We now define "sketches" which describe colored drawings. Let $C$ be a well-formed colored drawing. A chain in $C$ is *white* if every vertex in the chain is white. The *contraction* of $C$ is the colored drawing obtained from $C$ by contracting each maximal white chain into a single white vertex (which inherits all neighbors from the vertices of the chain). Let us note that the contraction of $C$ is unique up to isomorphism. The contraction of $C$ is well-formed since we are assuming that $C$ is well-formed. Moreover, the contraction operation preserves crossings: it does not introduce any new crossings or, since the vertices contracted are white (and hence are not incident any crossing edges), does not remove any crossings. We say that $C$ is *contracted* if it does not contain any white chain with at least two vertices and hence is the contraction of itself.

Let $X$ be a vertex set. A *sketch on* $X$ is a well-formed and contracted colored drawing that respects $X$. Let $C$ be a well-formed colored drawing of graph $H$ that respects some vertex set $X \subseteq V(H)$. Let $S$ be a sketch on $X$. We say that $S$ *describes* $C$ if $S$ is $X$-isomorphic to the contraction of $C$.

## 4 Recurrences

In this section, we fix a nice tree decomposition $T$ of $G$, and use the notation $X_t$ and $V_t$, where $t$ is a node of $T$, introduced in Section 2.

We say that a sketch $S$ on $X_t$ is *valid for* $V_t$ if there is a well-formed colored drawing $C$ of $G[V_t]$ that respects $X_t$ and is described by $S$. For brevity, we say a sketch on $t$ to mean a sketch on $X_t$ and also say that a sketch is valid on $t$, or simply valid if $t$ is clear from the context, to mean that it is valid for $V_t$.

Our dynamic programming algorithm enumerates, for each $t$, all valid sketches on $t$ that are small in the sense defined later. In the following lemmas, we establish recurrences that can be used to inductively decide if a given sketch on $t$ is valid, based on the validity of sketches on $t'$ for child nodes $t'$ of $t$.

### Leaf node

▶ **Observation 9.** *Let $t$ be a leaf node of $T$. Then, the empty sketch $(D_\emptyset, \emptyset, \emptyset)$ on $X_t = \emptyset$, where $D_\emptyset$ is an empty drawing, is valid.*

### Forget node

Let $t$ be a forget node in $T$ and $t'$ the unique child of $t$. Let $v \in V_{t'}$ be the vertex forgotten: $X_t = X_{t'} \setminus \{v\}$. Let $S$ be a sketch on $X_t$ and $S'$ a sketch on $X_{t'}$. We say that $S$ is the *parent* of $S'$ if either

**F1** $v \in B(S')$ and $S = S'$ or

**F2** $v \notin B(S')$, $v$ is chained in $S'$, and $S$ is the contraction of the colored drawing obtained from $S'$ by changing the color of $v$ from gray to white (which is well-formed since $v$ is chained).

We say that $S'$ is a *child* of $S$ if $S$ is the parent of $S'$. Note that the parent of a sketch on $X_{t'}$ is unique if one exists, while a sketch on $X_t$ may have any number of children: zero, one, or more.

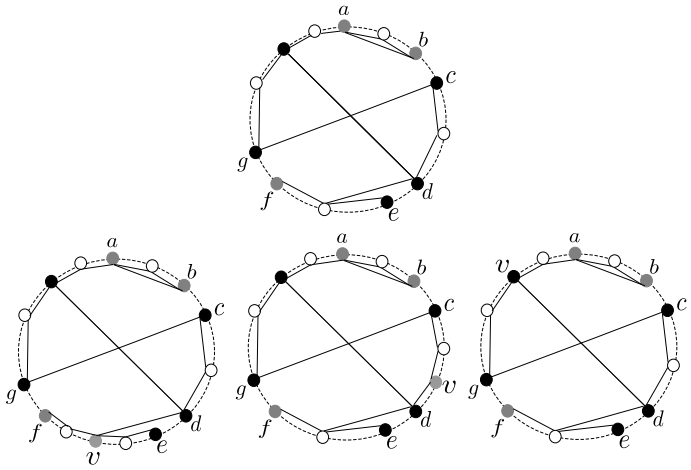



**Figure 2** The figure shows an example of a sketch on a forget node $t$. The top figure depicts a sketch on $X_t$ and the bottom figures depict some of its children. Here, $X_t = \{a, b, c, d, e, f, g\}$ and $v \in X_{t'} \setminus X_t$ is the vertex forgotten.

▶ **Lemma 10.** *Let $t$ be a forget node and $t'$ its unique child node. Then, a sketch on $t$ is valid if and only if it has a child that is valid on $t'$.*

**Proof.** Let $v$ the vertex forgotten at node $t$: $X_t = X_{t'} \setminus \{v\}$.

Let $S$ be a valid sketch on $t$. Let $C$ denote the well-formed colored drawing of $G[V_t]$ that respects $X_t$ and is described by $S$. Since $v \in V_t \setminus X_t$ and $C$ respects $X_t$, we have either $v \in B(C)$ or $v \in W(C)$.

Suppose first that $v \in B(C)$. Then, since sketch $S$ describes $C$, we have $v \in B(S)$. Then, $S$ is a sketch on $t'$ and, by Case F1, it is a child of $S$ itself. Moreover, as $W(C) \cap X_{t'} = W(C) \cap (X_t \cup \{v\}) = W(C) \cap X_t = \emptyset$, $C$ respects $X_{t'}$ as well. Since $S$ is a sketch on $t'$ and describes a colored drawing $C$ for $G[V_{t'}] = G[V_t]$ that respects $X_{t'}$, it is valid on $t'$. Therefore, we are done in this case.

Suppose next that $v \in W(C)$. Let $C'$ be the colored drawing of $G[V_{t'}] = G[V_t]$ that is identical to $C$ except that the color of $v$ is gray instead of white. Since every white vertex of $C'$ is chained as it is chained in $C$, $C'$ is well-formed. Since $W(C') \cap X_{t'} = (W(C) \setminus \{v\}) \cap (X_t \cup \{v\}) = W(C) \cap X_t = \emptyset$, $C'$ respects $X_{t'}$. Let $S'$ be the sketch on $X_{t'}$ that describes $C'$, which is unique up to $X_{t'}$-isomorphisms. Then, as $v \notin B(C')$ and the $X_{t'}$-isomorphism from $S'$ to the contraction of $C'$ fixes $v \in X_{t'}$, we have $v \notin B(S')$. Since $v$ is white in $C$ and $C$ is well-formed, $v$ is chained in $C$ and hence in $C'$. As $v$ is gray in $C'$ the contraction of $C'$ keeps $v$ chained in $S'$. Therefore, Case F2 applies and we obtain the parent $S''$ of $S'$ by turning the gray vertex $v$ white and then contracting the result. We may view the relationship between $C$ and $S''$ as follows: we first turn the color of $v$ in $C$ from white to gray and contract the result $C'$ to obtain $S'$ through an $X_{t'}$-isomorphism; then, we turn $v$ white and further contract the result into $S''$. The contraction of $C$ in one step gives us an $X_t$-isomorph of $S''$ which, since $S$ describes $C$ by assumption, is an $X_t$-isomorph of $S$ as well. As $S''$ is the parent of $S'$ and is $X_t$-isomorphic to $S$, $S$ has a child that is $X_{t'}$-isomorphic to $S'$. Since $S'$ describes $C'$, this child of $S$ is valid and we are done in this case as well.

For the converse, suppose that $S$ has a valid child $S'$. Let $C'$ be a well-formed colored drawing of $G[V_{t'}]$ that respects $X_{t'}$ and is described by $S'$. First suppose $v \in B(C')$. Since $v \in X_{t'}$, the $X_{t'}$-isomorphism from $S'$ to the contraction of $C'$ fixes $v$. Therefore, we have $v \in B(S')$. Case F1 applies and we have $S = S'$. Since $S$ describes $C'$ and $C'$ respects $X_t \subseteq X_{t'}$, $S$ is valid on $t$ and we are done in this case.

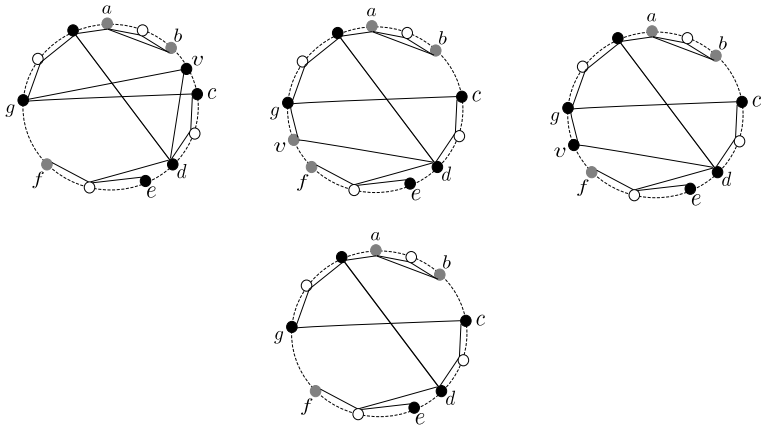



**Figure 3** The figures show an example of sketches on an introduce node $t$. The bottom figure depicts a sketch on $X_{t'}$ and the top figures depict some of its parents. Here, $X_t = \{a, b, c, d, e, f, g, v\}$ and $v \in X_t \setminus X_{t'}$ is the vertex introduced.

So suppose that $v \notin B(C')$. Then we have $v \notin B(S')$ and Case F2 must apply since $S'$ is a child of $S$. Let $C$ be the colored drawing of $G[V_t] = G[V_{t'}]$ which is identical to $C'$ except that the color of $v$ is turned white from gray. Since $v$ is chained in $S'$ by the condition of Case F2 and the $X_{t'}$-isomorphism from $S'$ to the contraction of $C'$ fixes $v$, $v$ is chained in $C'$ and hence in $C$. We may obtain an $X_t$-isomorph of $S$ from $C$ by turning the color of $v$ gray, contracting the result $C'$ into an $X_{t'}$-isomorph of $S'$, turning the color of $v$ back to white, and then contracting the result. Since $v$ is chained in $C$, we may perform the contraction in one step, which shows that $S$ describes $C$ and hence is valid on $t$. ◀

## Introduce node

Let $t$ be an introduce node in $T$ and $t'$ the unique child of $t$. Let $v \in X_t$ be the vertex introduced: $X_t = X_{t'} \cup \{v\}$. Let $S$ be a sketch on $X_t$ and $S'$ a sketch on $X_{t'}$. We say that $S'$ is the *child* of $S$ if either

1. $|X_t| = 1$ and $S'$ is an unique empty sketch, or
2. $|X_t| > 1$, $S' = S|V_{t'}$, and neither of the two vertices in $N_{\mathrm{cycle}(S)}(v)$, which are not necessarily distinct, are white in $S$.

Note that $V_{t'} = V_t \setminus \{v\}$ and hence $S'$ is obtained from $S$ by removing $v$. We say that $S$ is a parent of $S'$ if $S'$ is the child of $S$. Note that the child of a sketch on $X_t$ is unique up to isomorphism if one exists, while a sketch on $X_{t'}$ may have more than one parent in general.

▶ **Lemma 11.** *Let $t$ be an introduce node and $t'$ a unique child of $t$. A sketch $S$ on $t$ is valid if and only if it has a child that is valid on $t'$.*

**Proof.** Let $v$ be the vertex introduced in $t$: $X_t = X_{t'} \cup \{v\}$. We only prove the case 2, since the case 1 is straightforward.

Suppose that $S$ is a valid sketch on $t$ such that $|X_t| > 1$. Let $C$ be a well-formed colored drawing of $G[V_t]$ that respects $X_t$ and is described by $S$. Since $v \in X_t$ and $C$ respects $X_t$, $v$ is either black or gray in $C$. Let $C' = C|V_{t'}$. Since $V_t = V_{t'} \cup \{v\}$, $C'$ is obtained from $C$ by removing $v$. Let $u$ be a vertex adjacent to $v$ in cycle$(C)$. If $u \in W(C)$ then $u$ must be chained in $C$ since $C$ is well-formed. But this is a contradiction, since $v \notin V_{t'}$ has no neighbor in $W(C) = W(C') \subseteq V_{t'} \setminus X_{t'}$. Therefore, we conclude that neither of the two vertices in $N_{\mathrm{cycle}(C)}(v)$ are white in $C$. Since the contraction of $C$ does not change this local

structure around $v$, the vertices in $N_{\mathrm{cycle}(C)}(v)$ correspond to those in $N_{\mathrm{cycle}(S)}(v)$ through the $X_t$-isomorphism from $S$ to the contraction of $C$. Therefore, $S$ satisfies the condition for having a child. Let $S'$ be the child of $S$. Since $C$ and $C'$ have the same set of maximal white chains, the $X_t$-isomorphism from $S$ to the contraction of $C$ gives an $X_{t'}$-isomorphism from $S'$ to the contraction of $C'$, when restricted to $V(S') = V(S) \setminus \{v\}$. Therefore, $S'$ is valid.

For the converse, suppose $S$ has a child $S'$ that is valid on $t'$. Let $C'$ be a well-formed colored drawing of $G[V_{t'}]$ that respects $X_{t'}$ and is described by $S'$. Let $v_1$ and $v_2$ be the two vertices in $N_{\mathrm{cycle}(S)}(v)$. Each of $v_1$ and $v_2$ is either black or gray in $S$, since $S$ has a child. Since $S$ respects $X_t$ and $v \in X_t$, $v$ is also either black or gray. Since $v_1$ and $v_2$ are adjacent to each other in cycle$(S')$, the $X_{t'}$-isomorphism from $S'$ to the contraction of $C'$ maps $v_1$ and $v_2$ to vertices $v_1'$ and $v_2'$ of $C'$ that are either black or gray in $C'$ and are adjacent to each other in cycle$(C')$. Let $D$ be the drawing of $G[V_t]$ obtained from the drawing of $C'$ by adding $v$ between $v_1'$ and $v_2'$. Let $C = (D, B, W(C'))$, where $B = B(C') \cup \{v\}$ if $v \in B(S)$ and $B = B(C')$ otherwise.

We claim that $C$ is a colored drawing. To see this, let $e$ and $f$ be edges of $G[V_t]$ that cross each other in $D$. If neither $e$ nor $f$ is incident to $v$, then the crossing is in $C'$ and hence all the ends of $e$ and $f$ are black in $C'$ and hence in $C$. So suppose one of $e$ and $f$, say $e$, is incident with $v$. Let $u_1$ be the other end of $e$ and let $u_2$ and $u_3$ be the ends of $f$. For $i = 1, 2, 3$, $u_i$ has a vertex $u_i'$ in $S'$ corresponding to $u_i$: the $X_{t'}$-isomorphism from $S'$ to the contraction of $C'$ maps $u_i'$ to either $u_i$ or the contraction of a white maximal chain containing $u_i$. Since $e$ and $f$ cross each other in $C$, the edge between $v$ and $u_1'$ and the edge between $u_2'$ and $u_3'$ must cross each other in $S$. As sketch $S$ is a colored drawing, $v$, $u_1'$, $u_2'$, and $u_3'$ are black in $S$. By the definition of $C$, $v$ is in $C$. Moreover, from the definition of the contraction, we see that $u_1$, $u_2$, and $u_3$ must be black in $C'$ and hence in $C$. Therefore, $C$ satisfies the condition for being a colored drawing. $C$ is well-formed, since every white vertex in $C$ is a white vertex in $C'$ and therefore is chained in $C'$ and hence in $C$.

Since $C'$ is obtained from $C$ by removing $v$ and neither of the two vertices in $N_{\mathrm{cycle}(C)}(v)$ are white, $C$ is contracted in the same way as $C'$ is contracted into an $X_{t'}$-isomorph of $S'$, resulting in an $X_t$-isomorph of $S$. Therefore $S$ describes $C$ and hence is valid. ◀

## Join Node

Let $t$ be a join node with child nodes $t_1$ and $t_2$. From the definition of join node, we have $X_t = X_{t_1} = X_{t_2}$ and $(V_{t_1} \setminus X_t) \cap (V_{t_2} \setminus X_t) = \emptyset$. We may assume that neither $V_{t_1} \setminus X_t$ nor $V_{t_2} \setminus X_t$ is empty and hence $|V_t| \geq 3$.

Let $S$ be a sketch on $t$. Let $S_1$ and $S_2$ be sketches on $t_1$ and $t_2$, respectively. We say that $S$ is the parent of the pair $(S_1, S_2)$ if

1. $V(S_1) \setminus X_t$ and $V(S_2) \setminus X_t$ partition $V(S) \setminus X_t$,
2. there is no edge in $E(S)$ between $V(S_1) \setminus X_t$ and $V(S_2) \setminus X_t$,
3. $S_1 = S|V(S_1)$, and
4. $S_2 = S|V(S_2)$.

▶ **Lemma 12.** *Let $t$ be a join node with child nodes $t_1$ and $t_2$. A sketch $S$ on $t$ is valid if and only if there are valid sketches $S_1$ on $t_1$ and $S_2$ on $t_2$ such that $S$ is the parent of the pair $(S_1, S_2)$.*

**Proof.** Suppose first that $S$ is a valid sketch on $t$. Let $C$ be a well-formed colored drawing of $G[V_t]$ that respects $X_t$ and is described by $S$. Let $C_i = C|V_{t_i}$ for $i = 1, 2$. Then, for $i = 1, 2$, $C_i$ is well-formed, since each white vertex $v \in C_i$ is chained in $C$ and, since $v \in V_{t_i} \setminus X_t$, $N_G(v)$

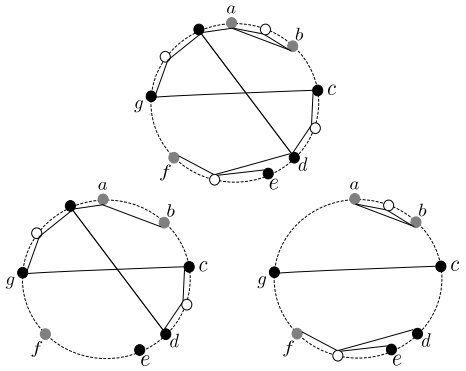



**Figure 4** The figures show an example of a sketch on a join node $t$. The top figure depicts a sketch on $X_t$ and the bottom figures depict a pair of sketches whose parent is the sketch in the top figure. Here, $X_t = \{a, b, c, d, e, f, g\}$.

are in $V_{t_i}$, is chained in $C_i$ as well. Moreover, for $i = 1, 2$, $C_i$ respects $X_{t_i}$. To confirm this, fix $i$. Since $W(C) \cap X_t = \emptyset$ as $C$ respects $X_t$, we have $W(C_i) \cap X_{t_i} = (W(C) \cap V_{t_i}) \cap X_t = \emptyset$. We also have $V(C_i) = B(C_i) \cup W(C_i) \cup X_{t_i}$, since $V(C_i) = V(C) \cap V_{t_i}$, $B(C_i) = B(C) \cap V_{t_i}$, and $W(C_i) = W(C) \cap V_{t_i}$. Therefore, $C_i$ respects $X_{t_i}$.

Observe that each chain in $C$ is either entirely contained in $C_1$ or entirely contained in $C_2$, as there is no edge of $G$ between $V_{t_1} \setminus X_t$ and $V_{t_2} \setminus X_t$. Therefore, the partition $(V_{t_1} \setminus X_t, V_{t_2} \setminus X_t)$ of $V_t \setminus X_t$ induces a partition $(R_1, R_2)$ of $V(S) \setminus X_t$ through the $X_t$-isomorphism from $S$ to the contraction of $C$. There is no edge in $E(S)$ between $R_1$ and $R_2$, since such an edge would correspond to an edge between $V_{t_1} \setminus X_t$ and $V_{t_2} \setminus X_t$, contradicting the definition of a join node. Therefore, $S$ is the parent of the pair $(S_1, S_2)$, where $S_i$ for $i = 1, 2$ is defined by $S_i = S|(R_i \cup X_t)$. The contraction of $C$ to an $X_t$-isomorph of $S$ induces the contraction of $C_i$ to an $X_t$-isomorph of $S_i$, for $i = 1, 2$. Therefore, $S_i$ describes $C_i$ and hence is valid, for $i = 1, 2$.

For the converse, suppose $S$ is the parent of a pair $(S_1, S_2)$ where $S_i$ is a valid sketch on $t_i$, for $i = 1, 2$. For $i = 1, 2$, let $C_i$ be a well-formed colored drawing of $G[V_{t_i}]$ that respects $X_{t_i}$ and is described by $S_i$. We combine $C_1$ and $C_2$ into a colored drawing $C$ of $G[V_t]$ as follows. We take the sketch $S$ and replace each vertex $v \in V(S)$ as follows. If $v \in X_t$ then we keep $v$ as it is. If $v \in B(S_1) \setminus X_t$, then we replace $v$ by the vertex of $C_1$ to which $v$ is mapped by the $X_t$-isomorphism from $S_1$ to the contraction of $C_1$; the case $v \in B(S_2) \setminus X_t$ is similar. If $v \in W(S_1)$, then we replace $v$ by the maximal white chain of $C_1$, the contraction of which $v$ is mapped to by the $X_t$-isomorphism from $S_1$ to the contraction of $C_1$; the case $v \in W(S_2)$ is similar. We let the resulting colored drawing be $C$. All edges of the drawing except for those in the maximal white chains correspond to edges in $S$ and edges in white chains are not crossing. Therefore, every vertex incident to a crossing edge is colored black in $C$ as it is in $S$ and hence $C$ is indeed a colored drawing. That $C$ respects $X_t$ is trivial. To see that $C$ is well-formed, let $w$ be a white vertex of $C$. Then, either $w$ corresponds to a white vertex of $S$ or $w$ is in a white chain that corresponds to a white vertex of $S$. Since this white vertex in $S$ is chained since $S$ is well-formed, $w$ is chained in $C$. Finally, combining the contractions of $C_i$ into $X_t$-isomorphs of $S_i$ for $i = 1, 2$, we get the contraction of $C$ into an $X_t$-isomorph of $S$. Therefore, $S$ describes $C$ and hence is valid. ◀

## 5 Algorithm

By Lemma 7, we can solve our problem independently for each biconnected component of $G$. Moreover, the biconnected components can be computed in linear time [14]. When the treewidth of $G$ is larger than $c\sqrt{k}$ for some constant $c > 0$, by Lemma 6, we can conclude $\mathrm{cr}_1(G) > k$. Thus, in the following, we can assume that the given graph $G$ is biconnected and its treewidth is at most $c\sqrt{k}$. Applying the algorithm of Bodlaender et al. [5], we can obtain a tree decomposition of $G$ whose width is $O(\sqrt{k})$ in $2^{O(\sqrt{k})}n$ time and its nice tree decomposition by Lemma 4.

We say that a sketch is *small* if it has at most $4k$ black vertices and contains at most $k$ crossings. Our dynamic programming algorithms inductively enumerates the set of all valid sketches on $t$ that are small, for each node $t$ of the nice tree decomposition, using the recurrences established in the previous section. The dynamic programming table for $t$ contains one representative sketch from each $X_t$-isomorphism class. It is straightforward to verify that, to decide if a small sketch on $t$ is valid or not, only small sketches on child node(s) of $t$ need to be examined. Thus, the computation gives us the set of all small and valid sketches on the root of the tree decomposition. Therefore, the proof of the correctness of our algorithm lies in showing the following lemma.

▶ **Lemma 13.** *$G$ has a drawing with at most $k$ crossings if and only if there is a small sketch on $\emptyset$ that is valid for $V(G)$.*

**Proof.** Suppose first that there is a small sketch $S$ on $\emptyset$ that is valid for $V(G)$. Then, since $S$ is valid, there is a colored drawing $C$ of $G$ described by $S$. The number of crossings of $C$ is equal to the number of crossings in $S$ and is at most $k$.

For the converse, suppose that $G$ has a drawing with at most $k$ crossings. Turn this drawing into a colored drawing $C$ of $G$ by making the ends of all crossing edges black and all other vertices white. We have at most $4k$ black vertices and $C$ respects $\emptyset$. Moreover, $C$ is well-formed by the chain lemma (Lemma 8). Contracting $C$ (with respect to boundary $\emptyset$), we obtain a small sketch on $\emptyset$ that describes $C$ and hence is valid for $V(G)$. ◀

For the running time, we prove the following:

▶ **Lemma 14.** *For each $t \in V(T)$, the number of small sketches on $t$, counting one from each $X_t$-isomorphism class, is $2^{O(k \log k)}$.*

**Proof.** The number of non-isomorphic graphs with $p$ vertices and $q$ edges is upper bounded by $p^{2q}$. For such a graph, the number of different colorings is at most $3^p$, and the number of different drawings is at most $(p-1)!$. Observe that every small sketch of $p$ vertices has $2p + k - 3$ edges. This follows from the fact that we can obtain an outerplanar drawing by removing at most $k$ edges from an arbitrary small sketch. Since the color of the two vertices in $N_{\mathrm{cycle}(S)}(v)$ for every white vertex $v$ in an arbitrary sketch $S$ is either black or gray, the number of white vertices is at most $4k + O(\sqrt{k})$. Therefore, each small sketch has at most $8k + O(\sqrt{k})$ vertices and at most $17k + O(\sqrt{k})$ edges, and hence the number of such drawings is $2^{O(k \log k)}$. ◀

## References


1 M. J. Bannister and D. Eppstein. Crossing minimization for 1-page and 2-page drawings of graphs with bounded treewidth. In *Proc. of GD 2014*, pages 210–221. Springer, 2014.

2 M. Baur and U. Brandes. Crossing reduction in circular layouts. *WG*, 3353:332–343, 2004.

3 F. Bernhart and P. C. Kainen. The book thickness of a graph. *Journal of Combinatorial Theory, Series B*, 27(3):320–331, 1979.

4 G. Blin, G. Fertin, D. Hermelin, and S. Vialette. Fixed-parameter algorithms for protein similarity search under mRNA structure constraints. *Journal of Discrete Algorithms*, 6(4):618–626, 2008.

5 H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk. A $c^k n$ 5-Approximation Algorithm for Treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016.

6 G. Chartrand and F. Harary. Planar Permutation Graphs. *Annales de l'I.H.P. Probabilités et statistiques*, 3(4):433–438, 1967.

7 F. R. K. Chung, F. Thomson Leighton, and A. L. Rosenberg. Embedding graphs in books: a layout problem with applications to VLSI design. *SIAM Journal on Algebraic Discrete Methods*, 8(1):33–58, 1987.

8 B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

9 B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109(1):49–82, 1993.

10 V. Dujmović, M. R. Fellows, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. Rosamond, S. Whitesides, and D. R. Wood. On the parameterized complexity of layered graph drawing. *Algorithmica*, 52(2):267–292, 2008.

11 M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, 130(1):3–31, 2004.

12 R. Fulek, H. He, O. Sỳkora, and I. Vrt'o. Outerplanar crossing numbers of 3-row meshes, Halin graphs and complete $p$-partite graphs. *SOFSEM 2005: Theory and Practice of Computer Science*, pages 376–379, 2005.

13 H. He, A. Sălăgean, and E. Mäkinen. One-and two-page crossing numbers for some types of graphs. *International Journal of Computer Mathematics*, 87(8):1667–1679, 2010.

14 J. Hopcroft and R. Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

15 P. C. Kainen. The book thickness of a graph II. *Congressus Numerantium*, 71:121–132, 1990.

16 T. Kloks. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994.

17 E. Mäkinen. On circular layouts. *International Journal of Computer Mathematics*, 24(1):29–37, 1988.

18 S. Masuda, T. Kashiwabara, K. Nakajima, and T. Fujisawa. On the NP-completeness of a computer network layout problem. In *Proceedings of the 1987 IEEE International Symp. on Circuits and Systems*, pages 292–295, 1987.

19 S. L. Mitchell. Linear algorithms to recognize outerplanar and maximal outerplanar graphs. *Information Processing Letters*, 9(5):229–232, 1979.

20 F. Shahrokhi, O. Sỳkora, L. A. Székely, and I. Vrt'o. Book embeddings and crossing numbers. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 256–268. Springer, 1994.

21 J. M. Six and I. G. Tollis. Circular drawings of biconnected graphs. In *ALENEX*, volume 99, pages 57–73. Springer, 1999.

22 M. M. Sysło. Characterizations of outerplanar graphs. *Discrete Mathematics*, 26(1):47–53, 1979.

23 M. Yannakakis. Embedding planar graphs in four pages. *Journal of Computer and System Sciences*, 38(1):36–67, 1989.

# Treewidth with a Quantifier Alternation Revisited

**Michael Lampis[1] and Valia Mitsou*[2]**

1 **Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE, Paris, France**
michail.lampis@dauphine.fr

2 **Université de Lyon, LIRIS, CNRS, UMR 5205, Université Lyon 1, Villeurbanne, Lyon, France**
vmitsou@liris.cnrs.fr

## Abstract

In this paper we take a closer look at the parameterized complexity of ∃∀SAT, the prototypical complete problem of the class $\Sigma_2^p$, the second level of the polynomial hierarchy. We provide a number of tight fine-grained bounds on the complexity of this problem and its variants with respect to the most important structural graph parameters. Specifically, we show the following lower bounds (assuming the ETH):

- It is impossible to decide ∃∀SAT in time less than *double-exponential* in the input formula's treewidth. More strongly, we establish the same bound with respect to the formula's primal vertex cover, a much more restrictive measure. This lower bound, which matches the performance of known algorithms, shows that the degeneration of the performance of treewidth-based algorithms to a tower of exponentials already begins in problems with one quantifier alternation.
- For the more general ∃∀CSP problem over a non-boolean domain of size $B$, there is no algorithm running in time $2^{B^{o(vc)}}$, where $vc$ is the input's primal vertex cover.
- ∃∀SAT is already NP-hard even when the input formula has constant modular treewidth (or clique-width), indicating that dense graph parameters are less useful for problems in $\Sigma_2^p$.
- For the two weighted versions of ∃∀SAT recently introduced by de Haan and Szeider, called $\exists_k\forall$SAT and $\exists\forall_k$SAT, we give tight upper and lower bounds parameterized by treewidth (or primal vertex cover) and the weight $k$. Interestingly, the complexity of these two problems turns out to be quite different: one is double-exponential in treewidth, while the other is double-exponential in $k$.

We complement the above negative results by showing a double-exponential FPT algorithm for QBF parameterized by vertex cover, showing that for this parameter the complexity never goes beyond double-exponential, for any number of quantifier alternations.

**1998 ACM Subject Classification** F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics, G.2.2 Graph Theory

**Keywords and phrases** Treewidth, Exponential Time Hypothesis, Quantified SAT

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.26

## 1 Introduction

The main goal of this paper is to provide a fine-grained complexity analysis of ∃∀SAT, the prototypical complete problem for the second level of the polynomial hierarchy, with respect to the most important structural graph parameters, and especially treewidth.

* Valia Mitsou was supported by the ANR-14-CE25-0006 project of the French National Research Agency.





Treewidth, a graph parameter that roughly measures how tree-like a graph is, is one of the central notions of parameterized complexity theory. Its popularity and success rest largely on the fact that when a graph's treewidth is small a very large variety of problems which are normally intractable can be solved efficiently (often in linear time [3]). This success has strongly motivated research that seeks to apply the ideas of treewidth to other domains, such as satisfiability problems. This can be done by defining a graph that (partially) encodes the structure of the input instance, and then using this graph's structure to design efficient algorithms. Much work has been devoted in this direction, and by now the complexity of satisfiability and related constraint satisfaction problems (counting, maximization, etc.) is well-understood not only with respect to treewidth, but also other related structural graph parameters such as clique-width, modular treewidth, vertex cover, and others [8, 19, 21, 23, 25, 24].

Despite this success, one notable weakness of the treewidth approach is that its effectiveness rapidly deteriorates once one starts considering problems outside NP. Indeed, it has long been known that as one considers problems that need more and more quantifier alternations to be expressed, the "hidden constants" in treewidth-based algorithms become towers of exponentials [10, 16, 20] and for an unbounded number of alternations, even constant treewidth does not help [1]. One response to this weakness has been the search for other graph measures which are more robust in the face of alternating quantifiers [9, 11, 12, 15]. Our goal in this paper on the other hand is to more closely examine whether we can hope to evade this deteriorating performance not necessarily by changing the graph parameter, but by restricting ourselves to problems with only one quantifier alternation, that is, problems in $\Sigma_2^p$, the second level of the polynomial hierarchy.

Given the current state of the art, one would probably expect the typical $\Sigma_2^p$-complete problem to have double-exponential complexity when parameterized by treewidth, as a result of the extra quantifier alternation, and indeed the best currently known algorithms typically do have this complexity. Nevertheless, almost no *concrete lower bounds* are known showing that natural problems in this class need a double-exponential dependence on treewidth. The lower bounds given in [10, 16, 20] are either asymptotic or only give concrete tight bounds for the *odd* levels of the polynomial hierarchy. Perhaps the only tight double-exponential lower bound for the complexity of a $\Sigma_2^p$-complete problem parameterized by treewidth is given in [17], which proves such a result for Choosability. To the best of our knowledge no such bounds are known for other concrete problems, and in particular, no such tight bounds are known for perhaps the most basic problem in this class: $\exists\forall$SAT.

**Our contribution:** Our main conceptual contribution is a simple direct reduction showing that the complexity of $\exists\forall$SAT when parameterized by the treewidth of the input formula has to be double-exponential, unless the ETH is false (Theorem 1). This essentially matches the running time of the best currently known algorithm [2]. Before this work similar bounds were only known for satisfiability problems complete for odd levels of the polynomial hierarchy [20], and thus this fills a natural hole in the literature. However, beyond advancing our understanding of the complexity of quantified satisfiability, we believe the main value of this result is in providing the first "textbook" example of a double-exponential lower bound for treewidth. Unlike currently known lower bounds of somewhat similar flavor ([15, 17, 20]), our result is a completely self-contained reduction that essentially does not need any gadgets. Furthermore, because of the central role of $\exists\forall$SAT in the class $\Sigma_2^p$ we expect that our lower bound may serve as a starting point to prove similar bounds for various other problems in $\Sigma_2^p$.

Building and extending on the above result we give a number of tight upper and lower bounds on ∃∀SAT and related problems. Specifically, we observe that we are able to give a similar double-exponential lower bound on the treewidth dependence for ∃∀3-SAT, and that our reduction shows that ∃∀SAT is NP-hard even for formulas of constant modular treewidth or clique-width (meaning that dense graph parameters are likely to be much less useful for ∃∀SAT than they are for SAT). We also observe that the main lower bound for ∃∀SAT applies not only when the problem is parameterized by treewidth, but much more strongly, when it is parameterized by vertex cover.

Finding this latter fact surprising, since most problems are significantly easier when parameterized by vertex cover than by treewidth, we investigate vertex cover as a parameter more closely. We show that, despite matching closely the complexity of treewidth for ∃∀SAT, this parameter is indeed much more algorithmically amenable in other ways: first, it allows a double-exponential algorithm for QBF with any number of quantifier alternations (a problem PSPACE-complete for constant treewidth); and second, it allows a single-exponential algorithm for ∃∀3-SAT. We note that QBF was already known to be FPT parameterized by vertex cover [9] as a corollary of an algorithm for a much more general parameter (prefix pathwidth), but since we concentrate on vertex cover we are able to give an algorithm that is both faster and significantly simpler.

Having analyzed the complexity of ∃∀SAT with respect to some of the most important graph parameters, we move towards two more recently introduced variations with special interest for parameterized complexity: $\exists_k\forall$SAT and $\exists\forall_k$SAT. In these problems, introduced in the works of de Haan and Szeider [6, 5, 7], one of the two quantifiers is weighted, that is, we only consider assignments that set $k$ out of its variables to true. Though both problems are FPT parameterized by treewidth we show that their complexity is quite different: one is double-exponential with respect to treewidth, while the other is double-exponential with respect to $k$ (and single-exponential in the treewidth). Both lower bounds match algorithms that follow from simple adaptations of [2].

Finally, we consider a more general version of our problem: ∃∀CSP where variables are no longer necessarily boolean. The question here is how the complexity of the problem changes not only with respect to treewidth, but also with the size of the domain of a non-boolean CSP. Once again we show a double-exponential bound that essentially matches the performance of the best known algorithm.

## 2 Definitions and Preliminaries

### 2.1 Problem Definitions

We recall some standard definitions related to satisfiability problems: a *literal* is a boolean variable, or its negation; a clause is a disjunction of literals; a term is a conjunction of literals; a formula is in conjunctive normal form (CNF) if it is a conjunction of clauses; it is in disjunctive normal form (DNF) if it is a disjunction of terms. We will sometimes overload notation and view clauses and terms as sets of literals.

The main problem we study in this paper is ∃∀SAT, the prototypical complete problem for the second level of the polynomial hierarchy [26]. In this problem we are given a quantified formula of the form $\exists\mathbf{x}\forall\mathbf{y}\phi(\mathbf{x},\mathbf{y})$, where $\mathbf{x},\mathbf{y}$ are disjoint tuples of boolean variables and $\phi$ is a DNF formula. The question is to decide whether there exists an assignment to the variables of $\mathbf{x}$ so that for all assignments to $\mathbf{y}$, $\phi$ evaluates to True. We refer to $\mathbf{x}$ as the existential variables and $\mathbf{y}$ as the universal variables. When all terms of $\phi$ have size at most $d$ we refer to this problem as ∃∀$d$-SAT. We also consider two *weighted* versions: in $\exists_k\forall$SAT we are asked

if there exists an assignment that sets exactly $k$ existential variables to True, such that for all assignments to $\mathbf{y}$, $\phi$ evaluates to True; in $\exists\forall_k$SAT we are asked if there exists any assignment to $\mathbf{x}$ such that for all assignments that set exactly $k$ of the variables of $\mathbf{y}$ to True, $\phi$ evaluates to True. A more general version of $\exists\forall$SAT allows the input formula to have any number of quantifiers, that is, we are given a formula of the form $\exists x_1 \forall x_2 \exists x_3 \dots Q x_n \phi(x_1, x_2, \dots, x_n)$, where $\phi$ is in CNF, and are asked if it evaluates to True. We call this problem QBF.

We will also consider the more general $\exists\forall$CSP problem. Here we are given a CSP instance again involving two tuples of variables $\mathbf{x}, \mathbf{y}$, which can now take values from some finite domain $\Sigma$. Furthermore, we are given a set of constraints: a constraint of arity $r$ involves $r$ variables from $\mathbf{x} \cup \mathbf{y}$, and we are given a list of assignments that satisfy the constraint, that is, a subset of $\Sigma^r$ which determines which combinations of assignments to the involved variable satisfy the constraint. We say that an $\exists\forall$CSP instance is a Yes instance if there exists an assignment to $\mathbf{x}$ such that for all assignments to the variables of $\mathbf{y}$ at least one constraint is satisfied.

## 2.2 Graph Parameters

Given a propositional formula $\phi$ we consider two types of graphs associated with it. The primal graph, denoted $G_p(\phi)$ contains a vertex for each variable of $\phi$; two vertices of $G_p(\phi)$ are connected if and only if the corresponding variables of $\phi$ appear in a constraint together. The incidence graph, denoted $G_i(\phi)$ contains a vertex for each variable and for each constraint of $\phi$; two vertices of $G_i(\phi)$ are connected if the corresponding variable appears in the corresponding constraint.

In this paper we study various structural restrictions on formulas by considering graph parameters restricting the structure of $G_i(\phi)$ or $G_p(\phi)$. The parameters we consider are the treewidth of $G_i(\phi)$ and $G_p(\phi)$ denoted $tw_i(\phi)$ and $tw_p(\phi)$ respectively; the vertex cover of $G_p(\phi)$, denoted $vc(\phi)$; and the modular treewidth and clique-width of $G_i(\phi)$, denoted $mtw(\phi)$ and $cw(\phi)$ respectively. We refer the reader to standard textbooks for the definitions of treewidth, clique-width and vertex cover [4]. Modular treewidth is the treewidth of a graph after contracting all vertices that share the same neighborhood into single vertices [22]. We recall the following well-known relationships between these parameters. For all formulas $\phi$ we have $vc(\phi) \geq tw_p(\phi) \geq tw_i(\phi) \geq mtw(\phi)$. Because of this inequality, hardness results which apply for vertex cover automatically carry over to the other (more general) parameters, while algorithmic results which apply to modular treewidth also apply to the other (less general) parameters. We recall that it is also known that $cw(\phi)$ is bounded by some function of $mtw(\phi)$.

## 2.3 Binary Representations

Let $\mathbf{x}$ be a tuple of variables, and $i$ be an integer such that $i < 2^{|\mathbf{x}|}$. We define the function $B(i, \mathbf{x})$ which produces a set of *literals*, that is, a set which contains for each variable of $\mathbf{x}$ either the variable itself or its negation. Negations will be added according to the binary representation of $i$. More formally, if $\mathbf{x} = (x_1, x_2, \dots, x_n)$ we define inductively $B(i, x) = B(\lfloor i/2 \rfloor, \mathbf{x} \setminus \{x_n\}) \cup l_n$, where $l_n = x_n$ if $i$ is even, and $l_n = \neg x_n$ otherwise. For the base case ($n = 0$) we set $B(0, \emptyset) = \emptyset$.

## 3 Tight Bounds for ∃∀SAT

In this section we consider the complexity of the prototypical $\Sigma_2^p$-complete problem ∃∀SAT. Our first result is a fine-grained lower bound which states that any algorithm for ∃∀SAT must either use time exponential in the number of variables of the formula, or run in time double-exponential in the input formula's primal vertex cover (assuming the ETH). This result is obtained through a direct reduction from an $n$-variable instance of 3-SAT that produces an instance of ∃∀SAT with roughly the same number of variables, but vertex cover $O(\log n)$. In other words, our reduction trades the additional complexity of the problem (caused by the extra level of quantification), to encode the formula in such a way that its structure is significantly simplified. We note that this lower bound is tight, not only for vertex cover, but even for much more general parameters, such as treewidth, for which double-exponential algorithms for ∃∀SAT are known [2].

We then go on to give a second version of the same reduction that gives a similar running-time bound for ∃∀3-SAT parameterized by the input formula's primal treewidth. Here we use a standard trick to reduce the size of all terms to 3; the non-trivial part is to prove that this does not significantly increase the primal treewidth of the instance. Interestingly, the lower bound for ∃∀3-SAT does *not* apply for primal vertex cover. As we show in Section 4 there is a good reason for this, as ∃∀3-SAT is solvable in single-exponential time for this parameter.

We also observe that our main reduction produces instances with constant modular treewidth (and hence clique-width), indicating that these two dense parameters (for which SAT is in XP) are unlikely to be of help with satisfiability problems in $\Sigma_2^p$.

▶ **Theorem 1.** *There is no algorithm which, given an* ∃∀SAT *instance $\phi$ with $n$ variables, can decide if $\phi$ is True in time $2^{2^{o(vc(\phi))}}2^{o(n)}$, unless the ETH is false.*

**Proof.** We consider an instance of 3-SAT $\psi$ with $n$ variables and $m$ clauses. Recall that the ETH states that there is no $2^{o(n+m)}$ algorithm that decides 3-SAT[13, 14]. We will construct a quantified DNF formula $\exists \mathbf{x} \forall \mathbf{y} \phi(\mathbf{x}, \mathbf{y})$, whose primal vertex cover will be $O(\log n)$.

Let $\mathbf{x} = \{x_1, \ldots, x_n\}$ be the set of variables of $\psi$. We retain the same variables as the existential variables of $\phi$, and we introduce $\log m$ universally quantified variables $\mathbf{y} = (y_1, \ldots, y_{\log m})$. We assume here without loss of generality that $m$ is a power of 2, otherwise some clauses of $\psi$ can be repeated.

Suppose that the clauses of $\psi$ are numbered $C_0, C_1, \ldots, C_{m-1}$. Let $C_i$ be a clause of $\psi$ of length $l \leq 3$. We construct $l$ terms in $\phi$ as follows: each term contains one distinct literal of $C_i$, and all the literals of $B(i, \mathbf{y})$ (the binary encoding defined in Section 2.3). This completes the construction.

To see the correctness of the reduction, suppose that $\psi$ is satisfiable. We take a satisfying assignment and use the same values for the $\mathbf{x}$ variables in $\phi$. Now, for any assignment of the $\mathbf{y}$ variables there will be an $i$ such that all the literals in $B(i, \mathbf{y})$ are true. Consider the terms we added representing the clause $C_i$. One of them contains a true literal from the original clause, so it is a satisfied term. For the other direction, if $\exists \mathbf{x} \forall \mathbf{y} \phi(\mathbf{x}, \mathbf{y})$ is true, we use the same assignment for $\mathbf{x}$ in $\psi$. If some clause $C_i$ of $\psi$ is not satisfied, this would imply that setting $\mathbf{y}$ to the assignment that agrees with $B(i, \mathbf{y})$ would also make $\phi$ false, a contradiction.

Observe that the primal graph of the formula $\phi$ becomes an independent set if we remove the $\log m$ vertices corresponding to the variables of $\mathbf{y}$. Thus, $vc(\phi) = O(\log m) = O(\log n)$. Furthermore, the size of $\phi$ is linear in the size of $\psi$. Thus, if there exists an algorithm which can solve ∃∀SAT in time $2^{2^{o(vc(\phi))}}2^{o(|\phi|)}$ then, with the above reduction, this implies a $2^{o(n)}$ algorithm for 3-SAT. ◀

▶ **Corollary 2.** ∃∀SAT *is NP-hard when restricted to instances with incidence graphs of constant modular treewidth or clique-width.*

**Proof.** We observe that in the incidence graph of the instances of Theorem 1 the universal variables form a class of false twins. Contracting them to a single vertex results in a graph with constant treewidth. ◀

▶ **Theorem 3.** *There is no algorithm which, given an* ∃∀3-SAT *instance* $\phi$ *with* $n$ *variables can decide if* $\phi$ *is True in time* $2^{2^{o(tw_p(\phi))}}2^{o(n)}$*, unless the ETH is false.*

**Proof.** We build on the proof of Theorem 1. Recall that in that reduction we started from a 3-SAT instance $\psi$ with $n$ variables and $m$ clauses and constructed an ∃∀SAT instance $\phi$ with $n$ existential variables, $\log m$ universal variables, such that all terms contain all universal variables and exactly one existential variable. We will edit this instance to make the size of each term at most 3, without affecting the value of $\phi$ and without increasing its primal treewidth.

We perform the following modification: as long as there exists a term $t$ of $\phi$ with size greater than 3, we introduce to $\phi$ a new universally quantified variable $z$, remove $t$ from $\phi$ and replace it with two new terms. The first contains $z$ and two of the literals of $t$, while the second contains $\neg z$ and the remaining literals of $t$. We repeat this until all terms have size at most 3. It is not hard to see that this transformation does not affect the answer (we performed the standard reduction from SAT to 3-SAT). Furthermore, it is not hard to perform this transformation in such a way that every term of the resulting instance contains at most 2 of the new $z$ variables, or at most one existential variable and one $z$ variable.

Let us now examine the primal treewidth of the modified instance. As previously, we remove all the $\log m$ universal variables of $\phi$. What remains is made up of the existential variables and the universal $z$ variables added while breaking up large terms. We observe however, that all vertices corresponding to $z$ variables have degree at most 2, because each appears in only two terms, and each term may only contain either at most one other $z$ variable or one existential variable. Furthermore, among the $z$ variables introduced while breaking up a term $t$, there must now exist one with degree one, since one of them appears in the same term as an existential variable. We now use the fact that deleting a leaf does not change the value of a graph's treewidth, and applying this rule repeatedly we can eventually delete all the $z$ variables. This only leaves the existential variables in the primal graph, and these form an independent set. ◀

## 4 Algorithms

One surprising aspect of the lower bound given in Theorem 1 is that, if one takes into account known double-exponential algorithms for ∃∀SAT parameterized by treewidth [2], it indicates that ∃∀SAT has the same complexity parameterized by incidence treewidth, primal treewidth, or primal vertex cover. This is unexpected, because primal vertex cover is a significantly more restrictive measure than treewidth, and hence we would expect things to be significantly easier for this parameter. One indication in this direction is given by the fact that the lower bound for ∃∀3-SAT given in Theorem 3 does not apply to vertex cover.

In this section we give two algorithmic results that confirm that vertex cover is indeed a much more algorithmically amenable parameter. We first show in Theorem 4 that the complexity of QBF is double-exponential in the formula's vertex cover *for any* number of quantifier alternations. This is in sharp contrast with the case of treewidth, where it is known that (under the ETH) the complexity of solving QBF rapidly degenerates into a tower of

exponentials as the number of quantifier alternations increases [20]. Second, we show that there is a good reason why Theorem 3 cannot be extended to vertex cover, as the complexity of $\exists\forall d$-SAT for any fixed $d$ is "only" single-exponential in the input formula's vertex cover raised to the $d$-th power.

Both of the algorithms we give are quite simple, and rely on the standard branching procedure which can decide QBF in exponential time (hence both algorithms only need polynomial space), together with the elementary observation that when working with a CNF formula we can always discard a clause that is a proper superset of another clause (or a term that is a superset of another term for DNFs). The key idea in both cases is to measure progress as a function of the number of clauses the formula contains that use only variables from the vertex cover. Let us also recall that the fixed-parameter tractability of QBF parameterized by vertex cover was already shown in [9] as a corollary of a more general algorithm using the notion of prefix pathwidth. However, the algorithm following from these results is triple-exponential in the input formula's vertex cover [18], while here we give an algorithm which is much simpler and whose running time is optimal (under Theorem 1).

▶ **Theorem 4.** *There is an algorithm that, given a* QBF *instance $\phi$, decides $\phi$ in time $O^*(2^{2^{O(vc(\phi))}})$.*

**Proof.** We run the standard branching algorithm for quantified boolean formulas, where we branch on the variables of $\phi$ in the order that they appear quantified. We prove that the branching depth can be bounded by $3^k + k$, where $k$ is the size of the primal vertex cover. We assume that we are given an optimal vertex cover of the primal graph (otherwise, we can find one in time $2^k$).

Let $V$ be the set of variables and $S \subseteq V$ be the variables corresponding to a vertex cover of the primal graph ($|S| = k$). Let further $C'$ be a set containing all possible clauses that can be constructed using only variables from $S$. We observe that $|C'| \leq 3^k$, as each variable might appear positive, negative, or not appear in such a clause.

The essential reason that the standard branching algorithm works is that branching on a variable $x$ either decreases the size of the vertex cover (if $x \in S$) or adds to the formula clauses of $C'$ which are not already there. Indeed, since each clause of $\phi$ is represented by a clique in the primal graph, for every clause $c$ of $\phi$ there can be at most one variable that doesn't belong in $S$, so branching on a variable that doesn't belong in $S$ creates clauses in $C'$. The key observation now is that we should only branch on a variable $x \notin S$ if it is going to create clauses of $C'$ that do not already belong in $\phi$:

▶ **Observation 5.** *If $c, c'$ are clauses that both belong in $\phi$ and $c \supseteq c'$, then the formula $\phi'$ created by deleting $c$ from $\phi$ is equivalent to $\phi$.*

Let us now describe the algorithm more formally. Let our quantified formula be $\phi = Q_1 x_1 Q_2 x_2 \dots Q_n x_n \psi$, where $Q_i$ is the $i^{th}$ quantifier. We define $\phi^T = \phi[x_1 = T]$ and $\phi^F = \phi[x_1 = F]$. In order to evaluate $\phi$ we shall first evaluate at least one of $\phi^T, \phi^F$ (recursively) and take their disjunction if $Q_1 = \exists$ or conjunction if $Q_1 = \forall$. We consider the following cases.

If $x_1$ appears only positive (resp. negative) in $\psi$ then there is no need to branch on $x_1$ as setting it to the suitable truth value depending on whether $Q_1$ is existential or universal simplifies the formula: if $Q_1 = \exists$ then $\phi \equiv \phi^T$ (resp. $\phi \equiv \phi^F$), whereas if $Q_1 = \forall$ then $\phi \equiv \phi^F$ (resp. $\phi \equiv \phi^T$).

On the other hand, if $x_1$ appears both positive and negative, we argue that we only branch if $x_1 \in S$ or if doing so creates at least one clause $c'$ of $C'$ that doesn't already belong in $\psi$. It is clear that the branching depth should be bounded by $3^k + k$.

Let us describe the branching for $Q_1 = \exists$ (the case $Q_1 = \forall$ is similar). Step 1 is to remove all the clauses $c \in \psi$ which are supersets of some other clause $c' \in \psi$ (we can do this because of Observation 5). Now, if $x_1$ appears positive (resp. negative) in some clause $c$ of $\psi$, setting $x_1 = T$ (resp. $x_1 = F$) makes $c$ true. In this case $c$ doesn't appear in $\phi^T$ (resp. $\phi^F$) at all. If $x_1$ appears negative (resp. positive) in $c$ and we set $x_1 = T$ (resp. $x_1 = F$), then $c$ is replaced in $\phi^T$ (resp. $\phi^F$) by an equivalent clause $c'$, where $c = c' \vee (\neg)x$.

Observe that none of $\phi^T, \phi^F$ contains $x_1$, so if $x_1 \in S$ then the primal vertex cover of $\psi$ is decreased by one. If $x_1 \notin S$, both branches $\phi^T, \phi^F$ contain at least one positive and one negative appearance of $x_1$, thus they should each contain at least one clause $c' \in C' \setminus \psi$ (if $c = (\neg)x_1 \vee c'$ for some $c' \in C' \cap \psi$ then Step 1 removes $c$ from $\psi$).

Since the branching depth is $3^k + k$, the algorithm will run in time $O^*(2^{3^k+k})$. ◀

▶ **Theorem 6.** *For all fixed $d \geq 3$ there exists an algorithm that decides an input $\exists\forall d$-SAT formula $\phi$ in time $O^*(2^{O(vc(\phi)^d)})$.*

**Proof.** The proof uses similar arguments as the proof of Theorem 4. There are two important observations that need to be added: first, during the course of the execution of the standard branching algorithm we never increase the size of any clause. Hence, if we started with an instance of $\exists\forall d$-SAT, we always maintain an instance of $\exists\forall d$-SAT. Second, the set $C'$ that contains all terms that only use variables from the vertex cover now has size at most $O(k^d)$, where $k = vc(\phi)$. To see this, observe that there are at most $2^d\binom{k}{d}$ clauses of size exactly $d$ that use only variables from the vertex cover. Hence, whenever the algorithm is forced to branch, it either decreases the vertex cover or increases the number of terms from $C'$ in the formula, giving the promised running time. ◀

## 5 Tight Bounds for Weighted Problems

In this section we consider two variations of $\exists\forall$SAT that have recently attracted attention in the parameterized complexity community: $\exists_k\forall$SAT and $\exists\forall_k$SAT. Our main results are ETH-based lower bounds which provide evidence that the complexity of these two problems is quite different.

We first consider $\exists_k\forall$SAT. This is a problem that easily admits an algorithm running in time $n^k 2^{O(tw_i(\phi))}$: one could simply guess a weight $k$ assignment for the existential variables and then solve the remaining (single-quantifier) instance with standard algorithms. The problem also admits an algorithm running in time (roughly) $2^{2^{tw_i(\phi)}}$, simply by adapting the algorithms given in [2] to only consider existential assignments of weight $k$. Our first result is that neither of these algorithms can be significantly improved: any algorithm that runs in time $n^{o(k)}$ must have complexity double-exponential in treewidth (in fact, more strongly, double-exponential in primal vertex cover). We also extend this result to $\exists_k\forall$3-SAT, as in Section 3.

We then move on to $\exists\forall_k$SAT. For this problem it is possible to adapt the algorithm of [2] to run in time (roughly) $2^{tw_i(\phi)^k}$. Informally, the reason for this is that the double-exponential running time in the algorithm of [2] comes from the need to consider all subsets of all possible assignments to universal variables in a bag. Here we only need to worry about assignments of weight (at most) $k$, hence there are at most $\binom{tw_i(\phi)}{k}$ of them to consider. We prove that obtaining a running time better than this would contradict the ETH, again even in the case of vertex cover. Interestingly, we note that it is not immediate here to obtain similar bounds for $\exists\forall_k$3-SAT, because the standard method to break down terms by introducing universal variables does notnecessarily preserve the weight of universal assignments.

▶ **Theorem 7.** *There is no algorithm which, given an $\exists_k\forall$SAT instance $\phi$, can decide if $\phi$ is True in time $2^{2^{o(vc(\phi))}}|\phi|^{o(k)}$, unless the ETH is false.*

**Proof.** We begin with a 3-SAT instance $\psi$ with $n$ variables and $m$ clauses, and we first explain how to produce from this an equivalent $\exists_k$SAT instance $\psi'$ with $k2^{n/k}$ variables and $m+k$ clauses, for any $k$. We partition the variables $\mathbf{x}$ into $k$ sets $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k}$ of $n/k$ variables each (suppose without loss of generality that $k$ divides $n$). Now, for each $i \in \{1, \ldots, k\}$, for each assignment of truth values to the variables of $\mathbf{x_i}$ we construct a variable that will appear in $\psi'$. Call the set of such variables $\mathbf{x'}$, and we have $|\mathbf{x'}| = k2^{n/k}$. For every clause $C_i$ of $\psi$ we construct a clause $C'_i$ of $\psi'$ which contains all the variables of $\mathbf{x'}$ that agree with at least one of the literals of $C_i$. Finally, for each $i \in \{1, \ldots, k\}$ we add a clause that contains all the variables that represent an assignment of $\mathbf{x_i}$. This completes the construction, and we observe that $\psi'$ has $m+k$ clauses in total. It is not hard to see how a satisfying assignment of $\psi$ can be transformed into a satisfying assignment of $\psi'$ that sets exactly $k$ variables to true: for each $\mathbf{x_i}$ we set to true the unique variable of $\mathbf{x'}$ that represents its partial assignment, and set all other variables of $\mathbf{x'}$ to false. The $m$ clauses that were constructed from clauses of $\psi$ are satisfied, because we started with a satisfying assignment of $\psi$, while the $k$ remaining clauses are satisfied by definition. For the other direction, suppose that we have a satisfying assignment to $\psi'$ which sets exactly $k$ of the $\mathbf{x'}$ variables to True. The $k$ additional clauses ensure that any satisfying assignment to $\psi'$ that sets at most $k$ variables to true must select exactly one partial assignment for each $\mathbf{x_i}$. We can therefore obtain an assignment to the variables of $\psi$ from a weight-$k$ satisfying assignment to $\psi'$, and it is not hard to see that this assignment will satisfy $\psi$.

We now apply the construction of Theorem 1 to the weighted formula $\psi'$. Namely, we introduce a set $\mathbf{y}$ of $\log(m+k)$ universal variables (assume without loss of generality that $m+k$ is a power of two), and construct for each clause $C'_i$ of $\psi'$, for each of its literals $l_j$ a term that contains $l_j$ and the literals of $B(i, \mathbf{y})$. This completes the construction of the $\exists_k\forall$SAT instance $\phi$. As in Theorem 1, there exists a satisfying assignment of weight $k$ for $\psi'$ if and only if the same assignment to the existential variables of $\phi$ makes the formular true for all assignments to the universal variables.

We observe that, if the original 3-SAT formula had $n$ variables, then $|\phi| = 2^{O(n/k)}$ and $vc(\phi) = O(\log n)$. Thus, if there was an algorithm running in $2^{2^{o(vc(|\phi|)}}|\phi|^{o(k)}$ for $\exists_k\forall$SAT, then this would give a $2^{o(n)}$ algorithm for 3-SAT. ◀

▶ **Corollary 8.** *There is no algorithm which, given an $\exists_k\forall$3-SAT instance $\phi$, can decide if $\phi$ is True in time $2^{2^{o(tw_p(\phi))}}|\phi|^{o(k)}$, unless the ETH is false.*

**Proof.** The proof is identical to that of Theorem 3, except we start with an instance produced by the reduction of Theorem 7. In particular, we again introduce a new universal variable for each term of size larger than 3 and use it to break it down into two terms. The arguments that we used to bound the primal treewidth in Theorem 3 also apply here. ◀

▶ **Theorem 9.** *There is no algorithm which, given an $\exists\forall_k$SAT instance $\phi$ with $n$ variables, can decide if $\phi$ is True in time $2^{vc(\phi)^{o(k)}}2^{o(n)}$, unless the ETH is false.*

**Proof.** We begin with a 3-SAT formula $\psi$ with $n$ variables and $m$ clauses. Our aim is to construct an $\exists\forall_k$SAT formula $\phi$ with $vc(\phi) = m^{O(1/k)}$ and $|\phi| = O(n+m)$. Let $M = m^{1/k}$, and we assume without loss of generality that $M$ is an integer.

We construct $\phi$ as follows: let $\mathbf{x}$ be the set of variables of $\psi$; we retain these as the existential variables of $\phi$. We also introduce $k$ disjoint sets of universal variables, each of which

contains exactly $M$ distinct variables. We label these variables $y_{j_1,j_2}$, with $j_1 \in \{0,\ldots,k-1\}$ and $j_2 \in \{0,\ldots,M-1\}$.

Suppose that the clauses of $\psi$ are numbered $C_0,\ldots,C_{m-1}$. For each clause $C_i$ we do the following: for each of its literals $l$ we construct a term in $\phi$ that contains $l$. Consider now the integer $i$ written in base $M$; this representation of $i$ contains $k$ digits, each between 0 and $M-1$. We add to the term that contains $l$ all variables $y_{j_1,j_2}$ such that the $j_1$-th digit of $i$ written in base $M$ is $j_2$. We repeat this process for all literals of $C_i$, and clauses of $\psi$.

To complete the construction, we add to $\psi$ $k$ additional terms: for each $j_1 \in \{0,\ldots,k-1\}$ we add the term $(\wedge_{j_2=0}^{M-1}\neg y_{j_1,j_2})$.

Let us now argue for the correctness of the reduction. If $\psi$ has a satisfying assignment, we use the same assignment to the existential variables of $\phi$. We claim that any assignment of weight $k$ to the universal variables must make a term true. To see this, observe that because of the $k$ additional terms, we can assume that the assignment to the universal variables sets for each $j_1 \in \{0,\ldots,k-1\}$ exactly one $j_2 \in \{0,\ldots,M-1\}$ with $y_{j_1,j_2}=1$. Consider now any assignment to the universal variables with this property. We can now find an integer $i$ with the following property: when $i$ is written in base $M$, for all $j_1 \in \{0,\ldots,k-1\}$, if the $j_1$-th digit of $i$ has value $j_2$, then the assignment has set $y_{j_1,j_2}=1$. The terms we constructed for clause $C_i$ have all their universal variables set to true, and each contains one literal from $C_i$, hence at least one of these terms is true. It is not hard to see that the converse direction follows with a similar reasoning.

For the running time bound, we note that the primal graph of $\phi$ becomes an independent set if we remove the $km^{1/k}$ universal variables, hence $vc(\phi) = O(m^{1/k})$ and the lower bound follows. ◀

## 6 Non-binary ∃∀CSP

In this section we consider the more general ∃∀CSP problem. The difference between this problem and ∃∀SAT is that in ∃∀CSP the variables don't necessarily have a boolean domain, but may take values from some finite set $\Sigma$. As a result, that size of $\Sigma$ must be factored into the complexity. The algorithm of [2] runs in time (roughly) $2^{|\Sigma|^{tw_i(I)}}$, for an ∃∀CSP instance $I$, because it considers all possible subsets of all possible $|\Sigma|^{tw_i(I)}$ assignments to the variables of a bag. In other words, the second exponent is linear in $tw(I)\log|\Sigma|$. Our main result is that this dependence is optimal, even for the more restricted case of vertex cover, unless the ETH is false.

▶ **Theorem 10.** *There is no algorithm which, given an* ∃∀CSP *instance $I$ with $n$ variables and domain $\Sigma$, can decide if $I$ is a Yes instance in time $2^{|\Sigma|^{o(vc(I))}}|\Sigma|^{o(n)}$, unless the ETH is false.*

**Proof.** We give a reduction in two steps, beginning from a 3-SAT formula $\psi$. First, we construct from $\psi$ a CSP instance $I_1$ with alphabet $\Sigma$, and show that this instance cannot be solved in time $|\Sigma|^{o(n)}$, where $n$ is the number of variables of $I_1$. We then construct from $I_1$ an instance $I_2$ of ∃∀CSP for which we obtain the promised lower bound.

Suppose that $\psi$ has $n_1$ variables and $m_1$ clauses. We partition its clauses into $n = \lceil m_1/\log_7|\Sigma|\rceil$ groups, each containing at most $\log_7|\Sigma|$ clauses. For each one of these groups we construct a variable in $I_1$. Consider now a group of clauses and list all assignments of the variables that appear in this group and satisfy all clauses of the group. There are at most $7^{\log_7|\Sigma|}$ such assignments, since each clause has size at most three, and therefore at most 7 satisfying assignments. As a result, we can make an injective mapping to values in

the domain $\Sigma$ for the variable that represents this group of clauses in $I_1$, from the set of partial assignments that satisfy all the clauses. We now add some constraints to our CSP instance to ensure that the assignment to $\psi$ must be consistent. In particular, let $x_1, x_2$ be two variables of $I$ that represents groups of clauses of $\psi$ that share some variables. We add a constraint to $I$ which only allows the variables $x_1, x_2$ to receive a pair of values from $\Sigma$ such that the corresponding partial assignments to the variables of $\psi$ is consistent and satisfies all the clauses of the two groups. This completes the construction. Since the new instance has $n = O(m_1/\log|\Sigma|)$ variables, if there was an algorithm solving CSP in time $|\Sigma|^{o(n)}$ the ETH would be false.

Let us now use $I_1$ to produce an instance $I_2$ of $\exists\forall$CSP. We note that $I_1$ has $n$ variables and therefore, since all constraints have arity two, at most $m = O(n^2)$ constraints. We retain all variables of $I_1$ as existential variables, and introduce a set of $\lceil \log(m|\Sigma|^2)/\log|\Sigma| \rceil$ universal variables $\mathbf{y}$. The main observation now is that the total number of distinct assignments to $\mathbf{y}$ is at least $m|\Sigma|^2$. We can therefore define an injective mapping which, given a constraint of $I_1$ and an assignment to the variables of this constraint that *falsifies* the constraint, produces an assignment to the variables of $\mathbf{y}$. We now define the constraints of $I_2$ as follows: let $x_1, x_2$ be two variables involved in a constraint $C$ of $I_1$ and let $(v_1, v_2)$ be an assignment to $x_1, x_2$ that is not allowed by the constraint. Let $\mathbf{v}$ be the assignment to $v$ to which we have mapped the constraint $C$ and its falsifying assignment $(v_1, v_2)$. We add to $I_2$ the two following constraints: $(x_1 \neq v_1 \wedge \mathbf{y} = \mathbf{v})$ and $(x_2 \neq v_2 \wedge \mathbf{y} = \mathbf{v})$. We perform this step for every falsifying assignment of every constraint. To complete the construction, for every assignment $\mathbf{v}$ to $\mathbf{y}$ that is not mapped to, we add the constraint $(\mathbf{y} = \mathbf{v})$. We finally note that, because every constraint involves at most $\lceil \log(m|\Sigma|^2)/\log|\Sigma| \rceil + 1$ variables, all constraints can be explicitly described in polynomial time by listing all of their at most $O(m|\Sigma|^2)$ satisfying assignments.

Let us now argue for correctness. If there is a satisfying assignment to $I_1$, we select the same assignment for the existential variables of $I_2$. If the universal variables receive some assignment that is not mapped to by a constraint of $I_1$, then the new instance has a satisfied constraint, because of the constraints of the form $(\mathbf{y} = \mathbf{v})$. Otherwise, the assignment to $\mathbf{y}$ corresponds to a falsifying assignment $(v_1, v_2)$ to a constraint $C$ of $I_1$. However, since we started with a satisfying assignment to $I_1$, either $x_1 \neq v_1$ or $x_2 \neq v_2$, so again at least one constraint is satisfied. It is not hard to see that the converse direction follows with similar arguments.

For the running time lower bound, we note that the primal graph of $I_2$ has vertex cover at most $|\mathbf{y}| = O(\log m/\log|\Sigma|)$, from which the bound follows. ◀

## References


1. Albert Atserias and Sergi Oliva. Bounded-width QBF is pspace-complete. *J. Comput. Syst. Sci.*, 80(7):1415–1429, 2014.
2. Hubie Chen. Quantified constraint satisfaction and bounded treewidth. In *ECAI*, pages 161–165. IOS Press, 2004.
3. Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.
4. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
5. Ronald de Haan and Stefan Szeider. Compendium of parameterized problems at higher levels of the polynomial hierarchy. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:143, 2014.

6 Ronald de Haan and Stefan Szeider. Fixed-parameter tractable reductions to SAT. In *SAT*, volume 8561 of *Lecture Notes in Computer Science*, pages 85–102. Springer, 2014.

7 Ronald de Haan and Stefan Szeider. Parameterized complexity classes beyond para-np. *J. Comput. Syst. Sci.*, 87:16–57, 2017.

8 Holger Dell, Eun Jung Kim, Michael Lampis, Valia Mitsou, and Tobias Mömke. Complexity and approximability of parameterized max-csps. In *IPEC*, volume 43 of *LIPIcs*, pages 294–306. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

9 Eduard Eiben, Robert Ganian, and Sebastian Ordyniak. Using decomposition-parameters for QBF: mind the prefix! In *AAAI*, pages 964–970. AAAI Press, 2016.

10 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004.

11 Robert Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In *IPEC*, volume 7112 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2011.

12 Robert Ganian, Petr Hlinený, Jaroslav Nesetril, Jan Obdrzálek, Patrice Ossona de Mendez, and Reshma Ramadurai. When trees grow low: Shrubs and fast MSO1. In *MFCS*, volume 7464 of *Lecture Notes in Computer Science*, pages 419–430. Springer, 2012.

13 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

14 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

15 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012.

16 Michael Lampis. Model checking lower bounds for simple graphs. *Logical Methods in Computer Science*, 10(1), 2014.

17 Dániel Marx and Valia Mitsou. Double-exponential and triple-exponential bounds for choosability problems parameterized by treewidth. In *ICALP*, volume 55 of *LIPIcs*, pages 28:1–28:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

18 Sebastian Ordyniak. Private communication, 2017.

19 Sebastian Ordyniak, Daniël Paulusma, and Stefan Szeider. Satisfiability of acyclic and almost acyclic CNF formulas. *Theor. Comput. Sci.*, 481:85–99, 2013.

20 Guoqiang Pan and Moshe Y. Vardi. Fixed-parameter hierarchies inside PSPACE. In *LICS*, pages 27–36. IEEE Computer Society, 2006.

21 Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for CNF formulas of bounded modular treewidth. *Algorithmica*, 76(1):168–194, 2016.

22 Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for CNF formulas of bounded modular treewidth. *Algorithmica*, 76(1):168–194, 2016.

23 Sigve Hortemo Sæther, Jan Arne Telle, and Martin Vatshelle. Solving #sat and MAXSAT by dynamic programming. *J. Artif. Intell. Res.*, 54:59–82, 2015.

24 Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010.

25 Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *J. Comput. Syst. Sci.*, 76(2):103–114, 2010.

26 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.

# An Output Sensitive Algorithm for Maximal Clique Enumeration in Sparse Graphs

George Manoussakis

**LRI-CNRS, Université Paris Sud, Université Paris Saclay, France**
georgе@lri.fr

**Abstract**

The *degeneracy* of a graph $G$ is the smallest integer $k$ such that every subgraph of $G$ contains a vertex of degree at most $k$. Given an $n$-order $k$-degenerate graph $G$, we present an algorithm for enumerating all its maximal cliques. Assuming that $\alpha$ is the number of maximal cliques of $G$, our algorithm has setup time $\mathcal{O}(n(k^2 + s(k+1)))$ and enumeration time $\alpha\mathcal{O}((k+1)f(k+1))$ where $s(k+1)$ (resp. $f(k+1)$) is the preprocessing time (resp. enumeration time) for maximal clique enumeration in a general $(k+1)$-order graph. This is the first output sensitive algorithm whose enumeration time depends only on the degeneracy of the graph.

**1998 ACM Subject Classification** G.2.2 Graph Theory, Graph Algorithms

**Keywords and phrases** Enumeration algorithms, maximal cliques, $k$-degenerate graphs

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.27

## 1 Introduction

*Degeneracy*, introduced by Lick *et al.* [12], is a common measure of the sparseness of a graph and is closely related to other sparsity measures such as arboricity and thickness. Degenerate graphs often appear in practice. For instance, the World Wide Web graph, citation networks, and collaboration graphs have low arboricity, and therefore have low degeneracy [18]. Furthermore, planar graphs have degeneracy at most five [12] and the Barabàsi-Albert model of preferential attachment [9], frequently used as a model for social networks, produces graphs with bounded degeneracy.

Cliques are complete subgraphs of a graph. The problem of listing all maximal cliques in general and $k$-degenerate graphs has been extensively studied. We can essentially distinguish between two families of algorithms. On one side, *worst-case output size* algorithms have been proposed. Their complexities match the maximal number of maximal cliques one can find in the considered graphs. For instance, Tomita *et al.* [16] propose an algorithm enumerating all maximal cliques of a general $n$-order graph in time $\mathcal{O}(3^{n/3})$. This is worst-case output size optimal in general graphs as for instance the Moon-Moser graphs have $\Theta(3^{n/3})$ cliques [3, 15]. Thus, even printing the cliques of these graphs would require at least $\Omega(3^{n/3})$ time. Similarly, for $k$-degenerate graphs, Eppstein *et al.* [8] prove a $\mathcal{O}((n-k)3^{k/3})$ bound on the maximal number of maximal cliques and then show an algorithm running in time $\mathcal{O}(k(n-k)3^{k/3})$. The two algorithms described above rely on ideas of the Bron-Kerbosch algorithm [2]. These results are summarized in the first three rows of Table 1. This table is largely inspired by the one provided by Conte *et al.* [7].

Another family is the one of *polynomial delay output sensitive* algorithms. Their time complexities can be divided into a preprocessing phase followed by an enumeration phase. During the enumeration phase, maximal cliques of the graph are outputted with polynomial delay: the wait between the output of two maximal cliques is bounded by some polynomial


**Table 1** Bounds for maximal clique enumeration where $q-1 \leq k \leq \Delta \leq n-1$. $^+$ : these are polynomial time delay algorithms. Their delay is equal to their enumeration time divided by the number of maximal cliques $\alpha$. The space bounds do not include the space needed to store the graph.

| Algorithm | Setup | Enumeration | Space |
|---|---|---|---|
| Bron-Kerbosch [2] | $\mathcal{O}(m)$ | unbounded | $\mathcal{O}(n+q\Delta)$ |
| Tomita *et al.* [16] | $\mathcal{O}(m)$ | $\mathcal{O}(3^{n/3})$ | $\mathcal{O}(n+q\Delta)$ |
| Eppstein *et al.* [8] | $\mathcal{O}(m)$ | $\mathcal{O}(k(n-k)3^{k/3})$ | $\mathcal{O}(n+k\Delta)$ |
| Johnson *et al.* [11]$^+$ | $\mathcal{O}(mn)$ | $\alpha\mathcal{O}(mn)$ | $\mathcal{O}(\alpha n)$ |
| Tsukiyama *et al.* [17]$^+$ | $\mathcal{O}(n^2)$ | $\alpha\mathcal{O}((n^2-m)n)$ | $\mathcal{O}(n^2)$ |
| Chiba *et al.* [5]$^+$ | $\mathcal{O}(m)$ | $\alpha\mathcal{O}(mk)$ | $\mathcal{O}(m)$ |
| Makino *et al.* [13]$^+$ | $\mathcal{O}(mn)$ | $\alpha\mathcal{O}(\Delta^4)$ | $\mathcal{O}(m)$ |
| Chang *et al.* [4]$^+$ | $\mathcal{O}(m)$ | $\alpha\mathcal{O}(\Delta h^3)$ | $\mathcal{O}(m)$ |
| Makino *et al.* [13]$^+$ | $\mathcal{O}(n^2)$ | $\alpha\mathcal{O}(n^{2.37})$ | $\mathcal{O}(n^2)$ |
| Comin *et al.* [6]$^+$ | $\mathcal{O}(n^{5.37})$ | $\alpha\mathcal{O}(n^{2.09})$ | $\mathcal{O}(n^{4.27})$ |
| Conte *et al.* [7]$^+$ | $\mathcal{O}(m\log^{\mathcal{O}(1)}(m+n))$ | $\alpha\mathcal{O}(qd(\Delta+qd)\log^{\mathcal{O}(1)}(m+n))$ | $\mathcal{O}(q)$ |
| Conte *et al.* [7]$^+$ | $\mathcal{O}(m\log^{\mathcal{O}(1)}(m+n))$ | $\alpha\mathcal{O}(\min\{mk, qk\Delta\}\log^{\mathcal{O}(1)}(m+n))$ | $\mathcal{O}(k)$ |

$\Delta$ = max degree $k$ = degeneracy $q$ = maximum clique size
$\alpha$ = number of maximal cliques
h = smallest integer such that $|\{v \in V : |N(v)| \geq h\}|$, where $k \leq h \leq \Delta$.

in the parameters of the graph. For example, the algorithm of Johnson *et al.* [11] has setup time $\mathcal{O}(mn)$ and polynomial time delay $\mathcal{O}(mn)$. Thus, after the setup phase, this algorithm requires $\alpha\mathcal{O}(mn)$ time, $\alpha$ being the number of maximal cliques, to output all the maximal cliques of the graph. It is *output sensitive* since the enumeration time depends on the number of maximal cliques of the graph. All the algorithms that fall into this category are listed in the last nine rows of Table 1. For these specific algorithms, the time delay is equal to the enumeration time divided by $\alpha$, the number of maximal cliques.

**Our contribution.** Given a $k$-degenerate graph, we present an output sensitive algorithm with setup time $\mathcal{O}(n(k^2+s(k+1)))$ and enumeration time $\alpha\mathcal{O}((k+1)f(k+1))$, where $s(k+1)$ (resp. $f(k+1)$) is the preprocessing time (resp. enumeration time) for maximal clique enumeration in a general $(k+1)$-order graph. For example, using the algorithm of Makino *et al.* [13] which has setup time $\mathcal{O}((k+1)^2)$ and enumeration time $\mathcal{O}((k+1)^{2.37})$ for a $(k+1)$-order graph, our algorithm has setup time $\mathcal{O}(n(k^2+(k+1)^2))$ and enumeration time $\alpha\mathcal{O}((k+1)(k+1)^{2.37})$. Since in a $(k+1)$-order graph all the graph parameters (number of edges and vertices, the maximum degree, the clique size, etc.) are bounded by some function of $k$, our algorithm will always have enumeration time depending only on the degeneracy of the graph, whatever output sensitive algorithm of Table 1 we use. This is the first such algorithm.

On the downside, we were not able to prove that our algorithm has polynomial time delay. It also requires that the maximum cliques be stored. Thus, since the maximal cliques of a $k$-degenerate graph are of size at most $k+1$, our algorithm needs $\mathcal{O}((k+1)\alpha)$ space, besides the space needed to store the graph (in our case, the graph can be stored using adjacency lists). Further improvements are discussed in the conclusion.

The organization of the document is as follows. In Section 2 we introduce some notations and definitions. In Section 3 we prove basic results. These results are used in Section 4 to prove the correctness and time complexity of Algorithm 1, which is the main contribution of the paper.

# 2 Definitions

## 2.1 Graph terminologies

We consider graphs of the form $G = (V, E)$ which are simple, undirected, connected, with $n$ vertices and $m$ edges. We assume that they are stored in memory using adjacency lists. If $X \subset V$, the subgraph of $G$ induced by $X$ is denoted by $G[X]$. The vertex set of $G$ will be denoted by $V(G)$. The set $N(x)$ is called the *open neighborhood* of the vertex $x$ and consists of the vertex adjacent to $x$ in $G$. *The closed neighborhood* of $x$ is defined as $N[x] = N(x) \cup x$. Given an ordering $v_1, ..., v_n$ of the vertices of $G$, $V_i$ is the set of vertices following $v_i$ including itself in this ordering, that is, the set $\{v_i, v_{i+1}, ..., v_n\}$. By $G_i$ we denote the induced subgraph $G[N[v_i] \cap V_i]$. A graph is *k-degenerate* if there is an ordering $v_1, ..., v_n$ of its vertices such that for all $i$, $1 \leq i \leq n$, $|N(v_i) \cap V_i| \leq k$. The degeneracy ordering can be computed in $\mathcal{O}(m)$ time [1]. Given a graph $G$ we will denote by $\sigma_G$ its degeneracy ordering and if $x \in V(G)$ then $\sigma_G(x)$ will be the ranking of $x$ in $\sigma_G$.

## 2.2 Word terminologies

Let $\Sigma$ be an alphabet, that is, a non-empty finite set of symbols. Let a string $s$ be any finite sequence of symbols from $\Sigma$; $s$ will be a substring of a string $t$ if there exists strings $u$ and $v$ such that $t = usv$. If $u$ or $v$ is not empty then $s$ is a proper substring of $t$. It will be a *suffix* of $t$ if there exists a string $u$ such that $t = us$. If $u$ is not empty, $s$ is called a *proper suffix* of $t$.

# 3 Basic results

When not specified, we always assume that, given a $k$-degenerate graph $G$, we have its degeneracy ordering, denoted by $\sigma_G$. When referring to an ordering of the vertices, we always refer to $\sigma_G$. The family of subgraphs $G_i, i \in [n]$ described in Section 2.1 will always be constructed following the degeneracy ordering of $G$. Thus, these graphs have at most $k + 1$ vertices, since in a degeneracy ordering $v_1, ..., v_n$ of the vertices of $G$, the inequality $|N[v_i] \cap V_i| \leq k + 1$ holds.

We want to show in this section that, roughly, given some $k$-degenerate graph $G$, it is enough to compute all the maximal cliques of the induced subgraphs $G_i, i \in [n]$ to get all the maximal cliques of $G$. We first start by proving that the induced subgraphs $G_i, i \in [n]$ can be easily computed. To prove that we first introduce a special adjacency structure, in the following definition.

▶ **Definition 1.** Let $G = (V, E)$ be a $k$-degenerate graph. Assume that $G$ is given by the adjacency lists for each vertex. The degenerate adjacency list of a vertex $x \in V$ is its adjacency list in which every vertex that has lower ranking that $x$ in $\sigma_G$ has been deleted.

▶ **Lemma 2.** *The degenerate adjacency lists of a $n$-order $k$-degenerate graph $G$ can be computed in time $\mathcal{O}(m)$.*

**Proof.** Compute the degeneracy ordering $\sigma_G$ of $G$. As described before this can be done in $\mathcal{O}(m)$ time. Assume that we have the adjacency lists of $G$. Let $x \in V$ and let $d_x$ be its degree. In time $\mathcal{O}(d_x)$ remove all vertices from its adjacency lists that have lower ranking in $\sigma_G$. Repeat the procedure for all the vertices of the graph. This is done in total time $\mathcal{O}(m)$. ◀

▶ **Lemma 3.** *Given a $k$-degenerate graph $G$, there is an algorithm constructing the induced subgraphs $G_i, i \in [n]$ in time $\mathcal{O}(nk^2)$ and $\mathcal{O}(m)$ space.*

**Proof.** Compute the degenerate adjacency lists of $G$. This is done in time $\mathcal{O}(m)$ by Lemma 2. Observe first that the vertex set of graph $G_i, i \in [n]$ corresponds to $i$-th vertex of $\sigma_G$ plus all the vertices of its degenerate adjacency list. Thus it only remains to show how to compute the adjacency lists of each of these graphs. We proceed as follows. For every vertex $x \in V(G_i)$, go through its degenerate adjacency list and remove vertices which are not in the vertex set $V(G_i)$. Observe that this can be done in $\mathcal{O}(k)$ by coloring the vertices of $V(G_i)$ blue and removing non blue vertices from the degenerate adjacency list of $x$. This procedure takes time $\mathcal{O}(k^2)$ for each graph $G_i, i \in [n]$, thus in total, we need time $\mathcal{O}(nk^2 + m) = \mathcal{O}(nk^2)$, as $m = \mathcal{O}(nk)$. ◀

Now that we have seen how the induced subgraphs $G_i, i \in [n]$ can be constructed, we want to characterize their maximal cliques with respect to the maximal cliques of the graph. We show in the next two lemmas that the maximal cliques of graphs $G_i, i \in [n]$ which are not maximal in $G$ can be easily described.

▶ **Lemma 4.** *Let $G$ be a $k$-degenerate graph, $\sigma_G$ its degeneracy ordering, and let $K$ be a maximal clique of an induced subgraph $G_i, i \in [n]$. Clique $K$ is not a maximal clique of $G$ if and only if there exists a maximal clique $C$ of $G$ which is an induced subgraph of a $G_j$ with $j < i$ and such that $K$ is a strict induced subgraph of $C$.*

**Proof.** Let $\sigma_G$ be the degeneracy ordering of $G$. Assume that $K$ is a maximal clique of an induced graph $G_i$ for $i = 1, ..., n-k$ but is not a maximal clique of $G$. Observe that $v_i \in V(K)$ since, by definition, $v_i$ is connected to all the vertices of $V(G_i) \backslash v_i$. Since $K_i$ is a clique which is not maximal, then there exists a set $A$ of vertices such that $A \cap V(K) = \emptyset$ and the graph induced on $V(K) \cup A$ is a maximal clique of $G$. Let $v_j$ be the vertex of $A$ with lower ranking in $\sigma_G$. We have that $\sigma_G(v_j) < \sigma_G(v_i)$ since $v_j$ is connected to $v_i$ but does not appear in $V(G_i)$. (It does not appear otherwise $A \cap V(K) \neq \emptyset$). Let $C$ be the maximal clique induced on $V(K) \cup A$. Clique $C$ is an induced subgraph of $G_j$ with $j < i$. Observe that $K$ does not have $v_j$ in its vertex set. Therefore $K$ is a strict induced subgraph of $C$.

Conversely, assume that $K$ is a maximal clique of $G_i$ and $C$ a maximal clique of $G_j, j < i$ such that $K$ is an induced subgraph of $C$. Since $K$ is a strict induced subgraph of a maximal clique of $G$ then $K$ cannot be a maximal clique of $G$. ◀

▶ **Corollary 5.** *Let $G$ be a $k$-degenerate graph and let $K$ be a maximal clique of an induced subgraph $G_i, i \in [n]$ such that $K$ is not maximal in $G$. Let $C$ be a maximal clique of $G$ which is a subgraph of some graph $G_j, j < i$ and such that $K$ is a subgraph of $C$. Let $W(K)$ and $W(C)$ be the words obtained from the vertices of cliques $K$ and $C$ which have been ordered following $\sigma_G$. Then $W(K)$ is a proper suffix of $W(C)$.*

**Proof.** Observe first that by Lemma 4, clique $C$ is well defined. Since $K$ is a strict subgraph of $C$ then $V(K) \subset V(C)$. Recall that by definition, graph $G_i = G[N[v_i] \cap V_i]$ where $v_i$ is the $i$-th vertex of the degeneracy ordering. Observe that since $v_i$ is the vertex of $V(K)$

with smallest ranking in $\sigma_G$ then $v_i$ appears first in $W(K)$. We also have that $v_i \in V(C)$. Assume now by contradiction that $W(K)$ is not a proper suffix of $W(C)$. This implies that there exists at least a vertex $x \in V(C) \backslash V(K)$ that appears after vertex $v_i$ in $W(C)$. If that was not the case then $W(K)$ would have been a proper suffix of $W(C)$. This implies that vertex $x$ appears after vertex $v_i$ in $\sigma_G$. Observe now that $x$ is connected to all the vertices of $K$ since $x \in V(C)$ and $V(K) \subset V(C)$. Thus $G[V(K) \cup \{x\}]$ is a maximal clique of $G_i$, which is a contradiction by maximality of $K$. ◀

To conclude the section, we prove some additional results regarding the maximal cliques of graphs $G_i, i \in [n]$, in the next three lemmas.

▶ **Lemma 6.** *Let $G$ be a $k$-degenerate graph. Every clique which is maximal in some subgraph $G_i, i \in [n]$ is not maximal in any subgraph $G_j$ with $j \neq i$.*

**Proof.** Let $K$ be a maximal clique of some subgraph $G_i, i \in [n]$. Assume by contradiction that there exists a $j \in [n]$ with $j \neq i$ such that $K$ is maximal in $G_j$. Assume first that $i < j$. Since vertex $v_i$ is connected to all the vertices of graph $G_i$ then necessarily $v_i \in V(K)$ or $K$ is not maximal in $G_i$. Since we assumed $i < j$ then $v_i \notin V(G_j)$. This implies that $K$ cannot be a subgraph of $G_j$, which gives a contradiction in that case. Thus assume now that $j < i$. The proof is similar. Vertex $v_j$ which is connected to all the vertices of $G_j$ does not belong to graph $G_i$. Since $K$ is maximal in $G_i$ and since $v_j \notin V(K)$ then $K$ cannot be maximal in $G_j$. ◀

▶ **Lemma 7.** *Let $G$ be a $k$-degenerate graph, $\sigma_G$ its degeneracy ordering. Every maximal clique of $G$ is a subgraph of exactly one graph $G_i, i \in [n]$.*

**Proof.** let $K$ be some maximal clique of $G$. We first prove that $K$ is a subgraph of at least a subgraph $G_i, i \in [n]$. Let $x \in V(K)$ be the vertex of $K$ which has minimum ranking in $\sigma_G$. Observe now that clique $K$ is subgraph of graph $G_{\sigma_G(x)}$. The fact that clique $K$ is a subgraph of at most a graph $G_i, i \in [n]$ is a consequence of Lemma 6. ◀

▶ **Lemma 8.** *Let $G$ be a $k$-degenerate graph. Let $G_i$, $i \in [n]$ be the family of induced subgraphs as defined in Section 2.1 and constructed in Lemma 3. Let $\alpha$ denote the number of maximal cliques of $G$ and $\alpha_i$ the number of maximal cliques of graph $G_i$. We have that $\sum_{j=1}^{n} \alpha_j \leq \alpha(k+1)$.*

**Proof.** Let $max_i$ denotes the number of maximal cliques of $G_i, i \in [n]$ which are maximal in $G$ and $Nmax_i$ the number of maximal cliques of $G_i, i \in [n]$ which are not maximal in $G$. We have that $\alpha_i = max_i + Nmax_i$. By Lemma 7, every maximal clique of $G$ is a subgraph of exactly one graph $G_i, i \in [n]$. This implies that $\sum_{j=1}^{n} max_j = \alpha$. Let $X$ be the set of cliques which are maximal in some graph $G_i, i \in [n]$ but not maximal in $G$ and let $x \in X$. By Lemma 5, the word obtained from the vertices of $x$ which have been ordered following $\sigma_G$ is a proper suffix of the word obtained from ordering the vertices, following $\sigma_G$, of some maximal clique of $G$. This implies that $X$ is of size at most $k\alpha$ since a maximum clique of a $k$-degenerate has at most $k+1$ vertices and that a word with $k+1$ letters has at most $k$ proper suffixes. To conclude the proof, Lemma 6 implies that clique $x$ is maximal in an unique graph $G_i, i \in [n]$ which implies that $\sum_{j=1}^{n} Nmax_j \leq k\alpha$. Thus in total $\sum_{j=1}^{n} \alpha_j \leq \alpha + \alpha k = \alpha(k+1)$. ◀

**Algorithm 1:**

**Data:** A graph $G$.
**Result:** All the maximal cliques of $G$.

```
1  Compute k the degeneracy of G and σ_G.
2  Construct the graphs G_i, i ∈ [n].
3  Initialize T an empty generalized suffix tree.
4  for j = 1 to n do
5      Compute all maximal cliques of graph G_j.
6      for every maximal clique K of graph G_j do
7          Order the vertices of K following σ_G
8          Search for K in T.
9          if there is a match then
10             Reject it.
11         else
12             Insert the proper suffixes of K in T.
13             Output K.
```

## 4 Algorithm for maximal clique enumeration

Before we describe the algorithm, we introduce suffix trees. We need a data structure to store the proper suffixes of all maximal cliques. Given a word of size $n$, we can construct a suffix tree containing all its suffixes in space and time $\mathcal{O}(n)$, see [14, 18, 19]. For a set of words $X = \{x_1, x_2, ..., x_r\}$, it is possible to construct a generalized suffix tree containing all the suffixes of the words in $X$, in an online fashion, in space and time $\mathcal{O}(\sum_{i=1}^{r} |x_i|)$, see [10, chapter 6] and [18] for instance.

The outline of the algorithm is the following. We start by computing the induced subgraphs $G_i, i \in [n]$. Then we consider each such subgraph, starting from $G_1$ up to $G_n$. We find all its maximal cliques and try to find them in a generalized suffix tree. If there is a match, the clique is rejected, otherwise it is outputted and its proper suffixes are inserted into the generalized suffix tree. The procedure is described in Algorithm 1. Its correctness is proved in Theorem 9 and its time complexity in Theorem 10.

▶ **Theorem 9.** *Given a $k$-degenerate graph $G$, Algorithm 1 outputs exactly all its maximal cliques, without duplication.*

**Proof.** By Lemma 7, every maximal clique of the graph is a subgraph of exactly one graph $G_i, i \in [n]$. Thus, every maximal clique $K$ of the graph is considered exactly once in Line 6 of the algorithm. If $K$ is matched in the generalized suffix tree at Line 7 then the vertices of $K$ ordered following $\sigma_G$ form a proper suffix of some clique of the graph. This contradicts the fact that $K$ is maximal in $G$. Thus, every maximal clique is outputted exactly once. Moreover, all the proper suffixes of all the maximal cliques are stored in the generalized tree. By Corollary 5 the word obtained from a maximal clique in some graph $G_i, i \in [n]$ which is not maximal in $G$ form a proper suffix of the world obtained from some maximal clique of $G$. Thus, all such cliques will be rejected in Line 9 of Algorithm 1. In conclusion, we proved that only the maximum cliques of $G$ are outputted, without duplication. ◀

▶ **Theorem 10.** *Given a $k$-degenerate graph $G$, Algorithm 1 has setup time $\mathcal{O}(n(k^2+s(k+1)))$ and enumeration time $\alpha\mathcal{O}((k+1)f(k+1))$ where $\alpha$ is the number of maximal cliques of $G$ and $s(k+1)$ (resp. $f(k+1)$) is the preprocessing time (resp. enumeration time) of maximal clique enumeration in a general $(k+1)$-order graph.*

**Proof.** Computing the degeneracy of $G$ in Line 1 is done in $\mathcal{O}(m)$ time. Constructing the graphs $G_i, i \in [n]$ in Line 2 is done in $\mathcal{O}(nk^2)$, by Lemma 3. To compute all the maximal cliques of every graph $G_i, i \in [n]$, we can use any output sensitive algorithm of Table 1. The chosen algorithm has preprocessing time $s(|V(G_i)|) = \mathcal{O}(s(k+1))$ and enumeration time $f(|V(G_i)|) = \mathcal{O}(f(k+1))$ for each graph $G_i, i \in [n]$ since these graphs have at most $k+1$ vertices. We first preprocess every such graph $G_i$ in total time $\mathcal{O}(n * s(k+1))$. Thus the preprocessing phase takes time $\mathcal{O}(nk^2 + m + n * s(k+1)) = \mathcal{O}(n(k^2 + s(k+1)))$. Then we enumerate all the maximal cliques of the graphs $G_i, i \in [n]$ in total time $(\sum_{j=1}^{n} \alpha_j) * \mathcal{O}(f(k+1))$ where $\alpha_j$ is the number of maximal cliques of graph $G_j$. By Lemma 8, $\sum_{j=1}^{n} \alpha_j \leq (k+1)\alpha$. Thus enumerating all the maximal cliques of the graphs $G_i, i \in [n]$ takes total time $\alpha\mathcal{O}((k+1)f(k+1))$. Searching and inserting the generated cliques in the suffix tree takes total time $\alpha\mathcal{O}((k+1)^2)$. In conclusion, Algorithm 1 has preprocessing time $\mathcal{O}(n(k^2+s(k+1)))$ and enumeration time $\alpha\mathcal{O}((k+1)f(k+1))$, as claimed. ◀

## 5 Conclusion

We presented the first output sensitive algorithm for maximal clique enumeration whose enumeration time depends only on the degeneracy of the graph. We were not able to prove that it has polynomial time delay. Our intuition is that in its current state, our algorithm has time delay $\mathcal{O}(k\alpha)$. Thus, we first ask whether this is true or not and if yes, if there is a way to modify our approach as to get a polynomial time delay. The second question that we ask is whether or not we can improve the space complexity. In its current state, our algorithm requires that the maximal cliques be stored. Can we modify our approach as to avoid that?

## References


1. V. Batagelj and M. Zaversnik. An $\mathcal{O}(m)$ algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003. URL: `http://arxiv.org/abs/cs.DS/0310049`.
2. C. Bron and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, 1973. `doi:10.1145/362342.362367`.
3. F. Cazals and C. Karande. A note on the problem of reporting maximal cliques. *Theoretical Computer Science*, 407(1-3):564–568, 2008.
4. L. Chang, J. X. Yu, and L. Qin. Fast maximal cliques enumeration in sparse graphs. *Algorithmica*, 66(1):173–186, 2013.
5. N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985.
6. C. Comin and R. Rizzi. An improved upper bound on maximal clique listing via rectangular fast matrix multiplication. *arXiv preprint arXiv:1506.01082*, 2015.
7. A. Conte, R. Grossi, A. Marino, and L. Versari. Sublinear-space bounded-delay enumeration for massive network analytics: Maximal cliques. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 148, pages 1–148, 2016.
8. David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM Journal of Experimental Algorithmics*, 18, 2013. `doi:10.1145/2543629`.

**9** M. Farach. Optimal suffix tree construction with large alphabets. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, FOCS '97, pages 137–, Washington, DC, USA, 1997. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=795663.796326.

**10** D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, 1997.

**11** David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988. doi:10.1016/0020-0190(88)90065-8.

**12** D. R. Lick and A. T. White. $d$-degenerate graphs. *Canad. J. Math.*, 22:1082–1096, 1970. URL: http://www.smc.math.ca/cjm/v22/p1082.

**13** K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *Scandinavian Workshop on Algorithm Theory*, pages 260–272. Springer, 2004.

**14** Edward M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976. doi:10.1145/321941.321946.

**15** J. W Moon and L. Moser. On cliques in graphs. *Israel journal of Mathematics*, 3(1):23–28, 1965.

**16** Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.*, 363(1):28–42, 2006. doi:10.1016/j.tcs.2006.06.015.

**17** S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977.

**18** Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995. doi:10.1007/BF01206331.

**19** Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 1–11. IEEE Computer Society, 1973. doi:10.1109/SWAT.1973.13.

# Merging Nodes in Search Trees: an Exact Exponential Algorithm for the Single Machine Total Tardiness Scheduling Problem*

**Lei Shang[1], Michele Garraffa[2], Federico Della Croce[3], and Vincent T'Kindt[4]**

1 **Université François Rabelais de Tours, Laboratoire d'Informatique (EA 6300), ERL CNRS OC 6305, Tours, France**
shang@univ-tours.fr
2 **Politecnico di Torino, DAUIN, Torino, Italy**
michele.garraffa@polito.it
3 **Politecnico di Torino, DIGEP, Torino, Italy**
federico.dellacroce@polito.it
4 **Université François Rabelais de Tours, Laboratoire d'Informatique (EA 6300), ERL CNRS OC 6305, Tours, France**
tkindt@univ-tours.fr

## Abstract

This paper proposes an exact exponential algorithm for the problem of minimizing the total tardiness of jobs on a single machine. It exploits the structure of a basic branch-and-reduce framework based on the well known Lawler's decomposition property. The proposed algorithm, called branch-and-merge, is an improvement of the branch-and-reduce technique with the embedding of a node merging operation. Its time complexity is $\mathcal{O}^*(2.247^n)$ keeping the space complexity polynomial. The branch-and-merge technique is likely to be generalized to other sequencing problems with similar decomposition properties.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, F.2.0 General (Analysis of Algorithms and Problem Complexity), G.2.1 Combinatorics

**Keywords and phrases** Exact exponential algorithm, Single machine total tardiness, Branch-and-merge

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.28

## 1 Introduction

The challenge of designing exact exponential algorithms for NP-hard problems is attracting more and more researchers, particularly since the beginning of this century. For a survey on the most effective techniques in designing exact exponential algorithms, readers are kindly referred to Woeginger's paper [13] and to the book by Fomin and Kratsch [4]. In spite of the growing interest on exact exponential algorithms, few results are yet known on scheduling problems, see the survey of Lenté et al. [8].[1] This paper focuses on a pure sequencing problem, the single machine total tardiness problem, denoted by $1||\sum T_j$. In this problem, a

* A full version of the paper is available at [5], https://hal.archives-ouvertes.fr/hal-01477835.
[1] Recent results on Parameterized Algorithms of scheduling problems can be found at http://fpt.wikidot.com/operations-research.


jobset $N = \{1, 2, \ldots, n\}$ of $n$ jobs must be scheduled on a single machine. For each job $j$, a processing time $p_j$ and a due date $d_j$ are given. The problem asks for arranging the jobset in a sequence $S$ so as to minimize $T(N, S) = \sum_{j=1}^{n} T_j = \sum_{j=1}^{n} \max\{C_j - d_j, 0\}$, where $C_j$ is the completion time of job $j$ in sequence $S$. The $1||\sum T_j$ problem is NP-hard in the ordinary sense [2]. It has been extensively studied in the literature. The current state-of-the-art exact method [11] solves to optimality problems with up to 500 jobs. Its complexity was not discussed in [11] but will be analyzed in this paper. In [7] an exact pseudo-polynomial dynamic programming algorithm was proposed with complexity $\mathcal{O}(n^4 \sum p_i)$. Also, the standard technique of doing dynamic programming across the subsets (see, for instance, [4]) applies and runs with complexity $\mathcal{O}(n^2 2^n)$ both in time and in space. We refer to [6] for a comprehensive survey on the problem. In the rest of the paper, the $\mathcal{O}^*(\cdot)$ notation [13], commonly used in the context of exact exponential algorithms, is used. Let $T(\cdot)$ be a super-polynomial and $p(\cdot)$ be a polynomial, both on integers. In what follows, for an integer $n$, we express running-time bounds of the form $\mathcal{O}(p(n) \cdot T(n)))$ as $\mathcal{O}^*(T(n))$. As an example, the complexity of dynamic programming across the subsets for the total tardiness problem can be expressed as $\mathcal{O}^*(2^n)$. The aim of this work is to design a faster exact exponential algorithm running in $\mathcal{O}^*(c^n)$ ($c$ being a constant) and polynomial space, exploiting known decomposition properties of the problem. The designed algorithm, making use of a new technique called branch-and-merge that avoids the solution of several equivalent subproblems in the branching tree, is shown to have a complexity $\mathcal{O}^*(2.247^n)$ in time and requires polynomial space. We also provide a complexity analysis of the state-of-the-art exact algorithm [11], which runs in $\mathcal{O}^*(2.4143^n)$ in time.

## 2 Preliminaries

We recall some basic properties of the total tardiness problem and related notation. Given a jobset $N = \{1, 2, \ldots, n\}$, let $(1, 2, \ldots, n)$ be a LPT (Longest Processing Time first) sequence, where $i < j$ whenever $p_i > p_j$ (or $p_i = p_j$ and $d_i \leq d_j$). Let also $([1], [2], \ldots, [n])$ be an EDD (Earliest Due Date first) sequence, where $i < j$ whenever $d_{[i]} < d_{[j]}$ (or $d_{[i]} = d_{[j]}$ and $p_{[i]} \leq p_{[j]}$). The order of jobs having identical processing time and due date should be fixed arbitrarily. Jobs are processed with no interruption starting from time zero. Let $B_j$ and $A_j$ be the sets of jobs that precede and follow job $j$ in an optimal sequence being constructed. Correspondingly, the completion time of job $j$, $C_j = \sum_{k \in B_j} p_k + p_j$. Also, if job $j$ is assigned to position $k$, $C_j(k)$ denotes the corresponding completion time and $B_j(k)$ and $A_j(k)$ the sets of predecessors and successors of $j$, respectively. The following known theoretical properties hold.

▶ **Property 1.** *[3] Consider two jobs $i$ and $j$ with $p_i < p_j$. Then, in at least one optimal schedule, $i$ precedes $j$ if $d_i \leq \max\{d_j, C_j\}$, otherwise $j$ precedes $i$ if $d_i + p_i > C_j$.*

▶ **Property 2.** *[7] Let job 1 in LPT order correspond to job $[k]$ in EDD order. Then, job 1 can be set only in positions $h \geq k$ and the jobs preceding and following job 1 are uniquely determined as $B_1(h) = \{[1], [2], \ldots, [k-1], [k+1], \ldots, [h]\}$ and $A_1(h) = \{[h+1], \ldots, [n]\}$.*

▶ **Property 3.** *[7, 9, 10] Consider $C_1(h)$ for $h \geq k$. Job 1 cannot be set in position $h \geq k$ if:*
***(a)*** *$C_1(h) \geq d_{[h+1]}$, $h < n$;*
***(b)*** *$C_1(h) < d_{[r]} + p_{[r]}$, for some $r = k+1, \ldots, h$.*

▶ **Property 4.** *([12]) For any pair of adjacent positions $(i, i+1)$ that can be assigned to job 1, at least one of them is eliminated by Property 3.*

**Algorithm 1** Total Tardiness Branch-and-Reduce (TTBR)

**Input:** $N = \{1, ..., n\}$ is the problem to be solved
**function** TTBR$(S, t)$
  $seqOpt \leftarrow$ a random sequence of jobs
  $l \leftarrow$ the longest job in $N$
  **for** $i = 1$ to n **do**
    Branch by assigning job $l$ to position $i$ if not discarded by Property 3
    $seqLeft \leftarrow$ TTBR$(B_l(i), t)$
    $seqRight \leftarrow$ TTBR$(A_l(i), t + \sum_{k \in B_l(i)} p_k + p_l)$
    $seqCurrent \leftarrow$ concatenation of $seqLeft$, $l$ and $seqRight$
    $seqOpt \leftarrow$ best solution between $seqOpt$ and $seqCurrent$
  **end for**
  **return** $seqOpt$
**end function**

A basic branch-and-reduce algorithm TTBR (Total Tardiness Branch-and-Reduce) can be designed by exploiting Property 2, which allows to decompose the problem into two smaller subproblems when the position of the longest job $l$ is given and by taking into account Property 4 which states that for each pair of adjacent positions $(i, i+1)$, at least one of them can be discarded. The basic idea is to iteratively branch by assigning job $l$ to every possible position $\{1, ..., n\}$, discarding ineligible positions by means of the elimination rules of Property 3, and correspondingly decompose the problem. Each time a certain position $i$ is selected for job $l$, two different subproblems are generated, corresponding to schedule the jobs before $l$ (inducing subproblem $B_l(i)$) and after $l$ (inducing subproblem $A_l(i)$), respectively. The algorithm operates by applying to any given jobset $S$ starting at time $t$ function $TTBR(S, t)$ that computes the corresponding optimal solution. With this notation, the original problem is indicated by $N = \{1, ..., n\}$ and the optimal solution is reached when function $TTBR(N, 0)$ is computed. The algorithm proceeds by solving the subproblems along the branching tree according to a depth-first strategy and runs until all the leaves of the search tree have been reached. Finally, it provides the best solution found as an output. Algorithm 1 summarizes the structure of this approach, while Proposition 5 states its worst-case complexity.

▶ **Proposition 5.** *Algorithm TTBR runs in* $\mathcal{O}^*((1+\sqrt{2})^n) = \mathcal{O}^*(2.4143^n)$ *time and polynomial space in the worst case.*

**Proof.** We refer to problems where $n$ is odd, but the analysis for $n$ even is substantially the same. Whenever the longest job 1 is assigned to the first and the last position of the sequence, two subproblems of size $n-1$ are generated. For each $2 \leq i \leq n-1$, two subproblems with size $i-1$ and $n-i$ are generated. Hence, the total number of generated subproblems is at most $2n-2$. This would induce the following recurrence for the running time $T(n)$:

$$T(n) = 2T(n-1) + 2T(n-2) + ... + 2T(2) + 2T(1) + \mathcal{O}(p(n)) \tag{1}$$

However, Property 4 indicates that the elimination rules of Property 3 discard at least one position for every pair of adjacent positions. The worst case occurs when the largest possible subproblems are kept that is when subproblems with size $n-1, n-3, n-5, \ldots$ (that arise by branching on positions $i$ and $n-i+1$ with $i$ odd) are kept and correspondingly subproblems with size $n-2, n-4, n-6, \ldots$ are discarded. This induces a recurrence of the type:

$$T(n) = 2T(n-1) + 2T(n-3) + ... + 2T(4) + 2T(2) + \mathcal{O}(p(n)) \tag{2}$$

By replacing $n$ with $n-2$, the following expression is derived:

$$T(n-2) = 2T(n-3) + 2T(n-5) + ... + 2T(4) + 2T(2) + \mathcal{O}(p(n-2)) \tag{3}$$

Plugging expression 3 into expression 2, we get:

$$T(n) = 2T(n-1) + T(n-2) + \mathcal{O}(p(n)) \tag{4}$$

that induces as complexity $\mathcal{O}^*((1+\sqrt{2})^n) = \mathcal{O}^*(2.4143^n)$. The space requirement is polynomial since the branching tree is explored according to a depth-first strategy. ◀

The current state-of-the-art algorithm described in [11], noted hereafter as BB2001, is a branch and bound algorithm having a similar structure as that of TTBR. The main difference is that in BB2001, besides of the decomposition rule given in Property 2, another decomposition rule, based on Property 6, is applied simultaneously on each branching. We provide in Proposition 7 our analysis on the time complexity of BB2001, since this is not discussed in [11]. Notice that even though the time complexity of TTBR is the same as BB2001, the former one serves as a basis of the final algorithm branch-and-merge.

▶ **Property 6.** *[1] Let job $k$ in LPT sequence correspond to job [1] in EDD sequence. Then, job $k$ can be set only in positions $h \leq (n-k+1)$ and the jobs preceding job $k$ are uniquely determined as $B_k(h)$, where $B_k(h) \subseteq \{k+1, k+2, \ldots, n\}$ and $\forall i \in B_k(h), j \in \{n, n-1, \ldots, k+1\} \setminus B_k(h)$, $d_i \leq d_j$*

▶ **Proposition 7.** *Algorithm BB2001 runs in $\mathcal{O}^*(2.4143^n)$ time and polynomial space in the worst case.*

**Proof.** Before branching on a node, BB2001 first computes the possible positions for the longest job and the job with smallest due date. Then a new branch is created by assigning a pair of compatible positions to these two jobs. We consider two cases as follows.

Firstly, consider the case where job $1 = [n]$. The two decomposition rules become identical and if this condition is also verified in all subproblems, then the time complexity is $\mathcal{O}^*(2.4143^n)$ as proved in Proposition 5.

In the case where $1 \neq [n]$, the worst case occurs when $1 = [2]$ and $[n] = 2$, since in this case we have maximum available branching positions: job $[n]$ can be branched on position $i \in \{1, ..., n-1\}$ and job 1 can be branched on position $j \in \{2, ..., n\}$, with $i < j$ for each branching. Moreover, we recall that the Property 4 is also valid.

Three subproblems (left, middle and right) are created on each double branching (zero-sized problems are counted). For the sake of simplicity, we note $T(l, m, r) = T(l)+T(m)+T(r)$.

The following recurrence holds.

$$T(n) = \sum_{\substack{i=1 \\ i \text{ is odd}}}^{n-1} \sum_{\substack{j=i+1 \\ j \text{ is even}}}^{n} (T(i-1, j-i-1, n-j)) + \mathcal{O}(p(n)) \tag{5}$$

$$= T(0,0,n-2) + T(0,2,n-4) + T(0,4,n-6) + ... + T(0,n-2,0) + \tag{6}$$

$$T(2,0,n-4) + T(2,2,n-6) + ... + T(2,n-4,0) + \tag{7}$$

$$... \tag{8}$$

$$T(n-4,0,2) + T(n-4,2,0) + \tag{9}$$

$$T(n-2,0,0) + \tag{10}$$

$$\mathcal{O}(p(n)) \tag{11}$$

$$= 3 * (T(n-2) + 2T(n-4) + 4T(n-6) + ... + \frac{n}{2}T(0)) \tag{12}$$

$$\tag{13}$$

By applying a similar process of simplification as in the proof of Proposition 5, the following result is finally derived:

$$T(n) = 5T(n-2) - T(n-4). \tag{14}$$

Correspondingly, we have $T(n) = \mathcal{O}^*(\sqrt{\frac{5+\sqrt{21}}{2}}^n) = \mathcal{O}^*(2.1890^n)$. Therefore the worst case occurs when the two decomposition rules overlap, and the resulting time complexity is the same as TTBR, namely $\mathcal{O}^*(2.4143^n)$.

In terms of space complexity, BB2001 applies an extra technique called *Memorization* which makes use of exponential memory space for accelerating the solution. When this extra technique is not considered, the space complexity of BB2001 is also polynomial since depth-first exploration is adopted. ◀

## 3 Merging nodes in the search tree

In this section, we describe how to get an algorithm running with complexity $\mathcal{O}^*(2.247^n)$ in time and polynomial space by integrating a node-merging procedure into TTBR. The resulting algorithm will be called *branch-and-merge*. We recall that in TTBR the branching scheme is defined by assigning the longest unscheduled job to each available position and accordingly divide the problem into two subproblems. To facilitate the description of the algorithm, we focus on the worst-case scenario where the LPT sequence $(1, ..., n)$ coincides with the EDD sequence $([1], ..., [n])$: in this case no position can be eliminated by Property 2 at each branching.

Figure 1 shows how an input problem $\{1, ..., n\}$ is decomposed by the branching scheme of TTBR. Each node is labelled by the corresponding subproblem $P_j$ ($P$ denotes the input problem) and it is assumed in this example that Property 3 is not applied (for convenience purpose). Notice that from now on $P_{j_1,j_2,...,j_k}, 1 \leq k \leq n$, denotes the problem (node in the search tree) induced by the branching scheme of TTBR when the largest processing time job 1 is in position $j_1$, the second largest processing time job 2 is in position $j_2$ and so on till the k-th largest processing time job $k$ being placed in position $j_k$.

To roughly illustrate the guiding idea of the merging technique introduced in this section, consider Figure 1. Noteworthy, nodes $P_2$ and $P_{1,2}$ are identical except for the initial subsequence (21 vs 12). This fact implies, in this particular case, that the problem of

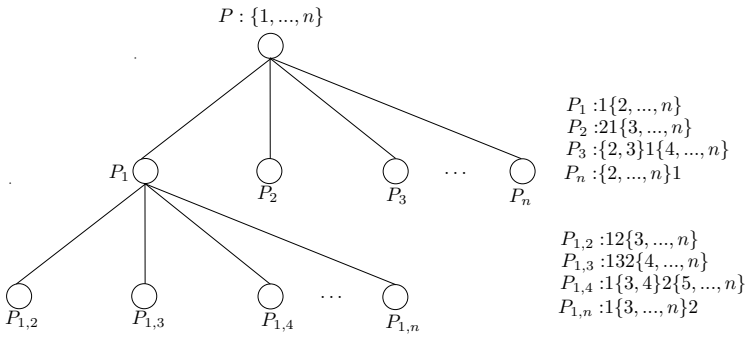



**Figure 1** The branching scheme of TTBR at the root node.

scheduling jobset $\{3, ..., n\}$ at time $p_1 + p_2$ is solved twice. This kind of redundancy can however be eliminated by merging node $P_2$ with node $P_{1,2}$ and creating a single node in which the best sequence among 21 and 12 is scheduled at the beginning and the jobset $\{3, ..., n\}$, starting at time $p_1 + p_2$, remains to be branched on. Furthermore, the best subsequence (starting at time $t = 0$) between 21 and 12 can be computed in constant time. Hence, the node created after the merging operation involves a constant time preprocessing step plus the search for the optimal solution of jobset $\{3, ..., n\}$ to be processed starting at time $p_1 + p_2$. We remark that, in the branching scheme of TTBR, for any constant $k \geq 3$, the branches corresponding to $P_i$ and $P_{n-i+1}$, with $i = 2, ..., k$, are decomposed into two problems where one subproblem has size $n - i$ and the other problem has size $i - 1 \leq k$. Correspondingly, the merging technique presented on problems $P_2$ and $P_{1,2}$ can be generalized to all branches inducing problems of sizes less than $k$. Notice that, by means of algorithm TTBR, any problem of size less than $k$ requires, to be solved, at most $\mathcal{O}^*(2.4143^k)$ time (that is constant time when $k$ is fixed). In the remainder of the paper, for any constant $k$, we denote by left-side branches the search tree branches corresponding to problems $P_1, ..., P_k$.

With respect to algorithm TTBR, the basic idea is to applying merging on the left-side branches (nodes $P_1$ to $P_k$) while Property 3 is applied on the remaining branches (nodes $P_{k+1}$ to $P_n$).

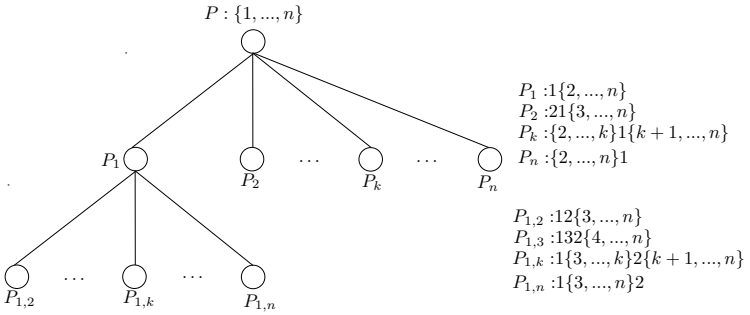
## 3.1 Merging left-side branches

We first illustrate the merging operations at the root node. The following lemma highlights two properties of the pairs of problems $P_j$ and $P_{1,j}$ with $2 \leq j \leq k$.

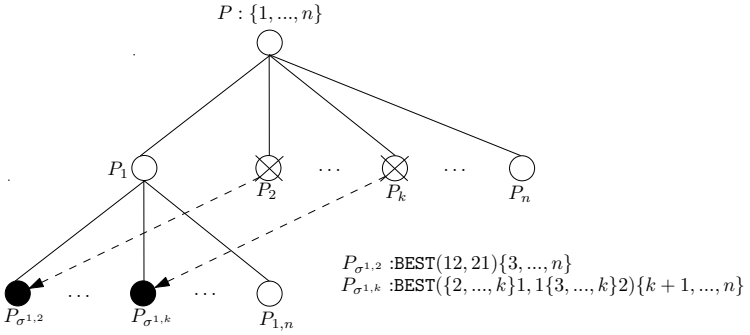
▶ **Lemma 8.** *For a pair of problems $P_j$ and $P_{1,j}$ with $2 \leq j \leq k$, the following conditions hold:*

1. *The solution of problems $P_j$ and $P_{1,j}$ involves the solution of a common subproblem which consists in scheduling jobset $\{j + 1, ..., n\}$ starting at time $t = \sum_{i=1,...,j} p_i$.*
2. *Both in $P_j$ and $P_{1,j}$, at most $k$ jobs have to be scheduled before jobset $\{j + 1, ..., n\}$.*

**Proof.** As problems $P_j$ and $P_{1,j}$ are respectively defined by $\{2, ..., j\}1\{j + 1, ..., n\}$ and $1\{3, ..., j\}2\{j + 1, ..., n\}$, the first part of the property is straightforward.
The second part can be simply established by counting the number of jobs to be scheduled before jobset $\{j+1, ..., n\}$ when $j$ is maximal, *i.e.* when $j = k$. In this case, jobset $\{k+1, ..., n\}$ has $(n - k)$ jobs which implies that $k$ jobs remain to be scheduled before that jobset. ◀




**(a)** Left-side branches of $P$ before performing the merging operations.




**(b)** Left-side branches of $P$ after performing the merging operations.

**Figure 2** Left-side branches merging at the root node.

Each pair of problems indicated in Lemma 8 can be merged as soon as they share the same subproblem to be solved. More precisely, $(k-1)$ problems $P_j$ (with $2 \leq j \leq k$) can be merged with the corresponding problems $P_{1,j}$.

Figure 2 illustrates the merging operations performed at the root node. For any given $2 \leq j \leq k$, problems $P_j$ and $P_{1,j}$ share the same subproblem $\{j+1,...,n\}$ starting at time $t = \sum_{i=1}^{j} p_i$. Hence, by merging the left part of both problems which is constituted by jobset $\{1,...,j\}$ having size $j \leq k$, we can delete node $P_j$ and replace node $P_{1,j}$ in the search tree by the node $P_{\sigma^{1,j}}$ which is defined as follows (Figure 2b):

- Jobset $\{j+1,...,n\}$ is the set of jobs on which it remains to branch.
- Let $\sigma^{1,j}$ be the sequence of branching positions on which the $j$ longest jobs $1,...,j$ are branched, that leads to the best jobs permutation between $\{2,...,j\}1$ and $1\{3,...,j\}2$ when these two are solved. This involves the solution of two problems of size at most $k-1$ (in $\mathcal{O}^*(2.4143^k)$ time by TTBR) and the comparison of the total tardiness value of the two sequences obtained.

In the following, we describe how to apply analogous merging operations on any node of the tree. With respect to the root node, the only additional consideration is that the children nodes of a generic node may have already been concerned by previous merging operations. Let us refer to `LEFT_MERGE` as the procedure which, for any node of the search tree, perform merging operations on its leftmost child branches. The `LEFT_MERGE` procedure operates based on a modified branching scheme, with respect to TTBR.

Let $\mathcal{L}_\sigma$ be a data structure associated to a problem $P_\sigma$. It represents a list of $k-1$ subproblems that result from a previous merging and are now the first $k-1$ children nodes of $P_\sigma$. When $P_\sigma$ is created by branching, $\mathcal{L}_\sigma = \emptyset$. When a merging operation sets the first

$k-1$ children nodes of $P_\sigma$ to $P_{\sigma^1}, ..., P_{\sigma^{k-1}}$, we set $\mathcal{L}_\sigma = \{P_{\sigma^1}, ..., P_{\sigma^{k-1}}\}$. As a conclusion, the following branching scheme for a generic node of the tree holds.

▶ **Definition 9.** The branching scheme for a generic node $P_\sigma$ is defined as follows:
- If $\mathcal{L}_\sigma = \emptyset$, use the branching scheme of TTBR;
- If $\mathcal{L}_\sigma \neq \emptyset$, extract problems from $\mathcal{L}_\sigma$ as the first $k-1$ branches, then branch on the longest job in the available positions from the $k$-th to the last according to Property 2.

This branching scheme, whenever necessary, will be referred to as **improved branching**.

Before describing how merging operations can be applied on a generic node $P_\sigma$, we highlight its structural properties by means of Proposition 10.

▶ **Proposition 10.** *Let $P_\sigma$ be a problem to branch on, and $\sigma$ be the permutation of positions assigned to jobs $1, \ldots, |\sigma|$, with $\sigma$ empty if no positions are assigned. The following properties hold:*

1. *$j^* = |\sigma| + 1$ is the job to branch on,*
2. *$j^*$ can occupy in the branching process, positions $\{\ell_b, \ell_b + 1, \ldots, \ell_e\}$, where*

$$\ell_b = \begin{cases} |\sigma| + 1 & \text{if } \sigma \text{ is a permutation of } 1, \ldots, |\sigma| \text{ or } \sigma \text{ is empty} \\ \rho_1 + 1 & \text{otherwise} \end{cases}$$

*with $\rho_1 = \max\{i : i > 0, \text{ positions } 1, \ldots, i \text{ are in } \sigma\}$ and*

$$\ell_e = \begin{cases} n & \text{if } \sigma \text{ is a permutation of } 1, \ldots, |\sigma| \text{ or } \sigma \text{ is empty} \\ \rho_2 - 1 & \text{otherwise} \end{cases}$$

*with $\rho_2 = \min\{i : \ i > \rho_1, \ i \in \sigma\}$*

**Proof.** According to the definition of the notation $P_\sigma$, $\sigma$ is a sequence of positions that are assigned to the longest $|\sigma|$ jobs. Since we always branch on the longest unscheduled job, the first part of the proposition is straightforward. The second part aims at specifying the range of positions that job $j^*$ can occupy. Two cases are considered depending on the content of $\sigma$:
- If $\sigma$ is a permutation of $1, \ldots, |\sigma|$, it means that the longest $|\sigma|$ jobs are set on the first $|\sigma|$ positions, which implies that the job $j^*$ should be branched on positions $|\sigma| + 1$ to $n$
- If $\sigma$ is not a permutation of $\{1, \ldots, |\sigma|\}$, it means that the longest $|\sigma|$ jobs are not set on consecutive positions. As a result, the current unassigned positions may be split into several ranges. As a consequence of Property 2, the longest job $j^*$ should necessarily be branched on the first range of free positions, that goes from $\rho_1$ to $\rho_2$. Under the worst-case scenario, let us consider as an example $P_{1,9,2,8}$, whose structure is $13\{5, \ldots, 9\}42\{10, \ldots, n\}$ and the job to branch on is 5. In this case, we have: $\sigma = (1, 9, 2, 8)$, $\ell_b = 3$, $\ell_e = 7$. It is easy to verify that 5 can only be branched on positions $\{3, \ldots, 7\}$ since 5 must stay before 4 as a direct result of Property 2. ◀

Corollary 11 emphasizes the fact that even though a node may contain several ranges of free positions, only the first range is the current focus since we only branch on the longest job in eligible positions.

▶ **Corollary 11.** *Problem $P_\sigma$ has the following structure:*

$$\pi\{j^*, \ldots, j^* + \ell_e - \ell_b\}\Omega$$

*with $\pi$ the subsequence of jobs on the first $\ell_b - 1$ positions in $\sigma$ and $\Omega$ the remaining subset of jobs to be scheduled after position $\ell_e$ (some of them can have been already scheduled). The merging procedure is applied on jobset $\{j^*, \ldots, j^* + \ell_e - \ell_b\}$ starting at time $t_\pi = \sum_{i \in \Pi} p_i$ where $\Pi$ is the jobset of $\pi$.*

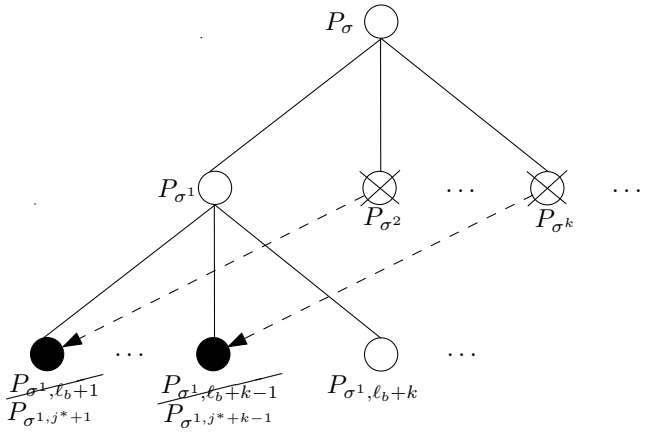



**Figure 3** Merging for a generic left-side branch.

The validity of merging on a general node still holds as indicated in Proposition 12, which extends the result stated in Proposition 8.

▶ **Proposition 12.** *Let $P_\sigma$ be a generic problem and let $\pi, j^*, \ell_b, \ell_e, \Omega$ be computed relatively to $P_\sigma$ according to Corollary 11. If $\mathcal{L}_\sigma=\emptyset$ the $j$-th child node $P_{\sigma^j}$ is $P_{\sigma,\ell_b+j-1}$ for $1\leq j\leq k$. Otherwise, the $j$-th child node $P_{\sigma^j}$ is extracted from $\mathcal{L}_\sigma$ for $1\leq j\leq k-1$, while it is created as $P_{\sigma,\ell_b+k-1}$ for $j=k$. For any pair of problems $P_{\sigma^j}$ and $P_{\sigma^1,\ell_b+j-1}$ with $2\leq j\leq k$, the following conditions hold:*

1. *Problems $P_{\sigma^j}$ and $P_{\sigma^1,\ell_b+j-1}$ with $2\leq j\leq k$ have the following structure:*
   - $P_{\sigma^j}$:
   $$\begin{cases} \pi^j\{j^*+j,\ldots,j^*+\ell_e-\ell_b\}\Omega & 1\leq j\leq k-1 \text{ and } \mathcal{L}_\sigma\neq\emptyset \\ \pi\{j^*+1,\ldots,j^*+j-1\}j^*\{j^*+j,\ldots,j^*+\ell_e-\ell_b\}\Omega & (1\leq j\leq k-1;\mathcal{L}_\sigma=\emptyset) \text{ or } j=k \end{cases}$$
   - $P_{\sigma^1,\ell_b+j-1}$:
   $\pi^1\{j^*+2,\ldots,j^*+j-1\}(j^*+1)\{j^*+j,\ldots,j^*+\ell_e-\ell_b\}\Omega$
2. *By solving all the problems of size less than $k$, that consist in scheduling the jobset $\{j^*+1,\ldots,j^*+j-1\}$ between $\pi$ and $j^*$ and in scheduling $\{j^*+2,\ldots,j^*+j-1\}$ between $\pi^1$ and $j^*+1$, both $P_{\sigma^j}$ and $P_{\sigma^1,\ell_b+j-1}$ consist in scheduling $\{j^*+j,...,j^*+\ell_e-\ell_b\}\Omega$ starting at time $t_{\pi^j}=\sum_{i\in\Pi^j}p_i$ where $\Pi^j$ is the jobset of $\pi^j$.*

**Proof.** The first part of the statement follows directly from Definition 9 and simply defines the structure of the children nodes of $P_\sigma$. The problem $P_{\sigma^j}$ is the result of a merging operation with the generic problem $P_{\sigma,\ell_b+j-1}$ and it could possibly coincide with $P_{\sigma,\ell_b+j-1}$, for each $j=1,...,k-1$. Furthermore, $P_{\sigma^j}$ is exactly $P_{\sigma,\ell_b+j-1}$ for $j=k$. The generic structure of $P_{\sigma,\ell_b+j-1}$ is $\pi\{j^*+1,\ldots,j^*+j-1\}j^*\{j^*+j,\ldots,j^*+\ell_e-\ell_b\}\Omega$, and the merging operations preserve the jobset to schedule after $j^*$. Thus, we have $\Pi^j=\Pi\cup\{j^*,...,j^*+j-1\}$ for each $j=1,...,k-1$, and this proves the first statement. Analougosly, the structure of $P_{\sigma^1,\ell_b+j-1}$ is $\pi^1\{j^*+2,\ldots,j^*+j-1\}(j^*+1)\{j^*+j,\ldots,j^*+\ell_e-\ell_b\}\Omega$. Once the subproblem before $j^*+1$ of size less than $k$ is solved, $P_{\sigma^1,\ell_b+j-1}$ consists in scheduling the jobset $\{j^*+j,...,j^*+\ell_e-\ell_b\}$ at time $t_{\pi^j}=\sum_{i\in\Pi^j}p_i$. In fact, we have that $\Pi^j=\Pi^1\cup\{j^*+2,\ldots,j^*+j-1\}\cup\{j^*+1\}=\Pi\cup\{j^*,\ldots,j^*+j-1\}$ . ◀

Analogously to the root node, each pair of problems indicated in Proposition 12 can be merged. Again, $(k-1)$ problems $P_{\sigma^j}$ (with $2\leq j\leq k$) can be merged with the corresponding

**Algorithm 2** `LEFT_MERGE` Procedure

**Input:** $P_\sigma$ an input problem of size $n$, with $\ell_b$, $j^*$ accordingly computed
**Output:** $Q$: a list of problems to branch on after merging
1: **function** `LEFT_MERGE`($P_\sigma$)
2: $Q \leftarrow \emptyset$
3: **for** $j$=1 to $k$ **do**
4: Create $P_{\sigma^j}$ ($j$-th child of $P_\sigma$) by the improved branching with the subproblem induced by jobset $\{j^*+1, \ldots, j^*+j-1\}$ solved if $\mathcal{L}_\sigma=\emptyset$ or $j=k$
5: **end for**
6: **for** $j$=1 to $k-1$ **do**
7: Create $P_{\sigma^{1j}}$ ($j$-th child of $P_{\sigma^1}$) by the improved branching with the subproblem induced by jobset $\{j^*+2, \ldots, j^*+j-1\}$ solved if $\mathcal{L}_{\sigma^1}=\emptyset$ or $j=k$
8: $\mathcal{L}_{\sigma^1} \leftarrow \mathcal{L}_{\sigma^1} \cup$ `BEST`$(P_{\sigma^{j+1}}, P_{\sigma^{1j}})$
9: **end for**
10: $Q \leftarrow Q \cup P_{\sigma^1}$
11: **return** $Q$
12: **end function**

problems $P_{\sigma^1,\ell_b+j-1}$. $P_{\sigma^j}$ is deleted and $P_{\sigma^1,\ell_b+j-1}$ is replaced by $P_{\sigma^{1,j^*+j-1}}$ (Figure 3), defined as follows:

- Jobset $\{j^*+j, ..., j^*+\ell_e-\ell_b\}\Omega$ is the set of jobs on which it remains to branch on.
- Let $\sigma^{1,j^*+j-1}$ be the sequence of positions on which the $j^*+j-1$ longest jobs $1, ..., j^*+j-1$ are branched, that leads to the best jobs permutation between $\pi^j$ and $\pi^1\{j^*+2, \ldots, j^*+j-1\}(j^*+1)$ for $2 \leq j \leq k-1$, and between $\pi\{j^*+1, \ldots, j^*+j-1\}j^*$ and $\pi^1\{j^*+2, \ldots, j^*+j-1\}(j^*+1)$ for $j=k$. This involves the solution of one or two problems of size at most $k-1$ (in $\mathcal{O}^*(2.4143^k)$ time by TTBR) and the finding of the sequence that has the smallest total tardiness value knowing that both sequences start at time 0.

The `LEFT_MERGE` procedure is presented in Algorithm 2. Notice that this algorithm takes as input one problem and produces as an output its first children nodes to branch on, which replace all its $k$ left-side children nodes.

▶ **Lemma 13.** *The `LEFT_MERGE` procedure returns one node to branch on in $\mathcal{O}(n)$ time and polynomial space. The corresponding problem is of size $n-1$.*

**Proof.** The creation of problems $P_{\sigma^1,\ell_b+j-1}$, $\forall j=2, \ldots, k$, can be done in $\mathcal{O}(n)$ time. The call of TTBR costs constant time. The `BEST` function called at line 8 consists in computing then comparing the total tardiness value of two known sequence of jobs starting at the same time instant: it runs in $\mathcal{O}(n)$ time. The overall time complexity of `LEFT_MERGE` procedure is then bounded by $\mathcal{O}(n)$ time as $k$ is a constant. Finally, as only node $P_{\sigma^1}$ is returned, its size is clearly $n-1$ when $P_\sigma$ has size $n$. ◀

## 3.2 Algorithm and complexity analysis

The main procedure TTBM (Total Tardiness Branch-and-Merge) is stated in Algorithm 3. It has a similar recursive structure as TTBR. However, each time a node is opened, the sub-branches required for the merging operations are generated, the subproblems of size less than $k$ are solved and the procedure `LEFT_MERGE` is called. Then, the algorithm proceeds recursively by extracting the next node from $Q$ with a depth-first strategy and terminates when $Q$ is empty.

▶ **Proposition 14.** *Algorithm TTBM runs in $\mathcal{O}^*(2.247^n)$ time and polynomial space.*

**Algorithm 3** Total Tardiness Branch-and-merge (TTBM)

**Input:** $P : \{1, ..., n\}$: input problem of size $n$
$k \geq 2$: an integer constant
**Output:** *seqOpt*: an optimal sequence of jobs
1: **function** TTBM($P$,$k$)
2: $Q \leftarrow P$
3: *seqOpt* $\leftarrow$ a random sequence of jobs
4: **while** $Q \neq \emptyset$ **do**
5: $P^* \leftarrow$ extract next problem from $Q$ (depth-first order)
6: **if** (the size of $P^* < k$) **then** Solve $P^*$ by calling TTBR
7: **end if**
8: **if** all jobs $\{1, ..., n\}$ are fixed in $P^*$ **then**
9: *seqCurrent* $\leftarrow$ the solution defined by $P^*$
10: *seqOpt* $\leftarrow$ best solution between *seqOpt* and *seqCurrent*
11: **else**
12: $Q \leftarrow Q \cup$ `LEFT_MERGE`$(P^*)$
13: **for** $i = k + 1, ..., n$ **do**
14: Create child node $P_i$ like in TTBR
15: **if** $P_i$ is not eliminated by Property 3 **then** $Q \leftarrow Q \cup P_i$
16: **end if**
17: **end for**
18: **end if**
19: **end while**
20: **return** *seqOpt*
21: **end function**

**Proof.** Starting from Algorithm 3, we can derive that for a given problem $P$ of size $n$, the $(k-1)$ first children nodes $P_2$ to $P_k$ are merged with children nodes of $P_1$. Consequently, among these nodes, only node $P_1$ remains as a child node of $P$. For the other $(n-k)$ children nodes, Property 3 is applied eliminating by the way one node over two. The worst-case is achieved when $n$ is odd and $k$ is even and we have the following recurrence:

$$T(n) = T(n-1) + (T(n-k-1) + T(k)) + (T(n-k-3) + T(k+2)) + ...$$

$$+(T(2) + T(n-3)) + T(n-1) + \mathcal{O}(p(n))$$

which can be reformulated as

$$T(n) = 2T(n-1) + T(n-3) + ... + T(n-k+1) + 2T(n-k-1) + ... + 2T(2) + \mathcal{O}(p(n))$$

Following the same approach used in the proof of Proposition 5, we plug $T(n-2)$ into the formula and we have

$$T(n) = 2T(n-1) + T(n-2) - T(n-3) + T(n-k-1) + \mathcal{O}(p(n)) - \mathcal{O}(p(n-2))$$

The solution of this recurrence is $T(n) = \mathcal{O}^*(c^n)$ with $c$ the largest root of

$$1 = \frac{2}{x} + \frac{1}{x^2} - \frac{1}{x^3} + \frac{1}{x^{k+1}}$$

When $k$ is large enough, the last term in the equation can be ignored, leading to a value of $c$ which tends towards 2.24698 as $k$ increases. More concretely, TTBM runs in $\mathcal{O}^*(2.247^n)$ when $k \geq 14$. ◀

## 4 Conclusions

In this paper an exact exponential algorithm for the single machine total tardiness problem was provided. By exploiting some inherent properties of the problem, we first

proposed a branch-and-reduce algorithm, denoted by TTBR running in $\mathcal{O}^*(2.4143^n)$ time and polynomial space. This algorithm is then improved by means of a merging technique leading to a time complexity $\mathcal{O}^*(2.247^n)$ and polynomial space. The resulting algorithm is named branch-and-merge. The merging technique is shown here on left-side branches only. However, at the price of a very long and technical study, also merging right-side branches can be considered leading to a general branch-and-merge algorithm converging to a $\mathcal{O}^*(2^n)$ worst-case time complexity (and still polynomial in space). The presentation of the right-side merging operation is omitted here due to paper length limitation. A complete description can be found in [5].

As a future development of this work, our aim is twofold. First, we aim at applying the branch-and-merge approach to other combinatorial optimization problems in order to establish its potential generalizability. Second, we want to explore the practical efficiency of branch-and-merge for the single machine total tardiness problem and check whether the merging mechanism and related memorization techniques may improve in practice the performances of known approaches such as the one in [11].

## References


1 Federico Della Croce, R Tadei, P Baracco, and A Grosso. A new decomposition approach for the single machine total tardiness scheduling problem. *Journal of the Operational Research Society*, pages 1101–1106, 1998.

2 Jianzhong Du and Joseph Y-T Leung. Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, 15(3):483–495, 1990.

3 Hamilton Emmons. One-machine sequencing to minimize certain functions of job tardiness. *Operations Research*, 17(4):701–715, 1969.

4 Fedor V Fomin and Dieter Kratsch. *Exact exponential algorithms*. Springer Science & Business Media, 2010.

5 Michele Garraffa, Lei Shang, Federico Della Croce, and Vincent T'Kindt. An Exact Exponential Branch-and-Merge Algorithm for the Single Machine Total Tardiness Problem. submitted to TCS, 2017. URL: `https://hal.archives-ouvertes.fr/hal-01477835`.

6 Christos Koulamas. The single-machine total tardiness scheduling problem: review and extensions. *European Journal of Operational Research*, 202(1):1–7, 2010.

7 Eugene L Lawler. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1:331–342, 1977.

8 Christophe Lenté, Mathieu Liedloff, Ameur Soukhal, and Vincent T'Kindt. Exponential Algorithms for Scheduling Problems. *HAL, https://hal.archives-ouvertes.fr/hal-00944382*, 2014. URL: `https://hal.archives-ouvertes.fr/hal-00944382`.

9 C.N Potts and L.N Van Wassenhove. A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters*, 1(5):177–181, 1982.

10 Wlodzimierz Szwarc. Single machine total tardiness problem revisited. *Creative and Innovative Approaches to the Science of Management, Quorum Books*, pages 407–419, 1993.

11 Wlodzimierz Szwarc, Andrea Grosso, and Federico Della Croce. Algorithmic paradoxes of the single-machine total tardiness problem. *Journal of Scheduling*, 4(2):93–104, 2001.

12 Wlodzimierz Szwarc and Samar K Mukhopadhyay. Decomposition of the single machine total tardiness problem. *Operations Research Letters*, 19(5):243–250, 1996.

13 Gerhard J. Woeginger. Exact Algorithms for NP-hard Problems: A Survey. In Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi, editors, *Combinatorial Optimization — Eureka, You Shrink!*, volume 2570 of *Lecture Notes in Computer Science*, pages 185–207. Springer Berlin Heidelberg, 2003.

# Computing Treewidth on the GPU*

**Tom C. van der Zanden[1] and Hans L. Bodlaender[2]**

1 **Department of Computer Science, Utrecht University, Utrecht, The Netherlands**
T.C.vanderZanden@uu.nl

2 **Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven and Department of Computer Science, Utrecht University, Utrecht, The Netherlands**
H.L.Bodlaender@uu.nl

## Abstract

We present a parallel algorithm for computing the treewidth of a graph on a GPU. We implement this algorithm in OpenCL, and experimentally evaluate its performance. Our algorithm is based on an $O^*(2^n)$-time algorithm that explores the elimination orderings of the graph using a Held-Karp like dynamic programming approach. We use Bloom filters to detect duplicate solutions.

GPU programming presents unique challenges and constraints, such as constraints on the use of memory and the need to limit branch divergence. We experiment with various optimizations to see if it is possible to work around these issues. We achieve a very large speed up (up to 77×) compared to running the same algorithm on the CPU.

**1998 ACM Subject Classification** G.2.2 Graph algorithms

**Keywords and phrases** treewidth, GPU, GPGPU, exact algorithms, graph algorithms, algorithm engineering

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.29

## 1 Introduction

Treewidth is a well known graph parameter that measures how 'tree-like' a graph is. The fact that many otherwise hard graph problems are linear time solvable on graphs of bounded treewidth [6] has been exploited in many theoretical and practical applications. For such applications, it is important to have efficient algorithms, that given a graph, determine the treewidth and find tree decompositions with optimal (or near-optimal) width.

The interest in practical algorithms to compute treewidth and tree decompositions is also illustrated by the fact that both the PACE 2016 and PACE 2017 challenges [12] included treewidth as one of the two challenge topics. Remarkably, while most tracks in the PACE 2016 challenge attracted several submissions [13], there were no submissions for the call for GPU-based programs for computing treewidth. Current sequential exact algorithms for treewidth are only practical when the treewidth is small (up to 4, see [17]), or when the graph is small (see [16, 4, 26, 14, 25]). As computing treewidth is NP-hard, an exponential growth of the running time is to be expected; unfortunately, the exact FPT algorithms that are known for treewidth are assumed to be impractical; e.g., the algorithm of [3] has a running

* Due to space constraints, several tables in this paper have been abridged or omitted. The complete set of results is presented in the full version of this paper, available on arXiv [23], https://arxiv.org/abs/1709.09990.


time of $2^{O(k^3)}n$. This creates the need for good parallel algorithms, as parallelism can help to significantly speed up the algorithms, and thus deal with larger graph sizes.

In this paper, we consider a practical parallel exact algorithm to compute the treewidth of a graph and a corresponding tree decomposition. The starting point of our algorithm is a sequential algorithm by Bodlaender et al. [4]. This algorithm exploits a characterization of treewidth in terms of the width of an *elimination ordering*, and gives a dynamic programming algorithm with a structure that is similar to the textbook Held-Karp algorithm for TSP [18].

Prior work on parallel algorithms for treewidth is limited to one paper, by Yuan [25], who implements a branch and bound algorithm for treewidth on a CPU with a (relatively) small number of cores. With the advent of relatively inexpensive consumer GPUs that offer more than an order of magnitude more computational power than their CPU counterparts, it is very interesting to explore how exact and fixed-parameter algorithms can take advantage of the unique capabilities of GPUs. We take a first step in this direction, by exploring how treewidth can be computed on the GPU.

Our algorithm is based on the elimination ordering characterization of treewidth. Given a graph $G = (V, E)$, we may *eliminate* a vertex $v \in V$ from $G$ by removing $v$ and turning its neighborhood into a clique, thus obtaining a new graph. One way to compute treewidth is to find an order in which to eliminate all the vertices of $G$, such that the maximum degree of each vertex (at the time it is eliminated) is minimized. This formulation is used by e.g. [16] to obtain a (worst-case) $O^*(n!)$-time algorithm. However, it is easy to obtain an $O^*(2^n)$-time algorithm by applying Held-Karp style dynamic programming as first observed by Bodlaender et al. [4]: given a set $S \subseteq V$, eliminating the vertices in $S$ from $G$ will always result in the same intermediate graph, regardless of the order in which the vertices are eliminated (and thus, the order in which we eliminate $S$ only affects the degrees encountered during its elimination). This optimization is used in the algorithms of for instance [15] and [25].

We explore the elimination ordering space in a breadth-first manner. This enables efficient parallelization of the algorithm: during each iteration, a wavefront of states (consisting of the sets of vertices $S$ of size $k$ for which there is a feasible elimination order) is expanded to the wavefront of the next level, with each thread of the GPU taking a set $S$ and considering which candidate vertices of the graph can be added to $S$. Since multiple threads may end up generating the same state, we then use a bloom filter to detect and remove these duplicates.

To reduce the number of states explored, we experiment with using the minor-min-width heuristic [16], for which we also provide a GPU implementation. Whereas normally this heuristic would be computed by operating on a copy of the graph, we instead compute it using only the original graph and a smaller auxiliary data structure, which may be more suitable for the GPU. We also experiment with several techniques unique to GPU programming, such as using shared/local memory (which can best be likened to the cache of a CPU) and rewriting nested loops into a single loop to attempt to improve parallelism.

We provide an experimental evaluation of our techniques, on a platform equipped with a Intel Core i7-6700 CPU (3.40GHz) with 32GB of RAM (4x8GB DDR4), and an NVIDIA GeForce GTX 1060 with 6GB GDDR5 memory (Manufactured by Gigabyte, Part Number `GV-N1060WF2OC-6GD`). Our algorithm is implemented in OpenCL (and thus highly portable). We achieve a very large speedup compared to running the same algorithm on the CPU.

## 2 Preliminaries

### Treewidth

For a detailed description of treewidth and its characterization, we refer to [11]. Our algorithm is based on the $O(2^n nm)$-time algorithm of Bodlaender et al. [4]. Though the characterization in terms of tree decomposition is more common, we recall only the characterization in terms of elimination orderings that is used by this algorithm:

Let $G = (V, E)$ be a graph with vertices $v_1, \ldots v_n$. An *elimination ordering* is a permutation $\pi : V \to \{1, \ldots, n\}$ of the vertices of $G$. The *treewidth* of $G$ is defined as $\min_\pi \max_v |Q(\{u \in V \mid \pi(u) < \pi(v)\}, v)|$, where $Q(S, v)$ is the set of vertices $\{u \in V \setminus S \mid$ there is a path $v, p_1, \ldots, p_m, u$ such that $p_1, \ldots, p_m \in S\}$, i.e., $Q(S, v)$ is the subset of vertices of $V \setminus S$ reachable from $v$ by paths whose internal vertices are in $S$.

An alternative view of this definition is that given a graph $G$, we can *eliminate* a vertex $v$ by removing it from the graph, and turning its neighborhood into a clique. The treewidth of a graph is at most $k$, if there exists an elimination order such that all vertices have degree at most $k$ at the time they are eliminated.

### GPU Terminology

Parallelism on a GPU is achieved by executing many *threads* in parallel. These threads are grouped into *warps* of 32 threads. The 32 threads that make up a warp do not execute independently: they share the same program counter, and thus must always execute the same "line" of code (thus, if different threads need to execute different branches in the code, this execution is serialized - this phenomenon, called *branch divergence*, should be avoided). The unit that executes a single thread is called a *CUDA core*.

We used a GTX1060 GPU, which is based on the Pascal architecture [20]. The GTX1060 has 1280 CUDA cores, which are distributed over 10 Streaming Multiprocessors (SMs). Each SM thus has 128 CUDA cores, which can execute up to 4 warps of 32 threads simultaneously. However, a larger number of warps may be assigned to an SM, enabling the SM to switch between executing different warps, for instance to hide memory latency.

Each SM has 256KiB[1] of register memory (which is the fastest, but which registers are addressed must be known at compile time, and thus for example dynamically indexing an array stored in register memory is not possible), 96KiB of shared memory (which can be accessed by all threads executing within the thread block) and 48KiB of L1 cache.

Furthermore, we have approximately 6GB of global memory available which can be written to and read from by all threads, but is very slow (though this is partially alleviated by caching and latency hiding). Shared memory can, in the right circumstances, be read and written much faster, but is still significantly slower than register memory. Finally, there is also texture memory (which we do not use) and constant memory (which is a cached section of the global memory) that can be used to store constants that do not change over the kernel's execution (we use constant memory to store the adjacency lists of the graph).

Shared memory resides physically closer to the SM than global memory, and it would thus make sense to call it "local" memory (in contrast to the more remote global memory). Indeed, OpenCL uses this terminology. However, NVIDIA/CUDA confusingly use "local memory" to indicate a portion of the global memory dedicated to a single thread.

[1] A *kibibyte* is $2^{10}$ bytes.

# 3 The Algorithm

## 3.1 Computing Treewidth

Our algorithm works with an iterative deepening approach: for increasing values of $k$, it repeatedly runs an algorithm that tests whether the graph has treewidth at most $k$. This means that our algorithm is in practice much more efficient than the worst-case $O^*(2^n)$ behavior shown by [4], since only a small portion of the $2^n$ possible subsets may be feasible for the target treewidth $k$. A similar approach (of solving the decision version of the problem for increasing values of $k$) was also used by Tamaki [22], who refers to it as *positive-instance driven dynamic programming*.

This algorithm lends itself very well to paralellization, since the subsets can be evaluated (mostly) independently in parallel. This comes at the cost of slightly reduced efficiency (in terms of the number of states expanded) compared to a branch and bound approach (e.g. [14, 25, 26]) since the states with treewidth $< k-1$ are expanded more than once. However, even a branch and bound algorithm needs to expand all of the states with treewidth $k-1$ before it can conclude that treewidth $k$ is optimal, so the main advantage of branch and bound is that it can settle on a solution with treewidth $k$ without expanding all such solutions (of width $k$).

**Listing 1** Algorithm for computing treewidth. Note that lines 7–19 compute the degree of $v$ in the graph that remains after eliminating the vertices in $S$.

```
1  for k:=0 to n-1 do
2    inp:={∅};
3    for i:= 0 to n-k-2 do
4      outp = {};
5      foreach set S in inp do
6        foreach vertex v ∉ S do
7          stack := {};
8          degree := 0;
9          push v to stack;
10         while stack ≠ ∅ do
11           pop vertex u from stack;
12           foreach unvisited neighbor w of u do
13             mark w as visited;
14             if w ∈ S
15               push w to stack;
16             else
17               degree := degree+1;
18           endforeach
19         endwhile
20         if degree ≤ k
21           outp := outp ∪ {S∪{v}};
22       endforeach
23     endforeach
24     inp := outp
25   endfor
26   if inp ≠ ∅
27     report the treewidth of G is k;
28 endfor
```

To test whether the graph has treewidth at most $k$, we consider subsets $S \subseteq V$ of increasing size, such that the vertices of $S$ can be eliminated in some order without eliminating a vertex of degree $> k$. For each $k$, the algorithm starts with an input list (that initially contains just

the empty set) and then forms an output list by for each set $S$ in the input list, attempting to add every vertex $v \notin S$ to $S$, which is feasible only if the degree of $v$ in the graph that remains after eliminating the vertices in $S$ is not too large. This is tested using a depth first search. Then, the input and output lists are swapped and the process is repeated. If after $n$ iterations the output list is not empty, we can conclude that the graph has treewidth at most $k$. Otherwise, we proceed to test for treewidth $k+1$. Pseudocode for this algorithm is given in Listing 1.

We include three optimizations: first, if $C \subseteq V$ induces a clique, there is an elimination order that ends with the vertices in $C$ [4]. We can thus precompute a maximum clique $C$, and on line 7 of Lisiting 1, skip any vertices in $C$. Next, if $G$ has treewidth at most $k$ and there are at least $k+1$ vertex-disjoint paths between vertices $u$ and $v$, we may add the edge $uv$ to $G$ without increasing its treewidth [10]. Thus, we precompute for each pair of vertices $u, v$ the number of vertex-disjoint paths between them, and when testing whether the graph has treewidth at most $k$ we add edges between all vertices which have at least $k+1$ disjoint paths (note that this has diminishing returns, since in each iteration we can add fewer and fewer edges). Finally, if the graph has treewidth at least $k$, then the last $k+1$ vertices can be eliminated in any order so we can terminate execution of the algorithm earlier.

We note that our algorithm does not actually compute a tree decomposition or elimination order, but could easily be modified to do so. Currently, the algorithm stores with each (partial) solution one additional integer, which indicates which four vertices were the last to be eliminated. To reconstruct the solution, one could either store a copy of (one in every four of) the output lists on the disk, or repeatedly add the last four vertices to $C$ and rerun the algorithm to obtain the next four vertices (with each iteration taking less time than the previous, since the size of $C$ has increased).

## 3.2 Duplicate Elimination using Bloom Filters

Each set $S$ may be generated in multiple ways by adding different vertices to subsets $S' \subseteq S$; if we do not detect whether a set $S$ is already in the output list when adding it, we risk the algorithm generating $\Omega(n!)$ sets. To detect whether a set $S$ is already in the output, we use a Bloom filter [2]: Bloom filters are a classical data structure in which an array $A$ of $m$ bits can be used to encode the presence of $n$ elements by means of $k$ hash functions. To insert an element $S$, we compute $k$ independent hash functions $\{H_i | 1 \leq i \leq k\}$ each of which indicates one position in the array, $A[H_i(S)]$, which should be set to 1. If any of these bits was previously zero, then the element was not yet present in the filter, and otherwise, the probability of a false positive is approximately $(1 - e^{-kn/m})^k$.

In our implementation, we compute two 32-bit hashes $h_1(S), h_2(S)$ using Murmur3 [1], which we then combine linearly to obtain hashes $H_i(S) = h_1(S) + i \cdot h_2(S)$ (which is nearly as good as using $k$ independent hash functions [19]).

In our experiments, we have used $\frac{m}{n} \geq 24$ and $k = 17$ to obtain a low (theoretical) false positive probability of around 1 in 100.000. We note that the possibility of false positives results in a Monte Carlo algorithm (the algorithm may inadvertently decide that the treewidth is higher than it really is). Indeed, given that many millions of states are generated during the search we are guaranteed that the Bloom filter will return some false positives, however, this does not immediately lead to incorrect results: it is still quite unlikely that all of the states leading to an optimal solution are pruned, since there are often multiple feasible elimination orders.

The Bloom filter is very suitable for implementation on a GPU, since our target architecture (and indeed, most GPUs) offers a very fast atomic OR operation [21]. We note that addressing a Bloom filter concurrently may also introduce false negatives if multiple threads

attempt to insert the same element simultaneously. To avoid this, we use the initial hash value to pick one of 65.536 mutexes to synchronize access (this allows most operations to happen wait-free, and only a collision on the initial hash value causes one thread to wait for another).

### 3.3 Minor-Min-Width

Search algorithms for treewidth are often enhanced with various heuristics and pruning rules to speed up the computation. One very popular choice (used by e.g. [16, 25, 26]) is minor-min-width (MMW) [16] (also known as MMD+(min-d)) [7]). MMW is based on the observation that the minimum degree of a vertex is a lower bound on the treewidth, and that contracting edges (i.e. taking minors) does not increase the treewidth. MMW repeatedly selects a minimum degree vertex, and then contracts it with a neighbor of minimum degree, in an attempt to obtain a minor with large minimum degree (if we encounter a minimum degree that exceeds our target treewidth, we know that we can discard the current state). As a slight improvement to this heuristic, the second smallest vertex degree is also a lower bound on the treewidth [7].

Given a subset $S \subseteq G$, we would like to compute the treewidth of the graphs that remains after eliminating $S$ from $G$. The most straightforward method is to explicitly create a copy of $G$, eliminate the vertices of $S$, and then repeatedly perform the contraction as described above. However, storing e.g. an adjacency list representation of these intermediate graphs would exceed the available shared memory and size of the caches. As we would like to avoid transferring large amounts of data to and from global memory, we implemented a method to compute MMW without explicitly storing the intermediate graphs.

Our algorithm tracks the current degrees of the vertices (which, conveniently, we already have computed to determine which vertices can be eliminated). It is thus easy to select a minimum degree vertex $v$. Since we do not know what vertices it is adjacent to (in the intermediate graph), we must select a minimum degree neighbor by using a depth-first search, similarly to how we compute the vertex degrees in Listing 1. Once we have found a minimum degree neighbor $u$, we run a second dept-first search to compute the number of neighbors $u$ has in common with $v$, allowing us to update the degree of $v$. To keep track of which vertices have been contracted, we use a disjoint set data structure.

The disjoint set structure and list of vertex degrees together use only two bytes per vertex (for a graph of up to 256 vertices), thus, they fit our memory constraints whereas an adjacency matrix or adjacency list (for dense graphs, noting that the graphs in question can quickly become dense as vertices are eliminated) would readily exceed it.

## 4 Experiments

### 4.1 Instances

We selected a number of instances from the PACE 2016 dataset [12] and libtw [24].

All instances were preprocessed using the preprocessing rules of our PACE submission [8], which split the graph using *safe* separators: we first split the graph into its connected components, then split on articulation points, then on articulation pairs (making the remaining components 3-connected) and finally - if we can establish that this is safe - on articulation triplets (resulting in the 4-connected components of the graph). We then furthermore try to detect (almost) clique separators in the graph, and split on those. For a more detailed treatment of these preprocessing rules, we refer to [5].

**Table 1** Performance of the algorithm on several benchmark graphs, using global memory and a work size of 128.

| Name | $\|V\|$ | tw | Time (sec.) | | Exp |
|---|---|---|---|---|---|
| | | | GPU | CPU | |
| 1e0b_graph | 55 | 24 | 779 | - | 1730 $\times 10^6$ |
| 1fjl_graph* | 57 | 26 | 1730 | - | 3680 $\times 10^6$ |
| 1igd_graph | 59 | 25 | 107 | 5120 | 261 $\times 10^6$ |
| 1ubq* | 47 | ~~11~~ | 1130 | - | 2300 $\times 10^6$ |
| 8x6_torusGrid* | 48 | ~~7~~ | 1110 | - | 2100 $\times 10^6$ |
| BN_98 | 47 | 21 | 689 | - | 1590 $\times 10^6$ |
| contiki_dhcpc_handle_dhcp* | 39 | 6 | 1490 | - | 2930 $\times 10^6$ |
| DoubleStarSnark | 30 | 6 | 34,5 | 873 | 87,6 $\times 10^6$ |
| KneserGraph_8_3* | 56 | ~~24~~ | 1710 | - | 4130 $\times 10^6$ |
| myciel5* | 47 | 19 | 2000 | 70.600 | 4000 $\times 10^6$ |
| NonisotropicUnitaryPolarGraph_3_3 | 63 | 53 | 1,16 | 60,4 | 1,56 $\times 10^6$ |
| queen8_8 | 64 | 45 | 26,3 | 2040 | 57,9 $\times 10^6$ |
| RandomBarabasiAlbert_100_2* | 41 | 12 | 1610 | - | 3280 $\times 10^6$ |
| RandomBoundedToleranceGraph_60 | 59 | 30 | 0,274 | 0,635 | 0,0560 $\times 10^6$ |
| SylvesterGraph | 36 | 15 | 248 | - | 632 $\times 10^6$ |
| te* | 62 | ~~7~~ | 1170 | - | 2160 $\times 10^6$ |

## 4.2 General Benchmark

We first present an experimental evaluation of our algorithm (without using MMW) on a set of benchmark graphs. Table 1 shows the number of vertices, computed treewidth, time taken (in seconds) on the GPU and the number of sets $S$ explored. Note that the time does not include the time taken for preprocessing, and that the vertex count is that of the preprocessed graph (and thus, the original graph may have been larger).

The size of the input and output lists were limited by the memory available on our GPU. With the current configuration (limited to graphs of at most 64 vertices - though the code is written to be flexible and can easily be changed to support up to 256 vertices), these lists could hold at most 180 million states (i.e., subsets $S \subseteq V$ that have a feasible partial elimination order) each. If at any iteration this number was exceeded, the excess states were discarded. The algorithm was allowed to continue execution for the current treewidth $k$, but was terminated when trying the next higher treewidth (since we might have discarded a state that would have lead to a solution with treewidth $k$, the answer would no longer be exact). The states were the capacity of the lists was exceed are marked with *, if the algorithm was terminated then the treewidth is stricken through (and represents the candidate value for treewidth at which the algorithm was terminated, and *not* the treewidth of the graph, which is likely higher).

For instance, for graph 1ubq the capacity of the lists was first exceeded at treewidth 10, and the algorithm was terminated at treewidth 11 (and thus the actual treewidth is at least 10, but likely higher). For graph myciel5, the capacity of the lists was first exceeded at treewidth 19, but still (despite discarding some states) a solution of treewidth 19 was nevertheless found (which we thus know is the exact treewidth).

For several graphs (those where the GPU version of the algorithm took at most 5 minutes), we also benchmarked a sequential version of the same algorithm on the CPU. In some cases,

**Table 2** Running time (sec.) for various work group sizes ($W$), using shared (S) or global (G) memory. Each cell lists the average result of 4 test runs, where the complete set of runs was executed in a randomized order.

| Name | $\|V\|$ | tw | Time (sec.) | | | |
|---|---|---|---|---|---|---|
| | | | $W = 32$ | $W = 64$ | $W = 128$ | $W = 256$ |
| 1igd_graph (G) | 59 | 25 | 109 | 107 | 107 | 107 |
| 1igd_graph (S) | 59 | 25 | 94,8 | 95,6 | 98,2 | 103 |
| 1ku3_graph (G) | 60 | 22 | 238 | 235 | 235 | 235 |
| 1ku3_graph (S) | 60 | 22 | 214 | 217 | 222 | 230 |
| queen8_8 (G) | 64 | 45 | 29,5 | 26,6 | 26,3 | 26,0 |
| queen8_8 (S) | 64 | 45 | 25,1 | 24,1 | 24,5 | 25,0 |

the algorithm achieves a very large speedup compared to the CPU version (up to 77×, in the case of queen8_8). Additionally, for myciel5, we also ran the CPU-based algorithm, which took more than 19 hours to finish. The GPU version only took 34 minutes.

The GPU algorithm can process a large amount of states in a very short time. For example, for the graph 1fjl, 3680 million states were explored in just 1730 seconds, i.e., over 2 million states were processed each second (and for each state, a $\Theta(|V||E|)$-time algorithm is executed). The highest throughput (2.5 million states/sec.) is achieved on SylvesterGraph, but this graph has relatively few vertices.

We caution the reader that the graph names are somewhat ambiguous. For instance, the queen7_7 instance is from libtw and has treewidth 35. The 2016 PACE instances include a graph called dimacs_queen7_7 which only has treewidth 28. The instances used in our evaluation are available from our GitHub repository [9].

## 4.3 Work Size and Global v.s. Shared Memory

In this section, we study the effect of work size and whether shared or global memory is used on the running time of our implementation.

Recall that shared memory is a small amount (in our case, 96KiB) of memory that is physically close to each Streaming Multiprocessor, and is therefore in principle faster than the (much larger, off-chip) global memory. We would therefore expect that our implementation is faster when used with shared memory.

Each SM contains 128 CUDA cores, and thus 4 warps of 32 threads each can be executed simultaneously on each SM. The work size (which should be a multiple of 32), represents the number of threads we assign to each SM. If we set the work size larger than 128, more threads than can physically be executed at once are assigned to one SM. The SM can then switch between executing different warps, for instance to hide latency of memory accesses. If the work size is smaller than 128, a number of CUDA cores will be unutilized.

In Table 2, we present some experiments that show running times on several graphs, depending on whether shared memory or global memory is used, for several sizes of work group (which is the number of threads allocated to a single SM).

There is not much difference between running the program using shared or global memory. In most instances, the shared memory version is slightly faster. Surprisingly, it also appears that the work size used does not affect the running time significantly. This suggests that our program is limited by the throughput of memory, rather than being computationally-bound.

**Table 3** The effect of using the Minor-Min-Width Heuristic. Time is in seconds. Global memory, work size 128.

| Name | $\|V\|$ | tw | With MMW | | Without MMW | |
|---|---|---|---|---|---|---|
| | | | Time | Exp | Time | Exp |
| 1e0b_graph | 55 | 24 | 2750 | 1660 $\times 10^6$ | 779 | 1730 $\times 10^6$ |
| 1fjl_graph* | 57 | 26 | timeout | 3260 $\times 10^6$ | 1730 | 3680 $\times 10^6$ |
| 1igd_graph | 59 | 25 | 471 | 235 $\times 10^6$ | 107 | 261 $\times 10^6$ |
| 1ubq* | 47 | ~~11~~ | 2010 | 1500 $\times 10^6$ | 1130 | 2300 $\times 10^6$ |
| 8x6_torusGrid* | 48 | ~~7~~ | 1350 | 1300 $\times 10^6$ | 1110 | 2100 $\times 10^6$ |
| BN_98 | 47 | 21 | 1480 | 1440 $\times 10^6$ | 689 | 1590 $\times 10^6$ |
| contiki_dhcpc_handle_dhcp* | 39 | 6 | 2670 | 2900 $\times 10^6$ | 1490 | 2930 $\times 10^6$ |
| DoubleStarSnark | 30 | 6 | 38,3 | 76,0 $\times 10^6$ | 34,5 | 87,6 $\times 10^6$ |
| KneserGraph_8_3* | 56 | ~~24~~ | 1330 | 1220 $\times 10^6$ | 1730 | 4130 $\times 10^6$ |
| myciel5* | 47 | 19 | 2550 | 3200 $\times 10^6$ | 2000 | 4000 $\times 10^6$ |
| NonisotropicUnitaryPolarGraph_3_3 | 63 | 53 | 3,36 | 1,30 $\times 10^6$ | 1,16 | 1,56 $\times 10^6$ |
| queen8_8 | 64 | 45 | 83,5 | 51,1 $\times 10^6$ | 26,3 | 57,9 $\times 10^6$ |
| RandomBarabasiAlbert_100_2* | 41 | 12 | 2390 | 2840 $\times 10^6$ | 1610 | 3280 $\times 10^6$ |
| RandomBoundedToleranceGraph_60 | 59 | 30 | 0,630 | 0,0478 $\times 10^6$ | 0,274 | 0,0560 $\times 10^6$ |
| SylvesterGraph | 36 | 15 | 274 | 503 $\times 10^6$ | 248 | 632 $\times 10^6$ |
| te* | 62 | ~~10~~ | 2260 | 1690 $\times 10^6$ | 1170 | 2160 $\times 10^6$ |

## 4.4 Minor-Min-Width

In Table 3, we list results obtained when using Minor-Min-Width to prune states.

The computational expense of using MMW is comparable to that of the initial computation (for determining the degree of vertices): the algorithm does a linear search for a minimum degree vertex (using the precomputed degree values), and then does a graph traversal (using BFS) to find a minimum degree neighbour (recall that we do not store the intermediate graph, and use only a single copy of the original graph). Once such a neighbour is found, the contraction is performed (by updating the disjoint set data structure) and another graph traversal is required (to compute the number of common neighbours, and thus update the degree of the vertex).

The lower bound given by MMW does not appear to be very strong, at least for the graphs considered in our experiment: the reduction in number of states expanded is not very large (for instance, from 1730 million states to 1660 million for `1e0b`, or from 1590 million to 1480 million for `BN_98`). The largest reductions are visible for graphs on which we run out of memory (for instance, from 4130 million to 1330 million for `KneserGraph_8_3`), but this is likely because the search is terminated before we reach the actual treewidth (so we avoid the part of our search where using a heuristic is least effective) and there are no graphs on which we previously ran out of memory for which MMW allows us to determine the treewidth (the biggest improvement is that we are able to determine that `te` has treewidth at least 10, up from treewidth at least 7).

Consistent with the relatively low reduction in the number of states expanded, we see the computation using MMW typically takes around $2-3$ times longer. On the graphs considered here, the reduction in search space offered by MMW does not offset the additional cost of computing it.

Again, the GPU version is significantly faster than executing the same algorithm on the CPU: we observed a 55× speedup for `queen8_8`. Still, given what we observed in Section

4.3, it is not clear whether our approach of not storing the intermediate graphs explicitly is indeed the best approach. Our main motivation for taking this approach was to be able to store the required data structures entirely in shared memory, but our experiments indicate that for MMW, using global memory gives better performance than using shared memory. However, the relatively good performance of global memory might be (partially) due to caching and the small amount of data transferred, so it is an interesting open question to determine whether the additional memory costs of using more involved data structures is compensated by the potential speedup.

## 4.5 Loop Unnesting

Finally, we experimented with another technique, which aims to increase parallelism (and thus speedup) by limiting branch divergence. However, as the results were discouraging, we limit ourselves to a brief discussion.

The algorithm of Listing 1 consists of a loop (lines 5–22) over the (not yet eliminated) vertices, inside of which is a depth-first search (which computes the degree of the vertex, to determine whether it can be eliminated). The depth-first search in turn consists of a loop which runs until the stack becomes empty (lines 10–19) inside of which is a final loop over the neighbours of the current vertex (lines 12–18). This leads to two sources of branch divergence:

- First, if the graph is irregular, all threads in a warp have to wait for the thread that is processing the highest degree vertex, even if they only have low-degree vertices.
- Second, all threads in a warp have to wait for the longest of the BFS searches to finish before they can start processing the next vertex.

To alleviate this, we proposed a technique which we call *loop unnesting*: rather than have 3 nested loops, we have only one loop, which simulates a state machine with 3 states: (1) processing the adjacency list of a vertex, (2) having finished processing of an adjacency list and being ready to pop a new vertex off the queue, or (3) having finished a BFS, and being ready to begin computing the degree of a new vertex.

We considered a slightly more general version of this idea: in an $(x, y)$-unnesting of our program, after every $x$ iterations of the inner loop (exploring neighbours of the current vertex) one iteration of the middle loop is executed (if exploring the adjacency list is finished, get a new vertex from the queue), and for every $y$ iterations of the middle loop, one iteration of the outer loop is executed (begin processing an entirely new vertex). Thus, a $(1, 1)$-unrolling corresponds to the state machine simulation described above, and an $(\infty, \infty)$-unrolling corresponds to the original program.

Picking the right values for $x, y$ means finding the right trade-off between checking frequently enough whether a thread is ready to start working on another vertex, and the cost of performing those checks. What we observed was surprising: while $(1, 1)$, $(3, 2)$ and $(1, \infty)$-unrollings gave reasonable results, the best results were obtained with $(\infty, \infty)$-unrollings (i.e. the original, unmodified algorithm) and the performance of $(\infty, 1)$-unrollings was abysmal.

We believe that a possible explanation may be that loop unnesting does work to some extent, but not unnesting the loops has the advantage that all BFS searches running simultaneously start from the same initial vertex, and (up to differences caused by different sets $S$ being used) will access largely the same values from the adjacency lists at the same time, which may increase the efficiency of read operations. On the other hand, $(\infty, 1)$-unnesting can not take advantage of either phenomenon: different initial vertices may be processed at any given time (so there is little consistency in memory accesses) and the inner loop

is not unnested at all so there is no potential to gain speedup there either. Perhaps for larger graphs, where the difference in length of adjacency lists may be more pronounced, or the amount of time a BFS takes varies more strongly with the initial vertes and $S$, loop unnesting does provide speed up, but for the graphs considered here it does not appear to be a beneficial choice.

## 5 Conclusions

We have presented an algorithm that computes treewidth on the GPU, achieving a very large speedup over running the same algorithm on the CPU. Our algorithm is based on the classical $O^*(2^n)$-time dynamic programming algorithm [4] and our results represent (promising) first steps in speeding up dynamic programming for treewidth on the GPU. The current best known practical algorithm for computing treewidth is the algorithm due to Tamaki [22]. This algorithm is much more complicated, and porting it to the GPU would be a formidable challenge but could offer an extremely efficient implementation for computing treewidth.

Given the large speedup achieved, we are no longer mainly limited by computation time. Instead, our ability to solve larger instances is hampered by the memory required to store the very large lists of partial solutions. Using minor-min-width did not prove effective in reducing the number of states considerably, so it would be interesting to see how other heuristics and pruning rules (such as simplicial vertex detection) could be implemented on the GPU.

GPUs are traditionally used to solve easy (e.g. linear time) problems on very large inputs (such as the millions of pixels rendered on a screen, or exploring a graph with millions of nodes), but clearly, the speedup offered by inexpensive GPUs would also be very welcome in solving hard (NP-complete) problems on small instances. Exploring how techniques from FPT and exact algorithms can be used on the GPU raises many interesting problems - not only practical ones, but also theoretical: how should we model complex devices such as GPUs, with their many types of memory and branch divergence issues?

**Acknowledgements.** We thank Jacco Bikker for discussions on the architecture of GPUs, and Gerard Tel for discussions on hash functions.

**Source Code and Instances.** We have made our source code, as well as the graphs used for the experiments, available on GitHub [9].

## References

1 Austin Appleby. SMHasher. Accessed 2017-04-12. URL: `https://github.com/aappleby/smhasher`.
2 Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
3 Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996.
4 Hans L. Bodlaender, Fedor V. Fomin, Arie M. C. A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos. On exact algorithms for treewidth. *ACM Trans. Algorithms*, 9(1):12:1–12:23, 2012.
5 Hans L. Bodlaender and Arie M.C.A. Koster. Safe separators for treewidth. *Discrete Mathematics*, 306(3):337–350, 2006.
6 Hans L. Bodlaender and Arie M.C.A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.

7 Hans L. Bodlaender and Arie M.C.A. Koster. Treewidth computations II. Lower bounds. *Information and Computation*, 209(7):1103–1119, 2011.

8 Hans L. Bodlaender and T. C. van der Zanden. BZTreewidth. Accessed 2017-04-11. URL: `https://github.com/TomvdZanden/BZTreewidth`.

9 Hans L. Bodlaender and T. C. van der Zanden. GPGPU treewidth. Accessed 2017-04-21. URL: `https://github.com/TomvdZanden/GPGPU-Treewidth`.

10 François Clautiaux, Jacques Carlier, Aziz Moukrim, and Stéphane Nègre. New lower and upper bounds for graph treewidth. In Klaus Jansen, Marian Margraf, Monaldo Mastrolilli, and José D. P. Rolim, editors, *Experimental and Efficient Algorithms: Second International Workshop, WEA 2003*, pages 70–80. Springer, 2003.

11 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 1st edition, 2015.

12 Holger Dell, Thore Husfeldt, Bart M. P. Jansen, Petteri Kaski, Christian Komusiewicz, and Frances A. Rosamond. The parameterized algorithms and computational experiments challenge (PACE). Accessed 2017-04-05. URL: `https://pacechallenge.wordpress.com/pace-2016/track-a-treewidth/`.

13 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 fuer Informatik, 2016. `doi:10.4230/LIPIcs.IPEC.2016.30`.

14 P. Alex Dow. *Search Algorithms for Exact Treewidth*. PhD thesis, 2010.

15 P. Alex Dow and Richard E. Korf. Best-first search for treewidth. In *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2*, AAAI'07, pages 1146–1151. AAAI Press, 2007.

16 Vibhav Gogate and Rina Dechter. A complete anytime algorithm for treewidth. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, pages 201–208. AUAI Press, 2004.

17 Alexander Hein and Arie M. C. A. Koster. An experimental evaluation of treewidth at most four reductions. In Panos M. Pardalos and Steffen Rebennack, editors, *Proceedings of the 10th International Symposium on Experimental and Efficient Algorithms, SEA 2011*, volume 6630 of *LNCS*, pages 218–229. Springer, 2011.

18 M. Held and R. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10:196–210, 1962.

19 Adam Kirsch and Michael Mitzenmacher. Less hashing, same performance: Building a better bloom filter. In Yossi Azar and Thomas Erlebach, editors, *Algorithms – ESA 2006: 14th Annual European Symposium*, pages 456–467. Springer, 2006.

20 NVIDIA. NVIDIA GeForce GTX 1080 Whitepaper. Accessed 2017-04-10. URL: `http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_1080_Whitepaper_FINAL.pdf`.

21 NVIDIA. NVIDIA's Next Generation CUDA Compute Architecture: FERMI. Accessed 2017-04-12. URL: `http://www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper.pdf`.

22 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 68:1–68:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. `doi:10.4230/LIPIcs.ESA.2017.68`.

**23** Tom C. van der Zanden and Hans L. Bodlaender. Computing Treewidth on the GPU. Preprint, 2017. `arXiv:arXiv:1709.09990`.

**24** Thomas C. van Dijk, Jan-Pieter van den Heuvel, and Wouter Slob. Computing treewidth with libtw, 2006. Accessed 2017-06-16. URL: `http://www.treewidth.com/treewidth`.

**25** Y. Yuan. A fast parallel branch and bound algorithm for treewidth. In *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, pages 472–479, 2011.

**26** Rong Zhou and Eric A. Hansen. Combining breadth-first and depth-first strategies in searching for treewidth. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence*, IJCAI'09, pages 640–645. Morgan Kaufmann Publishers Inc., 2009.

# The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration*

**Holger Dell[1], Christian Komusiewicz[2], Nimrod Talmon[3], and Mathias Weller[4]**

1 **Saarland University and Cluster of Excellence (MMCI), Saarbrücken, Germany**
hdell@mmci.uni-saarland.de
2 **Friedrich-Schiller-University Jena, Germany**
christian.komusiewicz@uni-jena.de
3 **Weizmann Institute of Science, Rehovot, Israel**
nimrod.talmon@weizmann.ac.il
4 **Laboratory of Informatics, Robotics, and Microelectronics of Montpellier (LIRMM), France**
weller@lirmm.fr

**Abstract**

In this article, the Program Committee of the Second Parameterized Algorithms and Computational Experiments challenge (PACE 2017) reports on the second iteration of the PACE challenge. Track A featured the Treewidth problem and Track B the Minimum Fill-In problem. Over 44 participants on 17 teams from 11 countries submitted their implementations to the competition.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, K.7.2 Organizations

**Keywords and phrases** treewidth, minimum fill-in, contest, implementation challenge, FPT

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2017.30

## 1 Introduction

The Parameterized Algorithms and Computational Experiments Challenge (PACE) was conceived in Fall 2015 to help deepen the relationship between parameterized algorithmics and practice. In particular, it aims to:

1. Bridge between algorithm design and analysis theory and algorithm engineering practice.
2. Inspire new theoretical developments.
3. Investigate the competitiveness of analytical and design frameworks developed in the communities.
4. Produce universally accessible libraries of implementations and repositories of benchmark instances.
5. Encourage dissemination of the findings in scientific papers.

* The PACE challenge was supported by Networks, an NWO Gravitation project of the University of Amsterdam, Eindhoven University of Technology, Leiden University, and the Center for Mathematics and Computer Science (CWI).




The first iteration of PACE was held in 2016 [10] and has met many of its aims. For instance, PACE is mentioned as inspiration in numerous papers [1, 2, 12, 13, 17, 19, 23, 31, 34, 35]. Here, we report on the second iteration of PACE.

The PACE 2017 challenge was announced on December 1, 2016 with two tracks: Track A (Treewidth) and Track B (Minimum Fill-In). The final version of the submissions was due on May 1, 2017. We informed the participants of the result on June 1, 2017, and announced them to the public on September 6, 2017, during the award ceremony at the International Symposium on Parameterized and Exact Computation (IPEC 2017) in Vienna.

## 2 Competition track A: Treewidth

The objective of this track is to compute a minimum tree-decomposition of a given graph:

**Input:** An undirected graph.

**Output:** A minimum-width tree-decomposition of the graph.

The *treewidth*, which is the width of a minimum-width tree-decomposition, is one of the most important graph parameters in parameterized algorithms. Treewidth implementations are used in various contexts, for example in register allocation (e.g. [33]), shortest path computation (e.g. [9]), probabilistic inference (e.g. [21]), graph theory (e.g. [22]), and low-dimensional topology (e.g. [7]).

Last year's PACE challenge achieved the first systematic comparison between different implementations that compute minimum tree decompositions [10]. Since treewidth is such a central problem, we wanted to see more improvement. We required all submission to be single-threaded (with the exception of wrapper threads forced by limitations of the programming language). The PACE 2017 treewidth track consisted of the following two independent competitions.

**Exact competition:** Compute a tree-decomposition of minimum width. You have 30 minutes per instance. Win by solving more instances than the other participants.

**Heuristic competition:** Compute some tree-decomposition. You have 30 minutes per instance. Win by printing solutions of smaller width than the other participants.

All instances used in the competition of PACE 2017 were derived from the following sources: transit and road networks that were submitted to PACE 2016, instances from the SAT competition, instances from the UAI 2014 inference competition, and some instances from `treedecomposition.com`. In contrast to the treewidth competition of PACE 2016, no randomly generated instances were used. We cleaned up all instances as follows. We deleted vertices of degree one. For vertices $v$ with exactly two non-adjacent neighbors $u$ and $w$, we deleted $v$ and added the edge $uw$. Finally, we kept only the largest connected component and deleted all other connected components. The base pool and all derived competition instances can be downloaded from `github.com/PACE-challenge/Treewidth`.

For the heuristic competition, we selected 200 instances `he001.gr`, ..., `he200.gr` from the base pool, ordered by file size and anonymized. For the exact competition, we systematically derived 200 instances `ex001.gr`, ..., `ex200.gr` from the base pool, roughly ordered by their believed hardness. We made the odd-numbered instances public when PACE 2017 was announced, and we kept the even-numbered instances secret until the submission deadline. The final competition and ranking was based only on the secret instances. Figure 1 shows some statistics for the two secret instance sets.

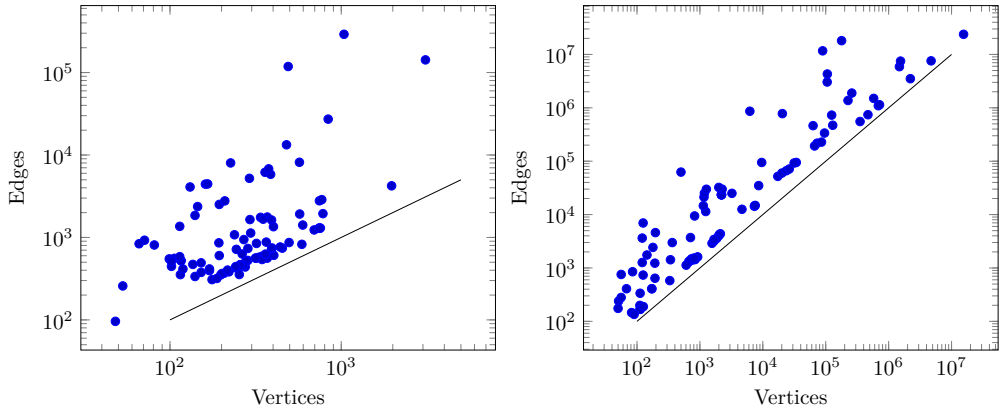
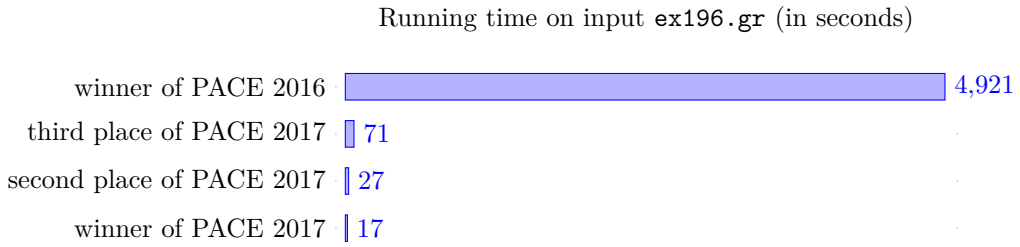



**Figure 1** The PACE 2017 secret instance sets consist of the even-numbered instances `ex002.gr`, ..., `ex200.gr` (*left*) and `he002.gr`, ..., `he200.gr` (*right*). We plot the number $n$ of vertices and the number $m$ of edges. The exact competition contained instances whose treewidth was between 6 and 540, with a median of 11 and a mean of 31; the median best treewidth found for the heuristic competition was 93, and the mean was 13,038.

## 2.1 Exact competition: Compute an optimal tree decomposition

Three implementations were submitted to the exact treewidth competition of PACE 2017. Let us compare the three submissions to PACE 2017 and the winning submission from last year on a particular instance `ex196.gr`, which has 242 vertices, 443 edges, and treewidth 11:




All three PACE 2017 submissions are two orders of magnitude faster than last year!

The instance `ex196.gr` was generated as follows: We started with the MMAP instance `Promedas_30` from the UAI 2014 inference competition. Since many probabilistic inference algorithms first compute a good tree decomposition of the input, instances like `Promedas_30` are great to include in the base pool for the treewidth competition. Exact treewidth solvers don't scale enough yet to solve the entire instance, which has 1155 vertices and 2290 edges. It took the winning submission for the PACE 2017 heuristic treewidth challenge about 7 days to compute an upper bound of 51 on the treewidth of `Promedas_30`. Thus for the exact treewidth competition, we had to make the instance more tractable. To do so; a random center vertex was chosen and the graph induced by all vertices at distance at most $r$ from this center was kept. The instance generated this way was then cleaned again by taking care of vertices of degree one and two. The radius $r$ was increased as much as possible so that last year's winning submission can solve the instance in just under two hours. The setting $r = 5$ gives rise to the graph `ex196.gr`.

All PACE 2017 instances for the exact treewidth competition were derived in a similar fashion from the base pool of instances and it took several CPU months to choose the radii appropriately. While the instances were chosen to be hard, we were surprised by the fantastic

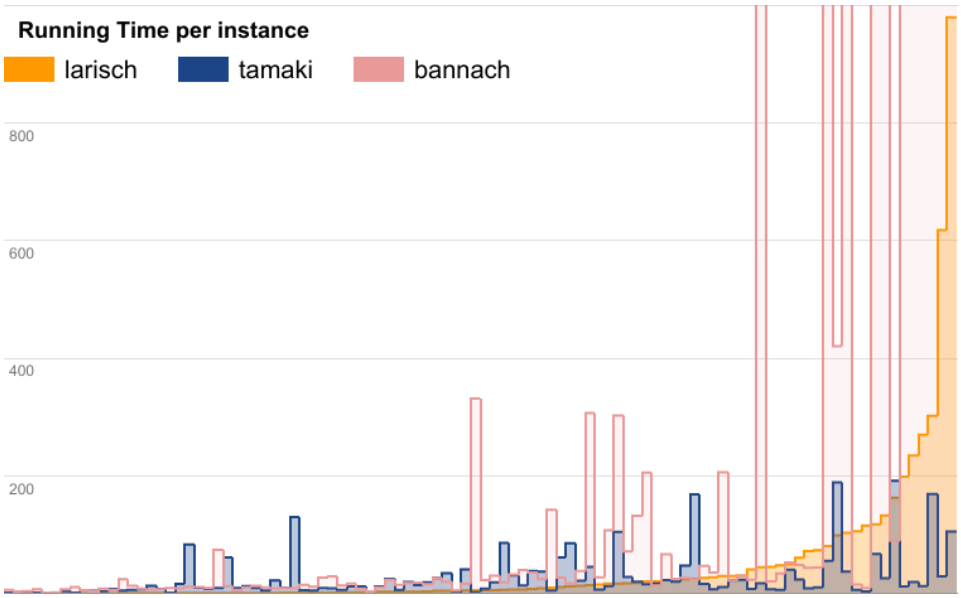



**Figure 2** For each competition instance `ex002.gr`, ..., `ex200.gr` (on the horizontal axis), we plot the running time in seconds (on the vertical axis). The instances are sorted by increasing running time for the winning solver of Larisch and Salfelder.

improvements seen in treewidth implementations when compared to last year: most of the instances we generated can now be solved in a few seconds. The mean running time on the secret instances `ex002.gr`, ..., `ex200.gr` was only 12 seconds, and the slowest was an instance solved in 162 seconds. As a consequence, the best two submissions solved *all* PACE 2017 competition instances within the allotted time of 30 minutes. In the announcement of the PACE 2017 competition, we defined the following tie-breaker rule for this situation: The winning solver is the one which is faster on most instances. The winning submission solved about 67% of the instances faster than the one we ranked second. Here is the final ranking:

- **1st place:** Lukas Larisch (King-Abdullah University of Science and Engineering) and Felix Salfelder (University of Leeds) solved all 100 instances
  `github.com/freetdi/p17`
- **2nd place:** Hiromu Ohtsuka and Hisao Tamaki (Meiji University) solved all 100 instances
  `github.com/TCS-Meiji/PACE2017-TrackA`
- **3rd place:** Max Bannach (University of Lübeck), Sebastian Berndt (University of Lübeck), and Thorsten Ehlers (University of Kiel) solved 89 instances
  `github.com/maxbannach/Jdrasil`

While the solver by Larisch and Salfelder is fastest on most instances, its running time has a large variance and there are some instances where its running time almost reaches the allowed 30 minutes. The cumulative total time to solve all 100 instances was 4478 seconds for Larisch and Salfelder and 2747 seconds for Ohtsuka and Tamaki.

## 2.2 Heuristic competition: Compute some tree decomposition

There were six participating teams in the heuristic challenge. The selected instances were chosen from the largest few instances from each category in the base pool.

For the ranking, each instance was considered to be a "voter" where smaller width leads to a better rank for the submission; outputting no solution was ranked the same as outputting a trivial solution. To avoid misguided micro-optimization incentives, a solution was considered

| Competitor | Approx. Ratio Avg. | Max. | Diff. to best Sol.[%] $= 0$ | $\leq 1$ | $\leq 2$ | $< \infty$ |
|---|---|---|---|---|---|---|
| tamaki_tw-heuristic | 1.30 | 9.61 | **51** | **52** | 52 | 92 |
| strasser_flow_cutter_pace17 | **1.08** | **1.55** | 42 | 50 | **56** | **100** |
| abseher_htd_gr2td_exhaustive.sh | 1.11 | 2.37 | 25 | 37 | 46 | 95 |
| bannach_tw-heuristic | 1.19 | 3.90 | 21 | 29 | 32 | 93 |
| terrioux1_minfill_mrs | 1.30 | 2.43 | 9 | 13 | 17 | 81 |
| terrioux2_minfillbg_mrs | 4.68 | 139.70 | 10 | 14 | 18 | 71 |
| larisch_tw-heuristic | 1.56 | 3.74 | 6 | 8 | 13 | 54 |

**Figure 3** This table lists statistics on the solution quality for each competitor, across the even-numbered instances `he002.gr`, ..., `he200.gr`. The *approximation ratio* that a solver achieves on an instance is the fraction of the treewidth upper bound achieved by the solver divided by the best width achieved by any solver. The *average approximation ratio* takes the average across all 100 instances, while the *maximum approximation ratio* is the worst approximation ratio achieved across the instances. The four columns on the right display *additive errors*. For example, Tamaki et al. achieve the best treewidth upper bound on 51 of the instances, achieve at most the best plus one on 52 instances, achieve the best plus two on 52 instances, and obtain a non-trivial solution (width $\leq n - 5$) on 92 instances. The best result in each column is highlighted in bold.

"non-trivial" if its width is at most the number of vertices of the input graph minus 5. The preference lists for each instance are then combined using the Schulze method. Since there is a Concordet winner, most sensible voting schemes would have given the same result. The final ranking for the PACE 2017 heuristic treewidth challenge is this:

- **1st place:** Keitaro Makii, Hiromu Ohtsuka, Takuto Sato, Hisao Tamaki (Meiji University)
  `github.com/TCS-Meiji/PACE2017-TrackA`
- **2nd place:** Ben Strasser (Karlsruhe Institute of Technology)
  `github.com/kit-algo/flow-cutter-pace17`
- **3rd place:** Michael Abseher, Nysret Musliu, Stefan Woltran (TU Wien, Institute of Information Systems)
  `github.com/mabseher/htd`
- **4th place:** Max Bannach (University of Lübeck), Sebastian Berndt (University of Lübeck), and Thorsten Ehlers (University of Kiel)
  `github.com/maxbannach/Jdrasil`
- **5th place:** Philippe Jégou, Hanan Kanso, Cyril Terrioux (Aix-Marseille Université, LSIS)
  `github.com/td-mrs/minfill_mrs.git`, `github.com/td-mrs/minfillbg_mrs.git`
- **6th place:** Lukas Larisch (King-Abdullah University of Science and Engineering) and Felix Salfelder (University of Leeds)
  `github.com/freetdi/p17`

The solver by Tamaki et al. dominates the one of Strasser on 51.65% of the instances. This means that, after subtracting the 9 instances on which they produce the same width, just a bit more than half of the remaining instances (namely, 47) are solved better by the first solver, and a bit less than half (namely, 44) are solved better by the second (see Figure 4). Strasser's submission dominates the one of Abseher et al. on 52.05% of the non-tied instances. The top 3 submissions dominate the submission of Bannach et al. with 62%, and the top 4 submissions each dominate the remaining two submission by 90% or more.

The solver of Tamaki et al. is particularly good at small instances, where it is able to avoid off-by-one errors well. In fact, as can be seen from Figure 3, the Schulze method ranking

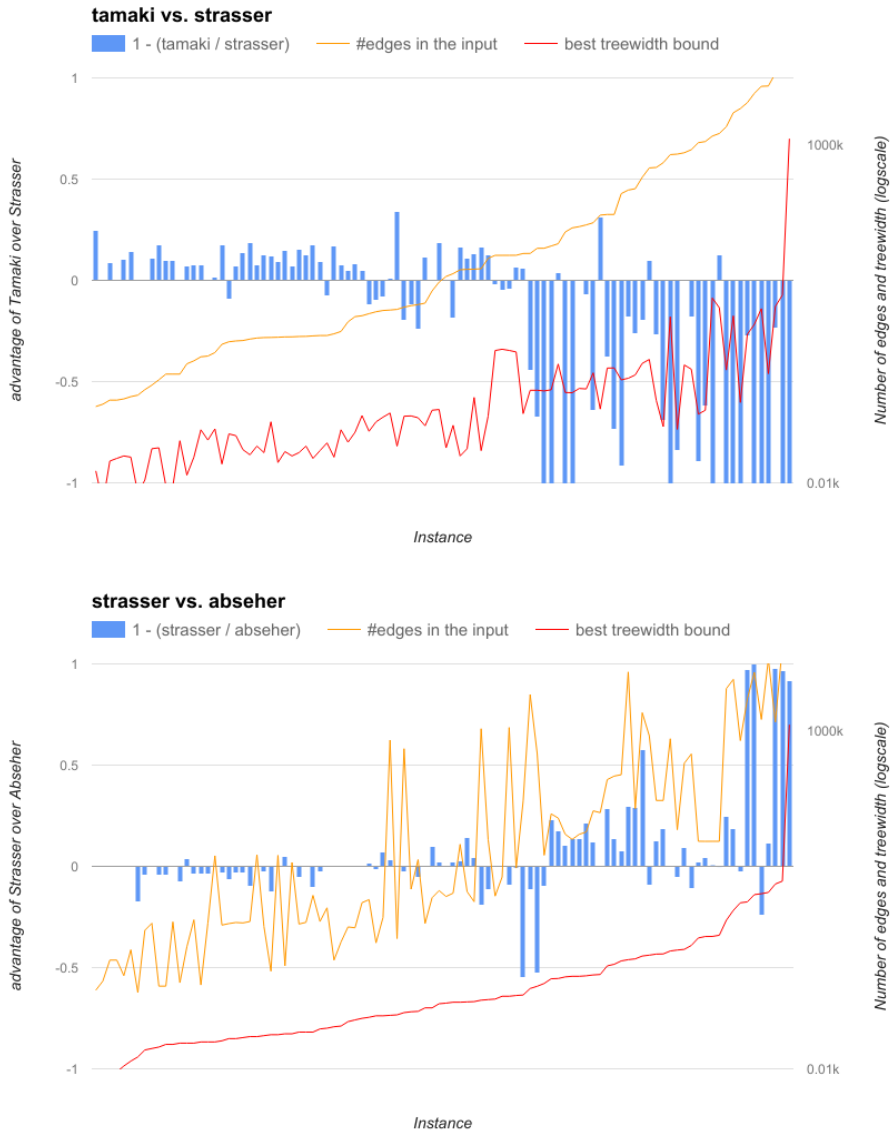



**Figure 4** Comparison between Tamaki et al. and Strasser (*top*) and Strasser and Abseher et al. (*bottom*). For each even-numbered instance `he002.gr`, ..., `he200.gr` (on the horizontal axis), we plot its number of edges in logscale (*yellow line*) and the best upper bound on the treewidth achieved during the competition (*red line*), also in logscale. It can be observed that the treewidth upper bound grows with the number of edges in our instance set. The *blue bars* depict the advantage for the first solver over the second (negative bars represent a disadvantage). More precisely, the advantage is $\varepsilon$ if $T = (1 - \varepsilon)S$ holds, where $T$ is the width produced by the first and $S$ is the width produced by the second solver. Interestingly, the solver of Tamaki et al. appears to do consistently well on the smaller instances, many of which were derived from the UAI 2014 inference competition and from `treewidth.com`, while Strasser's submission performs consistently better on larger instance.

system has, in this case, punished off-by-one errors quite a bit. On very large instances, a more meaningful measure of quality would be the average or maximum approximation ratio achieved, where the solver of Strasser et al. excels.

## 3 Competition track B: Minimum fill-in

The objective of this track was to solve the NP-hard Minimum Fill-In problem, defined as follows.

**Input:** An undirected simple graph $G = (V, E)$.

**Output:** A minimum-size set of edges such that adding these edges to $G$ results in a simple chordal graph, that is, a graph in which every induced cycle has length exactly three.

Minimum Fill-In has applications in optimizing Gaussian elimination on sparse symmetric matrices and in computational phylogenetics [6, 18] and it is notorious for being one of the open problems in the first edition of Garey and Johnson's monograph on NP-completeness [16]. Yannakakis later proved that the problem is NP-complete [36]. Minimum Fill-In is fixed-parameter tractable when parameterized by the number $k$ of edges that need to be added to make the graph chordal [8, 4, 15, 20]. The fastest parameterized algorithm for this parameter has a subexponential running time of $2^{o(k)} + O(k^2nm)$ [15]. The problem also admits a polynomial problem kernel with respect to this parameter [20, 26], the current best bound on the kernel size is $O(k^2)$ [26]. The best exact algorithms for solving Minimum Fill-In are based on the technique of dynamic programming over so-called potential maximal cliques [5, 14, 15].

### Test Instances

The test instances were derived from several sources. In light of the application of optimizing Gaussian elimination, several instances were taken as-is from the Matrix Market [24] and the Network Repository [27]. In context of phylogenetic tree reconstruction, we used 39 alignments of different mammalian DNA markers obtained from the OrthoMaM database [30, 11]. Each alignment is then interpreted as a character-state matrix. Buneman's theorem [6] states that each such character-state matrix allows reconstructing a perfect phylogeny if and only if the "character-state intersection graph[1]" has a chordal supergraph with some specific properties. Thus, we wrote a program to convert the character-state matrices to character-state intersection graphs [25] and used these graphs for the competition. To allow for a smooth transition from easy to hard instances, we sampled connected subgraphs of the real-world instances whose density did not deviate too much from the density of the original instance. An overview of the number of vertices and edges in the set of hidden instances is shown in Figure 5. The 200 resulting graphs were divided into 100 *public* instances (that the participants could test their implementations on), and 100 *hidden* instances (used to determine the ranking of the submissions).

### Rules

The PACE 2017 Minimum Fill-In track was a competition to solve as many test instances of Minimum Fill-In within the allowed time limit of 30 minutes per instance. The winners were selected based on the number of solved hidden instances. Concentrating on fixed-parameter tractability, we disallowed the use of SAT solvers, ILP solvers, or similar general solvers for NP-hard problems. Further, as the competition should be about exact solvers, submissions producing at least one incorrect or suboptimal solution were ranked below any other solvers producing only optimal solutions.

[1] For each species $s$, the character-state intersection graph contains a clique of the vertices $(c, i)$ such that character $c$ has state $i$ in species $s$.

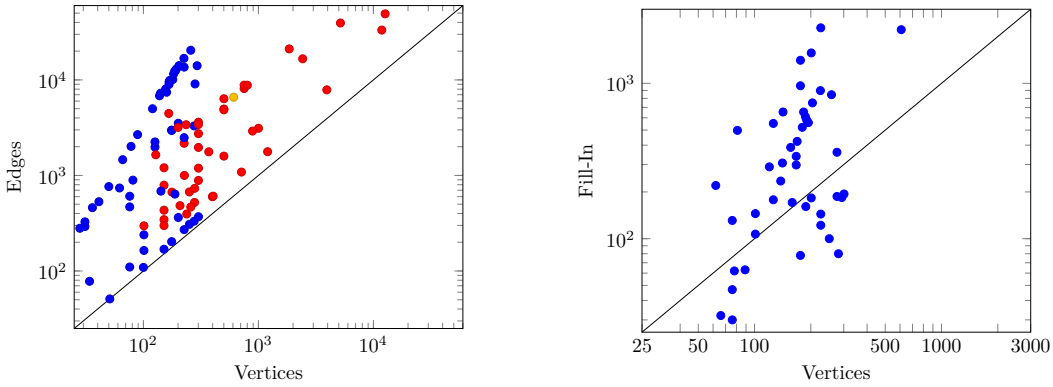



**Figure 5 Left:** Distribution of vertex and edge numbers for the hidden benchmark instances of Track B. Blue dots indicate that the instance was solved by the winning submission, the yellow dot indicates the one instance among all solved instances that was not solved by the winning submission (but by the runner-up), red dots indicate instances that have not been solved by any submission. **Right:** Instance size vs. solution size for the instances solved by the winning submission.

## Submissions

Eight implementations were submitted to the Minimum Fill-in competition of PACE 2017. Their ranking is given below, while we provide a brief explanation on the solutions of the three top-ranked submissions.

- **1st place:** Yasuaki Kobayashi (Kyoto University) and Hisao Tamaki (Meiji University). Their submission is written in Java and solved 54 of 100 instances (indeed, all but one instances that could not be solved by their submission could also not be solved by any other submission returning only optimal solutions). The submission first uses data reduction where some of the reduction rules are generalizations of rules developed by Bodlaender et al. [4]. Then the instance is decomposed using Tarjan's clique decomposition algorithm [32]. The resulting instances are solved by adapting Tamaki's modification [31] of Bouchitté and Todinca's dynamic programming algorithm [5]. The source code is available at `github.com/TCS-Meiji/PACE2017-TrackB`.
- **2nd place:** Jeremias Berg, Matti Järvisalo, and Tuukka Korhonen (University of Helsinki). Their submission is written in C++ and solved 45 of 100 instances. The general idea of the submission is to use Tarjan's clique decomposition [32], then to apply parts of the known kernelization algorithms, and finally to use dynamic programming over potential maximal cliques [5]. The source code is available at `github.com/Laakeri/PACE2017-min-fill`.
- **3rd place:** Édouard Bonnet (University Paris-Dauphine), R.B. Sandeep (Hungarian Academy of Sciences), and Florian Sikora (University Paris-Dauphine). Their submission is written in Python and solved 23 of 100 instances. The submission uses Tarjan's clique decomposition followed by kernelization and dynamic programming over potential maximal cliques [5]. The source is available at `bitbucket.org/Florian_dauphine/mfi`.
- **4th place:** Anders Wind Steffensen and Mikael Lindemann (IT University of Copenhagen). Their submission is written in Java and solved 13 out of 100 instances.
- **5th place:** Kaustubh Joglekar, Akshay Kamble, and Rajesh Pandian (Indian Institute Of Technology, Madras). Their submission is written in C++ and solved 10 of 100 instances.
- **6th place:** Saket Saurabh and Prafullkumar Tale (Institute of Mathematical Sciences,

Chennai). Their submission is written in Python and solved 19 of 100 instances. For 28 instances the provided solution was suboptimal. For 13 instances, the submission was the only one to output any solution. The suboptimal solutions are usually quite close to the optimal solutions. Hence, this submission can be viewed as a good heuristic.

- **7th place:** Mani Ghahremani (University of Portsmouth). Their submission is written in Java and solved 5 of 100 instances. For one instance, the provided solution was suboptimal.
- **8th place:** Frederik Madsen, Mikkel Gaub, and Malthe Kirkbro (IT University of Copenhagen). Their submission is written in C++ and solved 3 of 100 instances. For 3 instances, the provided solution was suboptimal.

We remark that the hidden instances were chosen to be especially challenging, hence the organizers expected that submissions would solve only a minority of the instances. Thus, solving more than 50% of the instances seems to be a remarkable contribution. All three winning contributions were faster than our naïve ILP formulation. An implementation we obtained of a more involved ILP formulation [3] was proven incorrect by the submissions; it remains to evaluate whether the error is due to the formulation itself or the scripts that generate the ILP from the graph data.

## 4 PACE organization

In September 2017, Frances Rosamond transferred the Steering Committee Chair to Bart Jansen. The Steering Committee and the PACE 2017 Program Committee are as follows.

| | | |
|---|---|---|
| **Steering committee:** | Holger Dell | Saarland University & Cluster of Excellence |
| | Thore Husfeldt | ITU Copenhagen & Lund University |
| | Bart M. P. Jansen (chair) | Eindhoven University of Technology |
| | Petteri Kaski | Aalto University |
| | Christian Komusiewicz | Friedrich-Schiller-University Jena |
| | Frances A. Rosamond | University of Bergen |
| **Track A:** | Holger Dell | Saarland University & Cluster of Excellence |
| **Track B:** | Christian Komusiewicz | Friedrich-Schiller-University Jena |
| | Nimrod Talmon | Weizmann Institute of Science |
| | Mathias Weller | Laboratory of Informatics, Robotics, and Microelectronics of Montpellier (LIRMM) |

The Program Committee of PACE 2018 will be chaired by Édouard Bonnet (Middlesex University, London) and Florian Sikora (Université Paris Dauphine).

## 5 Conclusion

As organizers, we consider the second iteration of PACE to be a huge success. We had great submissions building on existing and new theoretical ideas, which led to fast programs that performed well on the real-word inputs to which they were applied. The award ceremony at IPEC was very well attended, and many of the ALGO 2017 participants showed an interest in the competition. Tamaki [31] won a best paper award at ESA 2017 for ideas that led to his three PACE 2017 submissions, and his paper was directly inspired by PACE.

We thank all the participants for their enthusiasm and look forward to many interesting iterations of the challenge in the future. We also thank all members of the community for

their input in formulating the goals and setup of the challenge. We welcome anyone who is interested to add their name to the mailing list on the website [29] to receive PACE updates and join the discussion. In particular, plans for PACE 2018 will be posted there.

**Acknowledgments.** We are grateful to Szymon Wasik for the fruitful collaboration and for hosting the competition at `optil.io` . Prize money and travel grants were given through the generosity of Networks [28], an NWO Gravitation project of the University of Amsterdam, Eindhoven University of Technology, Leiden University, and the Center for Mathematics and Computer Science (CWI). We thank Ben Strasser for suggesting Figure 3.

## References

1 Michael Abseher, Nysret Musliu, and Stefan Woltran. htd - A free, open-source framework for (customized) tree decompositions and beyond. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, volume 10335 of *Lecture Notes in Computer Science*, pages 376–386. Springer, 2017. doi:10.1007/978-3-319-59776-8_30.

2 Max Bannach, Sebastian Berndt, and Thorsten Ehlers. Jdrasil: A modular library for computing tree decompositions. In Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman, editors, *16th International Symposium on Experimental Algorithms, SEA 2017, June 21-23, 2017, London, UK*, volume 75 of *LIPIcs*, pages 28:1–28:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.SEA.2017.28.

3 David Bergman and Arvind U. Raghunathan. A benders approach to the minimum chordal completion problem. In Laurent Michel, editor, *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*, volume 9075 of *Lecture Notes in Computer Science*, pages 47–64. Springer, 2015. doi:10.1007/978-3-319-18008-3_4.

4 Hans L. Bodlaender, Pinar Heggernes, and Yngve Villanger. Faster parameterized algorithms for minimum fill-in. *Algorithmica*, 61(4):817–838, 2011. doi:10.1007/s00453-010-9421-1.

5 Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31(1):212–232, 2001. doi:10.1137/S0097539799359683.

6 Peter Buneman. A characterisation of rigid circuit graphs. *Discrete Mathematics*, 9(3):205–212, 1974. doi:10.1016/0012-365X(74)90002-8.

7 Benjamin A. Burton, Ryan Budney, William Pettersson, et al. Regina: Software for low-dimensional topology, 1999–2016. URL: http://regina-normal.github.io.

8 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.

9 Krishnendu Chatterjee, Rasmus Ibsen-Jensen, and Andreas Pavlogiannis. Optimal reachability and a space-time tradeoff for distance queries in constant-treewidth graphs. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.28.

10 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 fuer Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.30.

11 Emmanuel J. P. Douzery, Celine Scornavacca, Jonathan Romiguier, Khalid Belkhir, Nicolas Galtier, Frédéric Delsuc, and Vincent Ranwez. OrthoMaM v8: A database of orthologous exons and coding sequences for comparative genomics in mammals. *Molecular Biology and Evolution*, 31(7):1923–1928, 2014. doi:10.1093/molbev/msu132.

12 Johannes Klaus Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Answer set solving with bounded treewidth revisited. In Marcello Balduccini and Tomi Janhunen, editors, *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings*, volume 10377 of *Lecture Notes in Computer Science*, pages 132–145. Springer, 2017. doi:10.1007/978-3-319-61660-5_13.

13 Johannes Klaus Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. DynASP2.5: Dynamic programming on tree decompositions in action. In *Proceedings of the 12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2017. doi:10.4230/LIPIcs.IPEC.2017.17.

14 Fedor V. Fomin, Dieter Kratsch, Ioan Todinca, and Yngve Villanger. Exact algorithms for treewidth and minimum fill-in. *SIAM J. Comput.*, 38(3):1058–1079, 2008. doi:10.1137/050643350.

15 Fedor V. Fomin and Yngve Villanger. Subexponential parameterized algorithm for minimum fill-in. *SIAM J. Comput.*, 42(6):2197–2216, 2013. doi:10.1137/11085390X.

16 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

17 Serge Gaspers, Joachim Gudmundsson, Mitchell Jones, Julián Mestre, and Stefan Rümmele. Turbocharging treewidth heuristics. 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 13:1–13:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.13.

18 Rob Gysel, Kristian Stevens, and Dan Gusfield. Reducing problems in unrooted tree compatibility to restricted triangulations of intersection graphs. In Benjamin J. Raphael and Jijun Tang, editors, *Algorithms in Bioinformatics - 12th International Workshop, WABI 2012, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, volume 7534 of *Lecture Notes in Computer Science*, pages 93–105. Springer, 2012. doi:10.1007/978-3-642-33122-0_8.

19 Yoichi Iwata. Linear-time kernelization for feedback vertex set. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 68:1–68:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.68.

20 Haim Kaplan, Ron Shamir, and Robert Endre Tarjan. Tractability of parameterized completion problems on chordal and interval graphs: Minimum fill-in and physical mapping. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 780–791. IEEE Computer Society, 1994. doi:10.1109/SFCS.1994.365715.

21 Kalev Kask, Andrew Gelfand, Lars Otten, and Rina Dechter. Pushing the power of stochastic greedy ordering schemes for inference in graphical models. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3771.

22 Masashi Kiyomi, Yoshio Okamoto, and Yota Otachi. On the treewidth of toroidal grids. *Discrete Applied Mathematics*, 198:303–306, 2016. doi:10.1016/j.dam.2015.06.027.

23 Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. Sat-encodings for special treewidth and pathwidth. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 429–445. Springer, 2017. doi:10.1007/978-3-319-66263-3_27.

24 Matrix market. URL: http://math.nist.gov/MatrixMarket/.

25 Multiple-alignment to character-state intersection graph converter, 2017. URL: https://github.com/PACE-challenge/phylo_converter.

26 Assaf Natanzon, Ron Shamir, and Roded Sharan. A polynomial approximation algorithm for the minimum fill-in problem. *SIAM J. Comput.*, 30(4):1067–1079, 2000. doi:10.1137/S0097539798336073.

27 Network repository. URL: http://networkrepository.com/.

28 Networks project, 2017. URL: http://www.thenetworkcenter.nl.

29 Parameterized Algorithms and Computational Experiments website, 2015–2017. URL: https://pacechallenge.wordpress.com.

30 Vincent Ranwez, Frédéric Delsuc, Sylvie Ranwez, Khalid Belkhir, Marie-Ka Tilak, and Emmanuel JP Douzery. OrthoMaM: A database of orthologous genomic markers for placental mammal phylogenetics. *BMC Evolutionary Biology*, 7(1):241, Nov 2007. Database accessed 2017 at http://orthomam.univ-montp2.fr. doi:10.1186/1471-2148-7-241.

31 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 68:1–68:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.68.

32 Robert Endre Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221–232, 1985. doi:10.1016/0012-365X(85)90051-2.

33 Mikkel Thorup. All structured programs have small tree-width and good register allocation. *Inf. Comput.*, 142(2):159–181, 1998. doi:10.1006/inco.1997.2697.

34 Tom C. van der Zanden and Hans L. Bodlaender. Computing treewidth on the GPU. In *Proceedings of the 12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2017. doi:10.4230/LIPIcs.IPEC.2017.29.

35 Rim van Wersch and Steven Kelk. Toto: An open database for computation, storage and retrieval of tree decompositions. *Discrete Applied Mathematics*, 217:389–393, 2017. doi:10.1016/j.dam.2016.09.023.

36 Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981. doi:10.1137/0602010.